Machine learning and GPU-accelerated computing applied to platelet inventory management in a hospital blood bank

Joseph Marc Farrington

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

UKRI CDT in AI-enabled Healthcare Systems

& Institute of Health Informatics

University College London

December 22, 2024

I, Joseph Marc Farrington, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Hospital blood banks must maintain an adequate stock of platelets and other blood products to ensure that patients can receive transfusions when needed, while trying to minimize wastage. Challenges include short product shelf lives, donor-recipient compatibility, and the fact that some requested units may not be transfused. I used machine learning (ML) and graphics processing unit (GPU)-accelerated computing methods to find policies for two key decisions: how many platelet units to order from a supplier (replenishment) and selecting platelet units to meet demand (issuing).

My review of the literature identified opportunities to: use deep reinforcement learning (DRL) to learn how to act in the large state spaces needed to represent perishable stock; use GPU hardware to compute the optimal replenishment policy for larger, more realistic problems; and improve issuing policies which have received less attention than replenishment decisions.

I first developed a novel reinforcement learning environment to model a simple platelet replenishment problem. DRL policies performed near-optimally on simulated data and outperformed commonly used heuristic policies on real demand trajectories from a UK hospital.

My GPU-accelerated implementation of value iteration enabled the optimal policy to be computed for perishable inventory management problems where this was recently deemed infeasible or impractical, with up to 16.8M states, using consumer-grade hardware. These results can support benchmarking approximate approaches including DRL.

I designed a novel ML-guided issuing policy to address the fact that not all

requested platelet units are transfused. I explored how the utility of the policy depended on the quality of patient-level ML predictions of transfusion and trained an ML model, with AUROC 0.74, sufficiently good to support wastage reductions under the new policy.

Finally, I extended the problem to jointly optimize replenishment and issuing policies to manage multiple perishable products with substitution, including platelets with all eight ABO/RhD blood types.

Impact statement

The findings of this thesis can help to improve the management of platelet inventory and, by extension, other perishable products.

In the more realistic problem settings, policies learned using DRL and neuroevolution often perform better than widely used heuristic rules, and could be extended to incorporate additional real-time information from electronic health records. Additionally, I have shown that using neuroevolution to fit policies enables key performance indicators (KPIs) to be incorporated directly into the optimization process. This avoids the need for invoking notional costs to balance competing objectives and supports informative visualizations of feasible trade-offs.

The novel ML-guided issuing policy, Youngest Unit for Predicted Returns (YUPR), could help to reduce platelet wastage, particularly in hospitals where the remaining useful life on arrival at the hospital blood bank is short and it is common for requested platelet units not to be transfused. Based on my findings, the transfusion laboratory at our partner site is considering adopting a YUPR policy using a rule-based system, combining insights from my ML model with expert knowledge.

Three aspects of the research have broader applications: a simulation-first approach to understanding the utility of a proposed ML model, the implementation of two widely used methods for finding policies using a high-level Python library developed for ML to take advantage of modern GPU hardware, and GPU-accelerated reinforcement learning environments for problems with multiple agents.

The simulation-first approach to predictive modelling, evaluating the utility

of a proposed prediction model using a simulated workflow prior to model development, may be useful in any setting where simulating the workflow is tractable. It could help to focus future research efforts on prediction tasks that are more likely to be deployed because the practical utility, in terms of KPIs of interest to decision makers, is demonstrated at an early stage.

The GPU-accelerated implementations of dynamic programming and simulation optimization provide a useful case study for operational research, where adoption of GPUs appears to be very limited. Access to the optimal policy for larger, more realistic problems can support better benchmarking of heuristic and approximate approaches. GPU-accelerated simulation enables many simulated episodes to be run in parallel, supporting a broader search for policy parameters and reducing the error in the estimates of performance.

The GPU-accelerated reinforcement learning environments for multi-agent problems can be used to jointly optimize the policies of multiple agents and the adapted single-agent environment may be particularly useful for problems where two agents share the same reward function and do not act in parallel or in a fixed order.

I have made my code available to support other researchers interested in solving related problems and to reduce duplicated effort. The reinforcement learning environments representing blood product inventory management workflows can be used to compare alternative optimization approaches and policies. The GPU-accelerated implementation of value iteration can be customized for alternative Markov decision processes and used to find optimal policies for a wide range of problems.

Acknowledgements

Thank you to my supervisors: Dr Kezhi (Ken) Li, Prof Martin Utley and Dr Wai Keong Wong for their guidance, encouragement, support and compassion. I've been very fortunate to work with, and learn from, each of them over the last five years. They have contributed immeasurably to both this thesis and to my development as a researcher.

I am grateful for the financial support of my funders: the UKRI Centre for Doctoral Training in AI-enabled Healthcare and the NIHR University College London Hospitals Biomedical Research Centre. Within the CDT, I would like to thank Prof Paul Taylor, Dr Kate Ricketts, Craig Smith, Victoria West, Antonia Coote, Pablo Fernandez Medina and Paula Ktorides for their help and support during a challenging time.

This work would not have been possible without the backing of the UCLH transfusion laboratory. I'm particularly grateful to Dr Samah Alimam for her enthusiastic support of both me and this work and for providing opportunities to shadow and present to the team. Thank you to Ian Longair for his time during my shadowing visit, his help with understanding the data and processes, and for his assistance in obtaining data. I am also grateful for additional help from Lubna Awas, Zeynab Jeewa, and Philip Russell.

I would like to thank several individuals at UCLH beyond the transfusion laboratory: Dr Sara Trompeter, Eirini Tsitsipa and Nathan Lea for their help in obtaining ethical approval for the project; Dr Roma Klapaukh, Saidul Haque, Dave Ramlakhan, Richard Clarke and Duncan Cartner for their assistance with identifying, extracting, and hosting hospital data; and Dr Steve Harris for

introducing me to the EMAP platform and for kindly agreeing to supervise my honorary contract. I'm also very grateful to the UCLH patients whose anonymized data was crucial for several parts of this thesis.

I've been fortunate to work with the NIHR Blood Transfusion Research Unit in Data Driven Transfusion Practice during this project. I am especially grateful to Dr Suzanne Maynard for her collaborative efforts on our review paper, I thoroughly enjoyed working together. My sincere thanks also go to Prof Simon Stanworth and Dr Hayley Evans for their interest in, and support of, the review paper. Additionally, I extend my appreciation to them and the wider group, including Prof Mike Murphy, Dr Samantha Warnakulasuriya, Mr Antony Palmer, Dr Paula Dhiman, Dr Akshay Shah, Dr Alwyn Kotze, and Linda von Nerée, for their thoughtful feedback on my work and for introducing me to the broader research environment at other universities and within the NHS.

I'm grateful to Prof Eligius Hendrix and Dr Mahdi Mirjalili for providing additional material that enabled me to test my implementation of the scenarios from their work in Chapter 3. Any errors or differences are my responsibility alone. I would also like to thank Dr Thomas Monks for sharing his expertise on simulation optimization during the preliminary stages of the work on Chapter 3.

Thank you to Dr Jonathan Loh for his generous assistance in proof-reading this thesis and for his thoughtful comments; to Dr Zella King for being an excellent sounding board for research ideas and providing an inspiring example by getting a model into daily use at UCLH; and to my conference buddy Amalia Gjerloev for excellent company all around Europe.

I am grateful for the support of my family, particularly my parents. I believe my Dad would have been proud that I've reached this milestone, even if he was initially bemused when I left my job. My Mum has been, as always, an endless source of love and support and I couldn't have done it without her. I would also like to thank my great uncle, Kevin Lavery, whose generosity enabled me to take a career break and pursue my MSc.

Thank you to my fellow CDT students, particularly my cohort (Abi, Andre,

Anthony, Dominic, Peter and Roy) and the office/social regulars (Adrito, Chris, Eva, Florence, James, Jamie, Maria, Nikita, Simon, Tina, Tom, and Vicky) for conversations worth commuting for, an epic game of Sardines, and many valiant-but-doomed attempts to win the Institute pub quiz.

Finally, I owe a huge thank you to my friends for their encouragement, excellent company, and for getting me out of the house and my head when I needed it. Particular thanks to Mel and the small ones, Katherine D, Richard S & Audri, Fiona, Jon & Mona, Ravi, Tom S & Jess, Yang & Trav, Sinead, Sophy, Natasha, Sabrina, Rob & Katherine P, Carl & Kyle, Felix & Nancy, Richard G, Eystein & Edith, Charles & Cat and Tom H & Lucy. I'm lucky to know you all.

UCL Research Paper Declarations

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

1. **1. For a research manuscript that has already been published** (if not yet published, please skip to section 2)**:**

   (a) **What is the title of the manuscript?**

   Machine learning in transfusion medicine: A scoping review

   (b) **Please include a link to or doi for the work:**

   https://doi.org/10.1111/trf.17582

   (c) **Where was the work published?**

   Transfusion

   (d) **Who published the work?**

   Wiley

   (e) **When was the work published?**

   November 2023

   (f) **List the manuscript's authors in the order they appear on the publication:**

   Suzanne Maynard*, Joseph Farrington*, Samah Alimam, Hayley Evans, Kezhi Li, Wai Keong Wong, Simon J. Stanworth

   (g) **Was the work peer reviewd?**

   Yes

   (h) **Have you retained the copyright?**

Yes

(i) **Was an earlier form of the manuscript uploaded to a preprint server (e.g. medRxiv)? If 'Yes', please give a link or doi**

No

If 'No', please seek permission from the relevant publisher and check the box next to the below statement:

☒ *I acknowledge permission of the publisher named under 1d to include in this thesis portions of the publication named as included in 1c.*

2. **For a research manuscript prepared for publication but that has not yet been published** (if already published, please skip to section 3)**:**

   (a) **What is the current title of the manuscript?**

   (b) **Has the manuscript been uploaded to a preprint server e.g. medRxiv?**

   If 'Yes', please please give a link or doi:

   (c) **Where is the work intended to be published?**

   (d) **List the manuscript's authors in the intended authorship order:**

   (e) **Stage of publication:**

3. **For multi-authored work, please give a statement of contribution covering all authors** (if single-author, please skip to section 4)**:**

   SM and JF jointly performed the literature review and data extraction, compiled the figures and tables and wrote the manuscript. SS was responsible for the initial study concept. SS and WKW defined the topic, reviewed and edited the manuscript. SA, HE, and KL reviewed and edited the manuscript.

4. **In which chapter(s) of your thesis can this material be found?**

   Chapter 2

e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/ senior author unless this is not appropriate, e.g. if the paper was a single-author work)**:**

Candidate:

J M Farrington

Date: 17 June 2024

Supervisor/Senior Author signature (where appropriate)**:**

K Li

Date: 17 June 2024

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

1. **1. For a research manuscript that has already been published** (if not yet published, please skip to section 2)**:**

   (a) **What is the title of the manuscript?**

   Deep Reinforcement Learning for Managing Platelets in a Hospital Blood Bank

   (b) **Please include a link to or doi for the work:**

   https://doi.org/10.1182/blood-2023-178306

   (c) **Where was the work published?**

   Blood

   (d) **Who published the work?**

   Elsevier

   (e) **When was the work published?**

   November 2023

   (f) **List the manuscript's authors in the order they appear on the publication:**

   Joseph Farrington, Martin Utley, Samah Alimam, Wai Keong Wong, Kezhi Li

   (g) **Was the work peer reviewd?**

   No

   (h) **Have you retained the copyright?**

   No

   (i) **Was an earlier form of the manuscript uploaded to a preprint server (e.g. medRxiv)? If 'Yes', please give a link or doi**

   No

   If 'No', please seek permission from the relevant publisher and check the box next to the below statement:

☒ *I acknowledge permission of the publisher named under 1d to include in this thesis portions of the publication named as included in 1c.*

2. **For a research manuscript prepared for publication but that has not yet been published** (if already published, please skip to section 3)**:**

   (a) **What is the current title of the manuscript?**

   (b) **Has the manuscript been uploaded to a preprint server e.g. medRxiv?**

   If 'Yes', please please give a link or doi:

   (c) **Where is the work intended to be published?**

   (d) **List the manuscript's authors in the intended authorship order:**

   (e) **Stage of publication:**

3. **For multi-authored work, please give a statement of contribution covering all authors** (if single-author, please skip to section 4)**:**

   JF: Conceptualization, Methodology, Software, Investigation, Writing - Original Draft, Visualization. **SA:** Writing - Review & Editing. **MU:** Conceptualization, Methodology, Writing - Review & Editing, Supervision. **KL:** Conceptualization, Methodology, Writing - Review & Editing, Supervision, Project administration, Funding acquisition. **WKW:** Conceptualization, Resources, Supervision, Writing - Review & Editing, Funding acquisition.

4. **In which chapter(s) of your thesis can this material be found?**

   Chapter 3

e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/ senior author unless this is not appropriate, e.g. if the paper was a single-author work)**:**

Candidate:

J M Farrington

Date: 17 June 2024

Supervisor/Senior Author signature (where appropriate)**:**

K Li

Date: 17 June 2024

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

1. **1. For a research manuscript that has already been published** (if not yet published, please skip to section 2)**:**

   (a) **What is the title of the manuscript?**

   (b) **Please include a link to or doi for the work:**

   (c) **Where was the work published?**

   (d) **Who published the work?**

   (e) **When was the work published?**

   (f) **List the manuscript's authors in the order they appear on the publication:**

   (g) **Was the work peer reviewd?**

   (h) **Have you retained the copyright?**

   (i) **Was an earlier form of the manuscript uploaded to a preprint server (e.g. medRxiv)? If 'Yes', please give a link or doi**

   If 'No', please seek permission from the relevant publisher and check the box next to the below statement:

   □ *I acknowledge permission of the publisher named under 1d to include in this thesis portions of the publication named as included in 1c.*

2. **For a research manuscript prepared for publication but that has not yet been published** (if already published, please skip to section 3)**:**

   (a) **What is the current title of the manuscript?**

   Going faster to see further: GPU-accelerated value iteration and simulation for perishable inventory control using JAX

   (b) **Has the manuscript been uploaded to a preprint server e.g. medRxiv?**

If 'Yes', please please give a link or doi:

https://arxiv.org/abs/2303.10672

(c) **Where is the work intended to be published?**

Annals of Operations Research

(d) **List the manuscript's authors in the intended authorship order:**

Joseph Farrington, Kezhi Li, Wai Keong Wong, Martin Utley

(e) **Stage of publication:** Under review

3. **For multi-authored work, please give a statement of contribution covering all authors** (if single-author, please skip to section 4)**:**

JF: Conceptualization, Methodology, Software, Investigation, Validation, Writing - Original Draft. **KL:** Supervision, Writing - Review & Editing. **WKW:** Supervision. **MU:** Investigation, Supervision, Writing - Review & Editing.

4. **In which chapter(s) of your thesis can this material be found?**

Chapter 4

e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/ senior author unless this is not appropriate, e.g. if the paper was a single-author work)**:**

Candidate:

J M Farrington

Date: 17 June 2024

Supervisor/Senior Author signature (where appropriate)**:**

K Li

Date: 17 June 2024

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

1. **1. For a research manuscript that has already been published** (if not yet published, please skip to section 2)**:**

   (a) **What is the title of the manuscript?**

   (b) **Please include a link to or doi for the work:**

   (c) **Where was the work published?**

   (d) **Who published the work?**

   (e) **When was the work published?**

   (f) **List the manuscript's authors in the order they appear on the publication:**

   (g) **Was the work peer reviewd?**

   (h) **Have you retained the copyright?**

   (i) **Was an earlier form of the manuscript uploaded to a preprint server (e.g. medRxiv)? If 'Yes', please give a link or doi**

   If 'No', please seek permission from the relevant publisher and check the box next to the below statement:

   □ *I acknowledge permission of the publisher named under 1d to include in this thesis portions of the publication named as included in 1c.*

2. **For a research manuscript prepared for publication but that has not yet been published** (if already published, please skip to section 3)**:**

   (a) **What is the current title of the manuscript?**

   Many happy returns: machine learning to support platelet issuing and waste reduction in hospital blood banks

   (b) **Has the manuscript been uploaded to a preprint server e.g. medRxiv?**

If 'Yes', please please give a link or doi:

No

(c) **Where is the work intended to be published?**

npj Digital Medicine

(d) **List the manuscript's authors in the intended authorship order:**

Joseph Farrington, Samah Alimam, Martin Utley, Kezhi Li*, Wai Keong Wong*

(e) **Stage of publication:** In revision

3. **For multi-authored work, please give a statement of contribution covering all authors** (if single-author, please skip to section 4)**:**

JF: Conceptualization, Methodology, Software, Investigation, Writing - Original Draft, Visualization. **SA:** Writing - Review & Editing. **MU:** Conceptualization, Methodology, Writing - Review & Editing, Visualization, Supervision. **KL:** Conceptualization, Methodology, Writing - Review & Editing, Supervision, Project administration, Funding acquisition. **WKW:** Conceptualization, Resources, Supervision, Writing - Review & Editing, Funding acquisition.

4. **In which chapter(s) of your thesis can this material be found?**

Chapter 5

e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/ senior author unless this is not appropriate, e.g. if the paper was a single-author work)**:**

Candidate:

J M Farrington

Date: 17 June 2024

Supervisor/Senior Author signature (where appropriate)**:**

K Li

Date: 17 June 2024

Contents

Glossary

A2C Advantage Actor-Critic

A3C Asynchronous Advantage Actor-Critic

ADP Approximate dynamic programming

AEC Agent-environment cycle

API Application programming interface

AUROC Area under the Receiver Operating Characteristic curve

CPU Central processing unit

CSV Comma separated value

CUDA Compute Unified Device Architecture

DQN Deep Q-Network

DRL Deep reinforcement learning

DSH Data Safe Haven

EHR Electronic health record

FIFO First-in First-out

FPGA Field Programmable Gate Array

GAE Generalized Advantage Estimation

GPU Graphics processing unit

IPPO Independent PPO

JIT Just-in-time

JPAC Joint UK Blood Transfusion and Tissue Transplantation Services Professional Advisory Committee

KPI Key performance indicator

LASSO Least Absolute Shrinkage and Selection Operator

LIFO Last-in First-out

MAPE Mean absolute percentage error

MDP Markov decision process

ML Machine learning

NHS National Health Service

NHSBT NHS Blood and Transplant

NIHR National Institute for Health and Care Research

NSGA-II Non-dominated Sorting Genetic Algorithm-II

OBOS Online Blood Ordering System

OUFO Oldest Unit First-out

pmap Parallel map

POMDP Partially observable Markov decision process

POSG Partially observable stochastic game

PPM Perfect predictive model

PPO Proximal Policy Optimization

PRBCs Packed red blood cells

RAM Random-access memory

ReLU Rectified linear unit

RL Reinforcement learning

ROC Receiver Operating Characteristic

SARSA State-action-reward-state-action

SHAP Shapley Additive Explanations

TD Temporal difference

TPE Tree-structured Parzen Estimator

TRPO Trust Region Policy Optimization

UCL University College London

UCLH University College London Hospitals NHS Foundation Trust

UK United Kingdom

USA United States of America

vmap Vectorizing map

YUFO Youngest Unit First-out

YUPR Youngest Unit for Predicted Returns

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem statement

Hospital blood banks need to maintain an adequate stock of products, including packed red blood cells (PRBCs) and platelets, to meet demand for transfusions while avoiding waste. Shortages may put patients at risk, for example by delaying a necessary procedure. In addition to the financial loss caused by waste, there is also a moral component: there is a responsibility for the blood supply chain to make the best use of products that have been altruistically donated and high wastage may reduce donor confidence and affect future donation rates. Determining adequate stock levels, and the orders required to maintain them, is complicated by the perishable nature of blood products, the requirement for compatibility between blood donors and recipients, and the fact that not all products requested for patients are subsequently transfused. A hospital blood bank must therefore hold an appropriate mix of products but cannot hold large quantities 'just in case' because that would lead to high levels of waste.

In this thesis I concentrated on platelet inventory management due to their short shelf life and high reported wastage rates. I investigated how machine learning (ML) and graphics processing unit (GPU)-accelerated computing can be used to support two key decisions: placing a replenishment order with the supplier and selecting units from stock to issue to a patient.

1.2 Motivation

A systematic review published in 2020 found that the common range for platelet wastage rates in hospitals was 10-20% [1]. The United Kingdom (UK) Blood Stocks Management Scheme reported that 4.5% of platelet units issued to hospitals in England were wasted at a cost of £2M in 2017/2018 [2], with a higher wastage rate of 4.8% (11,758 units reported as wasted, of 247,416 units issued) in 2022/2023 [3]. Additionally, in an effort to avoid wastage, it is common for patients to receive platelets from a donor whose blood type is not an exact match [4, 5] which, in some cases, can put them at risk of complications or make the transfusion less effective. Finding improved policies (functions that map observations of the world to the actions that should be taken) for platelet replenishment and issuing could help to reduce wastage while ensuring that the right unit is available for the right patient at the right time.

Blood product inventory management is an example of a perishable inventory problem. Perishable inventory problems have large state spaces, the number of possible states the system can occupy, due to the need to represent the age profile of the stock. Dynamic programming methods, such as value iteration, can be used to compute the optimal policy but do not scale well to large state spaces and are therefore difficult to apply to perishable inventory problems [6]. Deep reinforcement learning (DRL) methods offer an approximate counterpart to value iteration and have been specifically designed to learn how to act in large state spaces using the function approximation capabilities of deep neural networks [7]. A small number of studies have investigated the potential benefits of DRL methods for managing perishable inventory, but none have specifically considered inventory management in a hospital blood bank. Replenishment policies based on neural networks can represent complicated functions and may be able to achieve better performance than widely-used heuristic ordering rules by taking more of the available information into account.

The understanding in the operational research literature about the size of perishable inventory problems for which value iteration is feasible and practical

does not appear to reflect recent advances in computer hardware, in particular GPUs and the software libraries that make general purpose computing on GPUs accessible to researchers without specialist knowledge. GPUs were developed for rendering computer graphics, which requires the same operations to be efficiently applied to many inputs in parallel. Compared to a central processing unit (CPU), GPUs therefore have many more, albeit individually less powerful, cores. GPU-acceleration refers to offloading computationally intensive tasks that benefit from large-scale parallelization from the CPU to the GPU. Value iteration is well suited to parallel processing using this model. The speed increases available by running value iteration in parallel on GPUs may open up the possibility of computing the optimal policies for larger, more realistic problems where it is currently considered impractical or infeasible. Such policies could be useful both for solving a problem directly and for benchmarking heuristic and other approximate approaches.

My discussions with staff at our partner hospital revealed that not all platelets requested by clinical teams are subsequently transfused and that, due to the short useful life, this could lead to wastage when following a standard issuing policy that issues the oldest unit in stock. It may be possible to use ML to predict, for a given request, whether the patient will receive a transfusion. This prediction could in turn be used to support an alternative issuing policy and potentially reduce wastage.

Finally, while it is common in research to only consider a single type of platelet unit, UK and other hospital blood banks specify order quantities for different types of platelet units, including by blood type. Incorporating this aspect of the real problems brings the policy recommendations closer to real-life decisions. When managing multiple products, with complex substitution relationships, it may be beneficial to jointly optimize the replenishment and issuing policies. By modelling the task as a multi-agent problem, the two policies can be jointly optimized and extracted to be applied in practice.

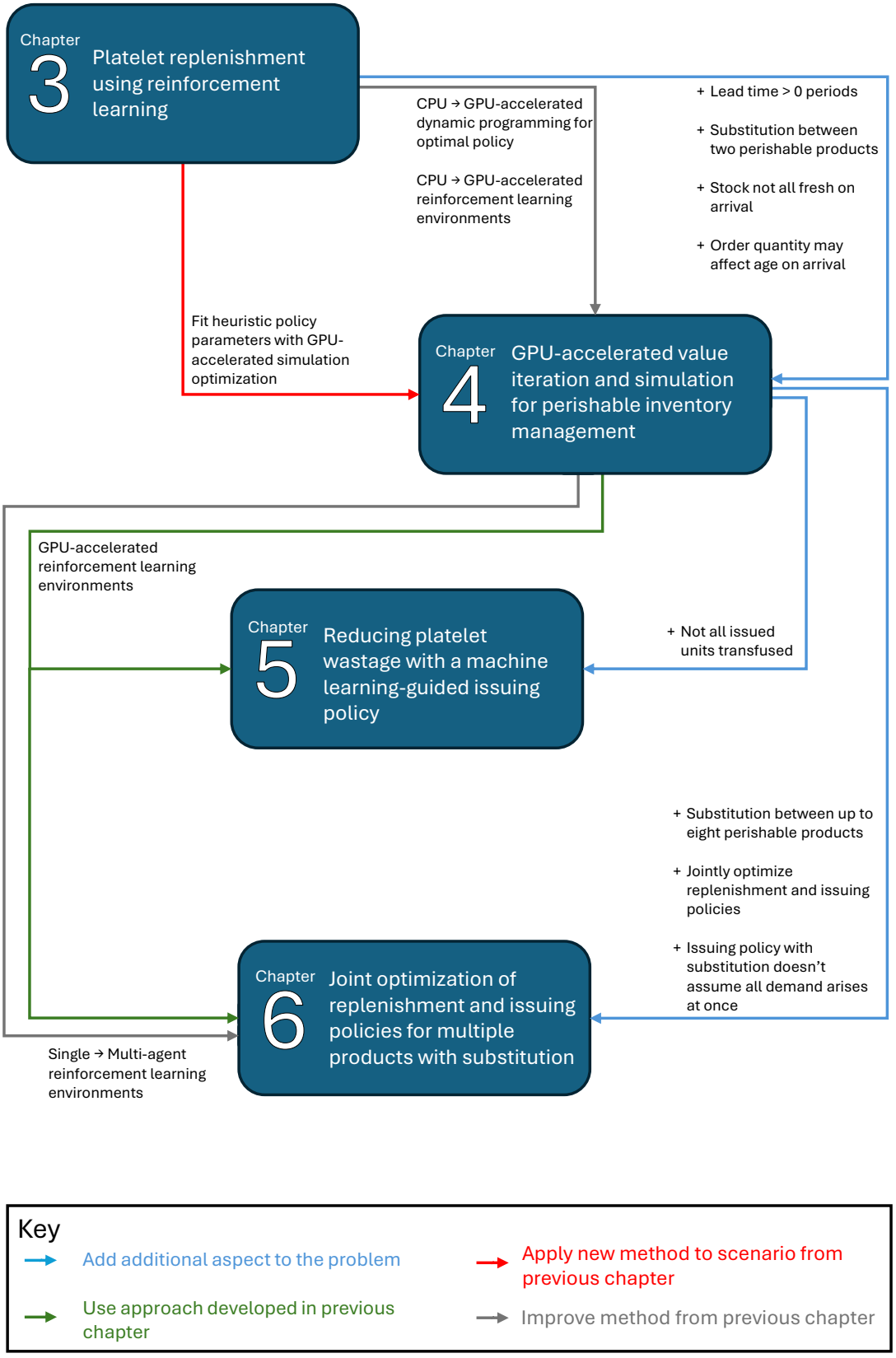


Figure 1.1: Overview of the connections between chapters of this thesis.

The connections between the experimental chapters of my thesis are illustrated in Figure 1.1. Each chapter addresses one of the supplemental research questions set out in Section 1.3 below, and begins by setting out a more detailed motivation for the work described within.

1.3 Research questions

The primary research question for this thesis is: can machine learning and GPU-accelerated computing be used to find improved policies for platelet replenishment and issuing in a hospital blood bank?

In support of this research question, I have investigated the following supplementary research questions:

1. Can DRL methods be used to learn effective policies for platelet replenishment?
2. Do recent advancements in GPU hardware and software increase the size of perishable inventory replenishment problems for which it is practical and feasible to find the optimal policy?
3. How can hospital blood banks modify their platelet issuing policies to account for discrepancies between clinician requests and subsequent transfusions?
4. Can the management of multiple products, with the potential for substitution between them, be improved by modelling the task as a multi-agent problem and jointly optimizing the replenishment and issuing policies?

1.4 Thesis outline

Chapter 2: Background and related work

In Chapter 2 I set out the background to this work including the blood supply chain, blood product inventory management, and reinforcement learning (RL). I also summarize the results of a scoping review on the applications of ML in transfusion medicine, including the management of hospital blood bank inventory, that I co-authored.

Chapter 3: Platelet replenishment using reinforcement learning

In Chapter 3 I investigate whether DRL methods can be used to learn effective policies for platelet replenishment. I implemented an RL environment based on a platelet replenishment scenario from the operational research literature and evaluated the performance of DRL methods using both simulated data and real demand data from our partner hospital site University College London Hospitals NHS Foundation Trust (UCLH).

Chapter 4: GPU-accelerated value iteration and simulation for perishable inventory management

In Chapter 4 I explore whether recent advances in GPU hardware and software increase the size of perishable inventory replenishment problems for which it is practical and feasible to compute the optimal policy. I developed a GPU-accelerated implementation of value iteration and demonstrated that it could be used to find the optimal replenishment policy for larger, more complex problem settings on three perishable inventory replenishment scenarios from the recent literature. Additionally, I developed GPU-accelerated RL environments of the perishable inventory scenarios, and used them to efficiently fit the parameters of heuristic replenishment policies using simulation optimization.

Chapter 5: Reducing platelet wastage with a machine learning-guided issuing policy

In Chapter 5 I address how hospital blood banks can modify their platelet issuing policies to reduce wastage when not all issued platelets are transfused. I proposed and evaluated an ML-guided issuing policy based on an observed challenge at our partner hospital. I used a simulated workflow to estimate the potential benefits of the issuing policy using key performance indicator (KPIs) and, based on these results, trained and evaluated a supervised machine learning model to predict whether requests for platelet units would result in a transfusion.

Chapter 6: Joint optimization of replenishment and issuing policies for multiple products with substitution

In Chapter 6 I investigate whether the management of multiple products, with the potential for substitution between them, can be improved by modelling the task as a multi-agent problem and jointly optimizing the replenishment and issuing policies. I developed two types of GPU-accelerated RL environments that supported distinct agents making replenishment and issuing decisions: a full multi-agent environment and, for computational reasons, an adapted single-agent environment. I used these environments to model perishable inventory management workflows, including managing platelet inventory while considering blood type compatibility requirements. I compared the performance of heuristic and neural network policies for both decisions, including neural network policies that were jointly fit using multi-agent RL and neuroevolutionary methods.

Chapter 7: Conclusion

In Chapter 7 I summarize the findings, strengths, and weaknesses of the thesis and discuss possible avenues for further research.

Appendices

In the appendices I set out additional supporting material, including a description of the inventory management processes at UCLH and supporting descriptive analysis of data from the hospital transfusion laboratory in Appendix B.

1.5 Ethical approval

This project, under the name 'TransfuseAI', was sponsored by the UCLH/University College London (UCL) Joint Research Office and received ethical approval from the National Health Service (NHS) Health Research Authority on 1 March 2021 (IRAS project ID 290615).

In line with the requirements of the application, all of the work with pseudonymized patient-level data was conducted on the UCLH Data Science

Desktop, a secure research computing environment hosted by the hospital.

1.6 Awards and honours

I was awarded the best poster prize at the UKRI AI CDTs in Healthcare Conference in May 2022 and second place for the Gallivan Award at the 48th annual meeting of the EURO Working Group on Operational Research Applied to Health Services (ORAHS) in July 2022 for posters based on the work in Chapter 3.

In June 2024 I collected the award for "Healthcare partnership of the year" at the London Higher Awards for work led by Dr Zella King at the UCL Clinical Operational Research Unit. The award recognized the partnership between UCL and UCLH, specifically the development and implementation of a machine learning-based system to support hospital bed planners by predicting admissions from the emergency department. I contributed to the development and evaluation of the initial machine learning models as reported in King *et al.* [8].

Chapter 2

Background and related work

In this chapter I discuss the background to my work including the blood supply chain, blood product inventory management, and RL.

2.1 The blood supply chain

The blood supply chain encompasses all of the processes required to go from collecting a donation to transfusing a recipient. In a recent review of quantitative modelling techniques applied to the blood supply chain, Osorio *et al.* [9] described four main echelons:

- **Collection**: the process of collecting whole blood and blood components from donors and transporting the collected products to a regional blood centre.
- **Production**: the process of testing the collected blood and, where necessary, fractionating whole blood into its component parts.
- **Storage and inventory**: the process of managing blood product inventory at a hospital blood bank: placing replenishment orders with a regional blood centre, transportation of blood products from the regional blood centre to the hospital blood bank, and issuing units to meet patient demand.
- **Distribution**: the process of transporting blood units from a hospital blood bank to a care unit, where they may be transfused to a patient.

The blood supply chains of different countries and regions have different levels of centralization. England has a highly centralized system in which all collected blood is processed in one of three manufacturing centres, and the blood components are then distributed to Stock Holding Units to fulfil orders from hospital blood banks [10]. In this case, therefore, the regional blood centre in the echelons described by Osorio *et al.* [9] is two different sites: the manufacturing centre and the Stock Holding Unit. In contrast, the Stanford Blood Centre in California in the United States of America (USA) performs the collection and production steps, and supplies a single hospital blood bank serving two associated hospitals with almost all of its required blood products [11].

This study concentrates on the third echelon described by Osorio *et al.* [9]: the management of inventory in a hospital blood bank.

In this work, unless stated otherwise, I follow the naming conventions of Prastacos [12], and consider a "regional blood centre" to receive products from donation sites and fulfil orders from hospital blood banks, and consider "hospital blood banks" to receive products from a "regional blood centre" and issue units for specific patients at the request of their care teams.

2.2 Platelets

A wide range of components can be extracted from donated whole blood, but the three major products in the blood supply chain are PRBCs, platelets and plasma. Platelets can also be collected separately from donors by plateletpheresis [13, p.19]. In this work I focus on platelets due to their relatively short useful lives after donation. However, much of the relevant work on blood product inventory management has focused on PRBCs and therefore I provide a description of PRBCs in Appendix A.

Platelets are required for the formation of blood clots. Patients may receive a platelet transfusion to either stop or prevent bleeding [14]. A common reason for transfusing platelets is to prevent bleeding in patients with a low platelet count (thrombocytopenia) who are receiving chemotherapy or undergoing a

haematopoietic stem cell transplant [15]. There are multiple potential causes of thrombocytopenia and both the underlying cause and the likelihood of bleeding (e.g., in patients undergoing surgical procedures) are important considerations when deciding whether to give patients a platelet transfusion [15].

The total demand for platelets from English hospitals in 2022/23 was 247k [3]. Reported wastage as a percentage of units issued to hospital blood banks was 4.8% and the two leading causes of wastage were units that units that were medically ordered but not used and time expiry [3]. As of June 2024, a single standard unit of platelets in England cost £246 [16].

In UK, platelet units have a useful life of five days from donation, which can be extended to seven days with automated bacterial screening, if kept under temperature control in an agitator [13, p.21]. Until recently these screening processes were not approved for use in the USA and the United States Food and Drug Administration limited storage to five days after donation [17, 18]. Platelets may be irradiated to inactivate lymphocytes, a type of white blood cell, in the donated unit which might otherwise cause transfusion associated graft-versus-host disease [13, p.156]. Unlike PRBCs, irradiation does not reduce the remaining useful life of platelet units.

A key consideration when issuing a unit to be transfused is whether the donor's blood type is compatible with the recipient's blood type. Blood groups are determined by the presence or absence of molecules on the surface of red blood cells: blood group antigens. While there are over 300 human blood groups, there are two blood group systems that are particularly important in clinical practice: ABO and Rh [13, p.7].

In the ABO system a patient's red blood cells (and other tissue) may express the A antigen (type A), the B antigen (type B), both (type AB) or neither (Type O). Most people have antibodies for the ABO antigens not found on their own red blood cells. I set out the compatible combinations of patient and donor ABO blood groups in Table 2.1 [13, p. 8, 19, p. 64]. A "major" ABO incompatibility occurs when the donor's ABO antigens are incompatible with the recipient's antibodies

and is the main considerations for PRBC transfusions - the response of these antibodies to incompatible transfused RBCs can be fatal. [13, p.8]. A “minor” ABO incompatibility is the opposite: the recipient’s ABO antigens are incompatible with the donors antibodies [20] and is the main concern for plasma-containing products.

There are multiple antigens in the Rh blood group system, but the presence (RhD+) or absence (RhD-) of the D antigen is the most clinically significant. The development of anti-D antibodies can lead to dangerous transfusion reactions, and can also be dangerous to an RhD+ fetus carried by an RhD- mother.

A person’s ABO/RhD blood type is determined by their ABO and RhD blood group antigens. When considering compatibility between a donor and recipient for platelet transfusion, the main concern is donor antibodies in the plasma included in the prepared unit of platelets - a “minor” ABO incompatibility. Anti-A and anti-B antibodies may cause a dangerous reaction with the recipient’s own red blood cells [13, p.8]. “Major” ABO incompatible platelet transfusions are relatively common but may result in smaller improvements in platelet count and therefore a shorter interval between transfusions [20].

Platelet units may also contain fragments of red blood cells, and therefore there is a risk of the development of anti-D antibodies if RhD- patients receive a platelet transfusion from an RhD+ donor [20].

Recipient ABO blood group	Antigens on RBCs	“Major” compatible donor blood groups	Antibodies in recipient plasma	“Minor” compatible donor blood groups
O	-	O	anti-A, anti-B	O, A, B, AB
A	A	A, O	anti-B	A, AB
B	B	B, O	anti-A	B, AB
AB	A, B	AB, A, B, O	-	AB

Table 2.1: ABO antigens, antibodies and compatible donor blood groups

2.3 Machine learning applications in transfusion medicine

There has been a growing interest in the application of ML across healthcare but, at the start of my work on this thesis, there were no reviews of the literature that provided an overview of ML research applied to challenges in transfusion medicine,

including the inventory management of blood products. I conducted a scoping review jointly with Dr Suzanne Maynard, a haematology registrar and DPhil student at the University of Oxford, which was published in *Transfusion* [21]. I performed the initial database search, based on discussions with the senior authors. Suzanne and I reviewed the titles and abstracts for relevance in duplicate, read each of the potentially relevant papers and extracted the data in duplicate, and jointly drafted the manuscript.

We defined eligible studies as those where blood transfusion in humans (or the support of transfusion) was the main outcome. We excluded research applying linear or logistic regression primarily for statistical inference and/or to construct a predictive risk score. A similar restriction was applied in a recent systematic review on the impact of ML on patient care [22]. When considering inventory management in a hospital blood bank, we concentrated on recent studies using patient-level or aggregated features from electronic health records (EHRs) and, therefore, excluded research predicting demand based on historical demand alone.

We searched the Clarivate Web of Science database on January 4, 2023 with the query:

> [TS = (machine learning OR artificial intelligence OR forecast* OR algorithm OR prediction model OR predictive model OR neural network)] AND TS = {transfus* OR blood product OR blood bank OR [reaction NEAR (blood OR transfus*)]}

Additional relevant studies were identified through a review of lists of publications in the literature, consultation with the other authors, and a review of the reference lists of the studies identified as relevant from the Web of Science search.

The Web of Science search returned 4,504 publications. Initial title and abstract screening identified 107 potentially relevant studies, and 91 of these papers were selected for inclusion after full-text review. Two additional papers were identified by citation review, for a total of 93 studies. Three of the 107 studies were excluded because the same work had been published in different journals aimed

at different audiences. The publication from a journal with a medical focus was selected where possible in these cases.

We assigned each of the 93 studies to a clinical category, based on our understanding of the literature and refined after the initial screening process. For each paper, we extracted details of the clinical application area, sources of data, ML methods, and a set of pre-defined factors agreed by all the authors as important to understanding the methodology, strengths and limitations of studies applying ML to healthcare problems.

More than half of the studies (52/93) were published between 2020 and 2022, supporting our understanding that there is a growing research interest in this area. The most common application area was the prediction of transfusion (58%), followed by transfusion safety (22%), inventory management in a hospital blood bank (10%), and supporting transfusion decisions (10%). Surgery was the most common setting for the studies predicting transfusion (33/54), with trauma the second most common (13/54). The remaining eight studies applied ML to predict transfusion in the settings of obstetrics, gastrointestinal bleeding, and haemato/oncology and, in three studies, to a broader set of patients (all inpatients and intensive care).

The main source of data used for the studies was EHR systems (70% of studies), which we defined broadly as systems storing routinely collected data, as opposed to data specifically collected for research or audit purposes. Alternative sources of data included research databases and primary research data collected from laboratory work. Supervised learning (learning a function that maps input features to a target output based on labelled examples) was the most frequently applied ML approach, used in 98% of the studies (91/93), while four (4%) used unsupervised learning (identifying patterns in unlabelled data) and none used RL (see Section 2.5). The majority of studies evaluated their models on historical data, with only eight studies (9%) reporting deployment or prospective evaluation. Four of these eight studies applied their models on prospectively collected data, one reported a “shadow test” in which predictions were made in real time but were not

made available decision makers, and three reported results for a live deployment of their models. Most publications (65%) compared the results of the ML models to some benchmark approach, such as logistic or linear regression, a previously published method, or current practice.

We did not conduct a meta-analysis of the results due the wide range of problems considered, the heterogeneity of the methods and the limited use of recognized reporting frameworks. Of the studies making individual level predictions, only 11/54 papers predicting transfusions, and none of the studies predicting transfusion reactions or adverse events, reported their results in line with a reporting framework. These factors made synthesizing the results of the studies challenging and, alongside the limited number of studies that stated that their research data (28%) and code (12%) was available, may limit the efforts of future researchers to validate the models and build upon existing research. The development of common task definitions, use of established frameworks to report results and, when possible, making data and code available to other researchers would make it more straightforward to compare the performance of different methods and identify promising candidates for expensive prospective validation studies.

While only 10% (9/93) of the studies related directly to the management of blood product inventory in a hospital blood bank, the other application areas may have indirect effects on the management of blood product inventory in the future. For example, the ability to more accurately predict the need for transfusion for individuals in advance, or predict the benefit of a transfusion for an individual patient, may alter how often, and how far in advance, clinical teams request blood products. Additionally, the ability to non-destructively determine the haemoglobin content and iron content of PRBC units [23, 24] may change how stock is ordered, labelled and managed to support more precise dosing.

Seven of the nine studies applying ML to hospital blood bank management focused on supporting ordering decisions with demand forecasting. These studies are discussed in detail below in Section 2.4.3. The two other studies directly applied

ML methods in an effort to reduce wastage: directly predicting PRBC units likely to be discarded [25] and using unsupervised learning to identify product movement patterns identified with wastage [26].

Li *et al.* [27] concurrently reviewed the application of novel computational techniques, including ML, to blood demand forecasting and supply management. In addition to most of the studies we identified concerning hospital blood bank inventory management, this review also covered work on other aspects of the blood supply chain including blood donation centres and the distribution of a scarce blood product for randomized controlled trials.

2.4 Blood product inventory management

Blood products are a classic example of perishable inventory: items which "*undergo change in storage so that in time they may become partially or entirely unfit for consumption*" [6]. Many of the early theoretical studies of perishable inventory were based on blood product inventory management, which Nahmias [6] suggested may have been due to public funds supporting the work, and easier access to data from non-profit making bodies (compared to, for example, food retailers who face related challenges). For readers who are unfamiliar with inventory management problems and the associated terminology I recommend Chapters 3 and 4 of Snyder and Shen [28] for a general introduction and Chaudhary *et al.* [29] and Nahmias [30] for more focused coverage of perishable inventory management.

The earliest academic work on managing blood product inventory appears to be Elston [31], but I have not been able to locate a copy. It is referenced in a related piece of work by Elston and Pickrel [32]. A 2012 review covering the whole blood supply chain [33] identified two major periods of research in this area: the first between the mid-1970s and the mid-1980s, and subsequently from 2001 to the time of the review. Prastacos [12] reviewed the work in blood product inventory management in this first period, while more recent reviews covering the whole blood supply chain also include sections on inventory management [9, 33, 34]. Flint *et al.* [1] reviewed work specifically focused on reducing the wastage of platelets.

One complication that has often been included in research is the assignment of inventory to patients following cross-matching. As I describe in Appendix A, most patients can now safely be allocated PRBCs based on an electronic cross-match following a Group and Screen test. This means that, except for patients with special transfusion requirements, it is no longer necessary to maintain sub-inventories of units assigned to specific patients. While many older papers (e.g. Cohen and Pierskalla [35]) emphasized the importance of the transfusion to cross-match ratio (proportion of cross-matched units eventually transfused) and the cross-match release period (the amount of time that cross-matched units would be assigned to a patient for before being returned to the general stock) when managing PRBC inventory, the change in procedure means that these factors are no longer as relevant and may be ignored, as in Dillon *et al.* [36].

While my focus is on work related to hospital blood banks, the third echelon described by Osorio *et al.* [9], I also discuss some studies at the regional blood centre level (the second echelon described by Osorio *et al.* [9]), where similar methods may be appropriate. In the context of platelet inventory management, Blake *et al.* [37] described these respectively as the "consumer's problem" and the "supplier's problem" and identified two key differences between them. The first difference is that it is reasonable to assume that a consumer's order arrives instantaneously, but not the supplier's where the order may be a collection target from donors. The lead time is the number of periods between placing and receiving an order, and therefore it is reasonable to assume a lead time of zero periods for the consumer's problem. The second difference is that it is reasonable to assume that all newly arriving inventory is of the same age for the supplier, but not the consumer where the products received may be a mix of ages.

2.4.1 Optimizing policies

2.4.1.1 Dynamic programming

Optimal policies for replenishing perishable products with a useful life greater than the inventory review period (the period between which orders may be placed) can, in theory, be found using dynamic programming [38, 39]. I provide some examples

of dynamic programming methods in the discussion of RL approaches in Section 2.5.2.1 below. In practice, finding optimal solutions for realistic problems has been considered infeasible due to the "curse of dimensionality". The state space grows exponentially with the useful life of the product because it must include the age profile of the stock. Nahmias [6] observed that this makes dynamic programming, which requires repeated operations over the whole state space, infeasible for realistic size problems if the useful life of the product is greater than or equal to three periods. Despite advances in computational power, this is still reported to be a challenge. In a study by De Moor *et al.* [40] published in 2022 the authors were able to use the optimal policy as a benchmark when the useful life of the product was two periods, but finding the optimal policy was considered to be computationally intractable for useful lifetimes of three, four and five periods and therefore heuristic benchmark policies were used instead.

Given the practical difficulties in finding an optimal policy, much research on both perishable inventory in general, and on blood inventory, has focused on finding good heuristic solutions. In addition to being less computationally intensive, some researchers [41, 42] suggest that the simpler rules derived from heuristic policies may be preferred by stock managers because they are easier to understand and implement. Research in this area has involved both identifying suitable structures for heuristic policies (see Nahmias [43] for an early example, and Haijema and Minner [42] for a recent example), and finding suitable parameters for those policies that work in specific situations - for example at a particular hospital blood bank.

One simple way to make a dynamic programming approach more computationally tractable is to reduce the size of the state space by aggregating stock items into batches. The solution to the down-sized dynamic program can then by factored up to give an approximate solution to the original problem. Blake *et al.* [44] used this approach to solve the ordering problem for a regional blood centre (the "producer"), based on the observation that one adult dose of platelets for transfusion is typically derived from five donations. The approach was tested on a two month period of real demand data, and orders generated using a time horizon of five steps and a batch

size of 10 would have reduced the daily inventory cost by 18% compared the actual system in place during that period and, just like the real system, would not have encountered any shortages.

2.4.1.2 Simulation optimization

An alternative approach is to use simulation optimization: setting a particular functional form for the desired policy, evaluating the results of the current parameterized policy using a simulation, and using a search procedure to update the policy parameters. I distinguish this from using a simulation to evaluate the potential impact of human-proposed policy changes, which I discuss below in Section 2.4.2. Unlike dynamic programming, simulation optimization may get 'stuck' at a locally rather than globally optimal policy but can be much less computationally demanding for large state spaces.

A widely used, simple heuristic replenishment policy is the base stock policy, parameterized by the order-up-to level $\mathtt{S}$. The order quantity is the difference between the current stock on hand and in-transit and the order-up-to level. The parameter $\mathtt{S}$ therefore corresponds to the largest order that would be placed when following the heuristic policy. Additional parameters can be used to represent more complicated rules. For example an $(\mathtt{R}, \mathtt{s}, \mathtt{S})$ policy also includes parameters for the review period, $\mathtt{R}$, and the reorder point, $\mathtt{s}$. The inventory position is reviewed every $\mathtt{R}$ periods. When the total stock on hand and in-transit is less than or equal to the reorder point $\mathtt{s}$, the order quantity is the same as under the base stock policy. No order is placed if the total stock on hand and in-transit is higher than the reorder point $\mathtt{s}$ [28]. The additional parameters can be particularly beneficial if there is a fixed order cost (e.g. a delivery charge) by preventing frequent small orders. In this thesis, I have only considered periodic review problems with a review period of one period and therefore, following the presentation of related work [45], omit the review-period parameter $\mathtt{R}$.

Early work by Elston and Pickrel [32] used this approach to find separate order-up-to levels for PRBC for each of the eight ABO/RhD blood types, assuming only exact matches were allowed, exhaustively considering different parameters and

simulating the results. More recently, Duan and Liao [46] used a metaheuristic optimizer to search for parameters for several heuristic policies, simultaneously fitting policy parameters for a regional blood centre and three hospitals that it supplied with platelets. A follow up paper [47] applied a similar approach to the PRBC supply chain and investigated the effects of allowing substitution between ABO/RhD blood types. Dalalah *et al.* [48] used simulation optimization, with simulated annealing as the local search procedure, to find fixed order quantities for platelets based on the day of the week with age-differentiated demand. Ejohwomu *et al.* [10] used simulation optimization to find target inventory levels for platelets held by a UK Stock Holding Unit supplying multiple hospitals, and included both ABO/RhD blood types and separately considered pooled, apheresis and paediatric platelets.

Haijema *et al.* [49] studied the platelet producer's problem for a regional blood centre in the Netherlands. They initially found the optimal policy on a downsized version of the problem using dynamic programming. They demonstrated that a simulation optimization approach using local search to fit weekday-specific order-up-to parameters resulted in a near optimal policy on the downsized version of the problem, and could be used to find a good ordering policy for the full-sized problem, where dynamic programming was considered computationally intractable. The simulation results, based on demand data from a real Dutch regional blood centre, presented in Dijk *et al.* [50], suggest that this approach could cut wastage due to expiry from 15-20% to less than 0.1% and Haijema *et al.* [51] extended the approach to consider production breaks during public holidays. The techniques were used to change the ordering procedures at a Dutch regional blood centre, and were eventually incorporated into a software tool [52], leading to a reduction in annual outdating from 5.2% to 1.3% during the five year study period.

2.4.1.3 Stochastic mixed integer linear programming

Several recent studies have applied stochastic mixed integer linear programming to the blood inventory replenishment problem. Unlike dynamic programming, which requires complete probability distributions to be known for any stochastic

variables, stochastic mixed integer linear programming can be used with a number of sampled trajectories of the stochastic variables, representing different possible scenarios. Gunpinar and Centeno [53] found sequences of orders for PRBCs and platelets that minimized the expected cost over the sampled demand trajectories, and Rajendran and Ravindran [54] performed a similar analysis for platelets only and compared the results to four heuristic replenishment policies including a base stock policy. Dillon *et al.* [36] and Meneses *et al.* [55] both used stochastic mixed integer linear programming to fit parameters to heuristic policies for ordering PRBCs, to minimize the costs over the sampled demand trajectories while Rajendran and Srinivas [45] performed similar analysis for platelets comparing the performance of four different heuristic policies with parameters fit using stochastic mixed integer linear programming. In a recent review of the whole blood supply chain, Meneses *et al.* [56] proposed a framework describing how decisions at each level of the blood supply chain, from collection to distribution, could be modelled using such mathematical programming approaches.

2.4.1.4 Performance evaluation and limitations

In most of the work in this area, notional costs incurred over a period are used as the optimization objective, and to compare the performance of different policies. This is often a combination of several different costs including purchase costs, holding costs, shortage costs and wastage costs. Determining some of these costs can be difficult in any setting, but is particularly challenging in the context of the blood supply chain because many of the costs are non-monetary [37]. For example, the reason a shortage cost needs to be imposed in this context is not because of lost income, but because it could lead to a delayed medical procedure, potentially putting a patient's life in danger. Similarly, while there may be a monetary cost associated with disposing of an expired unit, other key considerations for avoiding waste are the moral obligation to make best use of voluntarily donated blood [57, 58, 59] and the risk that donors may be less willing to donate if they believe their donation is likely to be wasted [37]. A common approach for PRBCs and platelets is to assume a ratio of 1:5 between wastage and shortage costs (e.g [45,

49]). The relationship between the different costs will affect the policy that appears to perform the best when evaluated using total (or mean) cost, and some studies (e.g. Rajendran and Srinivas [45]) compare the performance of different heuristic polices with different ratios between the costs. An alternative to using notional costs is to use KPIs. Blake *et al.* [37] suggested that it is more natural for hospital blood bank managers to target levels for service level (percentage of orders that are met from stock) and wastage and analytically found parameters for (`R`, `s`, `S`) policies that met these targets. With this approach, feasible policies were generally found for larger hospitals with relatively uniform demand distributions. Shih *et al.* [60] used goal-programming and a weighted objective methods to optimize platelet ordering decisions for a network of hospitals served by a regional blood centre using three criteria: total cost, wastage and shortages. Additionally, while their objective functions were based on costs, Dillon *et al.* [36] and Meneses *et al.* [55] both included constraints to enforce a minimum service level and maximum wastage limit in their studies fitting policy parameters using stochastic mixed integer linear programming, with Meneses *et al.* [55] also enforcing a minimum proportion of transfusions that were an exact blood type match.

One limitation of much of the work to date is the simplifying assumptions made about the demand. Most recent studies consider stochastic demand, but often using a single demand distribution (for example Dillon *et al.* [36] and Meneses *et al.* [55]). Others, including Rajendran and Srinivas [45], Haijema *et al.* [49], and Gunpinar and Centeno [53] and Duan and Liao [47] adopt a more realistic approach with day-of-the-week specific demand distributions. This dependence on weekday is consistent with descriptive analysis presented in Guan *et al.* [11], Motamedi *et al.* [61] and Li *et al.* [62]. It is rare to see methods evaluated on real demand trajectories: Blake *et al.* [44] reported the performance of their policy on two months of real demand data and Kort *et al.* [52] incorporated their method into a decision support tool and report on its performance prospectively.

Another key limitation, identified by Dumkreiger [63], is the limited coverage of issuing policies and substitution between products. A subsequent review of

the entire blood supply chain by Meneses *et al.* [56] also suggests that while the inventory management of blood products in hospital blood banks is well-covered by the literature relative to higher echelons, it is not common to include blood type compatibility requirements. When substitution has been considered for platelets, it is often in the context of age-differentiated demand where a subset of patients require fresh products [48, 49]. Ensafian *et al.* [64] allowed platelet substitution between ABO/RhD blood types following a fixed priority order and Bakmohammadi *et al.* [65] considered blood type substitution for both PRBCs and platelets with a penalty for non-exact matches based on a preference order for each blood type. When considering ABO/RhD blood type substitution for PRBCs alone, Najafi *et al.* [66], Dalalah and Alkhaledi [67], and Chithraponnu and Umamaheswari [68] used a fixed priority order for substitution for each blood type, Duan and Liao [47] allowed substitution as recourse action with more common blood types more likely to be selected as a substitutes, and Dillon *et al.* [36], Meneses *et al.* [55], and Hamdan and Diabat [69] used substitution penalties based on a preference order similar to Bakmohammadi *et al.* [65]. Dumkreiger [63] and Abdulwahab and Wahab [70] both used approximate dynamic programming approaches, which can be classified as RL, to find issuing policies for PRBCs and platelets, respectively, considering ABO/RhD compatibility and I discuss them below in Section 2.6.2. A recent study by Hendrix *et al.* [71], focused on the fresh food industry, illustrated the computational challenges of using a dynamic programming approach to find optimal replenishment policies for perishable, substitutable products because the state space grows exponentially with the number of products.

2.4.2 Using simulation to evaluate policy changes

An alternative to finding policies using optimization methods is to simulate parts of the blood supply chain to evaluate the potential impact of policy changes suggested by human experts. Rytilä and Spens [72] noted that without a simulation, many of the decisions about practices that they modelled were made using the "gut feelings" of experts and that simulation results provided a basis for informed decision making

and helped to improve communication between different stakeholders. Studies that can inform my work modelled different levels of the blood supply chain, from collection to transfusion [73, 74, 75, 76, 77, 78] to regional blood centres supplying hospitals [79, 80, 81] and a hospital blood bank [72]

These studies are of interest for two main reasons. The first is the way in which performance is evaluated, which provides KPIs relevant to the problem. These include expiries, shortages, the number of emergency and ad-hoc orders, inventory days, and mismatches between patient and donor blood types. The second is to understand the processes that are considered important to model when the limitations imposed by optimization approaches are relaxed. The main factor included in all of these papers (except Katz *et al.* [79]), but often neglected in studies in which optimization methods are used, is blood types and the potential for substituting between them. This requires maintaining a large state space which may make some of the mathematical optimization approaches intractable, but is obviously of interest when this is less of a concern and is very relevant to real-world decisions made in the blood supply chain. Additionally, Kopach *et al.* [82] considered different levels of severity for a shortage depending on the condition of the patient for whom the blood product was requested.

Unfortunately, to the best of my knowledge, none of the simulators used in these studies have been made publicly available. This could be of substantial benefit to the research community because it would facilitate the comparison of different methods on common problems.

2.4.3 Forecasting demand for blood products

Until recently, most of the studies forecasting blood demand used univariate forecasting methods: forecasting future demand based on past demand. This work was generally conducted at regional blood centres where the goal is to plan donor collection programs rather than the short-term ordering decisions that must be made by a hospital blood bank. The most common forecasting period is one month [83, 84, 85, 86, 87], but Wijayanayake and Dandunna [88] and Fanoodi *et al.* [89] both forecast daily demand for platelets. In the studies that consider different blood

types, the most common approach is to train separate models for each blood type [84, 85, 87, 89]. In addition to training separate models for each blood type, Fortsch and Khapalova [86] also used a Vector Auto-Regressive Moving-Average model which outputs a vector forecast (one element for each blood type) based on a multivariate input (the historical demand of each blood type) which outperformed an Auto-Regressive Moving-Average model trained to predict aggregate demand.

In the last few years, as the implementation of EHRs has increased the range and frequency of data available to researchers, several studies have been conducted demonstrating how data from the current state of the hospital and its patients can be used to forecast short-term demand (the following day, or the next few days) and, in some cases, contribute to collection or ordering decisions for PRBCs [62, 90, 91] and platelets [11, 61, 92, 93, 94]. The most common features, in addition to those derived from historical demand, are census information about the number of patients in different hospital locations, demographic information and laboratory test results, but Schilling *et al.* [94] also included information about planned surgical procedures. Khaldi *et al.* [95] used a wide range of features including procedures, patient events, and outside events including road traffic accidents, to make monthly forecasts for monthly demand.

Motamedi *et al.* [61] and Li *et al.* [62] compared the performance of univariate and multivariate models for predicting platelet and PRBC demand, respectively. In both cases, the multivariate methods outperformed the univariate methods. Motamedi *et al.* [61] showed that the difference is most pronounced when the amount of training data is small: the multivariate models achieved much lower mean absolute percentage error (MAPE) than the univariate models when trained using two years of data, but performed similarly when using eight years of data. These studies demonstrate the potential benefit of incorporating information about the state of the hospital and the patients into demand forecasting efforts.

Many of these studies evaluated the quality of the forecasts using KPIs estimated by simulating order decisions on real historical demand trajectories. One key difference in approach is whether there was a separate explicit forecasting step

subsequently used to make a decision [62, 94], or whether the ordering decisions were optimized directly [11, 92, 93]. A pipeline with a separate forecasting component may be more transparent to a human and may make it easier to identify the source of performance issues. However, by directly optimizing the decisions based on the benchmarks that are of interest to hospital blood bank managers such as wastage (instead optimizing a forecasting model to minimize forecasting error) it might be possible to get better decisions, by avoiding human judgements including the most appropriate forecasting horizon. Schilling *et al.* [94] reported that while their Least Absolute Shrinkage and Selection Operator (LASSO) forecasting model performed best on measures of prediction quality, best performance when simulating decision making came from forecasts made by their recurrent neural network - the most accurate forecast was not the best prediction for supporting decision making.

In addition to creating a simulation that closely mirrors real practice to evaluate the potential impact of their forecasting models, Li *et al.* [62] also considered whether they could support changes in practice - specifically moving from deliveries each day to every other day. They found that, based on the historical data, this could further reduce costs compared to receiving daily deliveries without increasing emergency orders or wastage.

I have identified a single study, Quinn *et al.* [90], that reported the outcome of the real-world deployment of a system that uses real-time information from EHRs to support blood ordering decisions. The full detail of the model was not provided, so it is not clear the extent to which ML was used, but the algorithm incorporated historical five-day rolling averages of demand with an estimate of blood-group specific demand in the next 48 hours derived from the haemoglobin test results of current patients and the historical probability that a patient would receive a transfusion within 48 hours following a haemoglobin test result in bands of 10g/L. The system generates a suggested order for PRBCs, based on the forecast demand and current stocks. The authors compared inventory KPIs for the year before the algorithm was implemented with the year after it was implemented and reported

a reduction in both mean monthly PRBC wastage due to expiry (from 1.79% to 0.72%) and mean total daily PRBC inventory (from 401.7 units to 309.0 units), and stated that there were no shortages and no delays to transfusions or surgical procedures due to a lack of PRBC units. Additionally, no increase in urgent orders to the supplier was observed for the year after the system was implemented. The PRBC wastage due to expiry before implementation of the system (1.79%) was similar to that reported for England in 2017/2018 [2] (1.68%). This suggests that there is scope for similar techniques, making use of real-time information about the state of the hospital, to be beneficial to UK hospital blood banks.

With the exception of Quinn *et al.* [90], these studies did not consider blood types. For a hospital blood bank ordering from a regional blood centre this is an important part of the ordering decision: how many units of each type should be ordered. In making this decision it is also likely to be important to consider the complex substitution relationships between different blood types.

2.4.4 Current practice

Despite significant research efforts going back to the 1960s, few if any of the methods outlined above are being regularly used by hospital blood banks to determine ordering schedules and quantities. A review of historical data and interviews with transfusion laboratory managers in England [58, 96] found that all seven of the transfusion laboratories in the study used order-up-to levels set on the basis of experience, and two of the transfusion laboratories also received standing orders of a fixed quantity. This work found that the main drivers to achieving a good balance between service level and wastage were good training, the use of target stock levels, collaboration with other departments, simple procedures, transparency about current stock holding and a focus on freshness (including the use of an Oldest Unit First-out (OUFO) policy for units suitable for a patient). This finding is supported by a recent review of work to reduce platelet wastage [1], in which the authors identified only one study in which a theoretical model was deployed and evaluated in practice [52]. I describe the current practice at the UCLH transfusion laboratory, our partner hospital, in detail in Appendix B, and

during my shadowing visit in July 2021 the ordering decisions were based on a mix of order-up-to levels (for PRBCs) and standing orders (for platelets), all of which were determined through experience rather than based on analytical models, consistent with the findings of Stanger *et al.* [96].

Despite this, there is clear evidence from Quinn *et al.* [90] that blood product inventory management can be improved by incorporating near real-time data into ordering decisions. Three of the seven hospital transfusion laboratory managers interviewed by Stanger *et al.* [58] stated that they altered their daily order-up-to levels based on information about scheduled surgeries and recurring transfusions. The implementation of EHRs mean that this type of data, and other relevant features, should now be more available in transfusion laboratories, but that does not mean that the staff necessarily have the time or the analytical tools to incorporate the information into their decision making. ML models can make use of the wide range of features to produce forecasts or recommend order quantities. RL, which I discuss in the next section, is particularly well suited to making order recommendations.

2.5 Reinforcement learning

In their classic introductory textbook to the field, Sutton and Barto [97] provide the following definition of RL:

> "*Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal. The learner is not told which actions to take, but must instead discover which actions yield the most reward by trying them.*"

RL differs from supervised learning in two major ways. The first is that, unlike in supervised learning, we cannot make the assumption that inputs are identically and independently distributed. The decision taken at one timestep may affect the state of the world and therefore future observations. RL methods therefore need to take into account both the immediate and future consequences of actions taken by the learner. The second is that the feedback for RL comes from a reward signal, some measure of the quality of the learner's behaviour that may be delayed relative

to an action or very sparse, as opposed to being provided with the correct answer.

Many of the most famous and influential works in RL have involved board games or, more recently, video games. These offer an ideal research platform because the rules are known and it is often possible to efficiently and safely collect a vast amount of experience from which to learn (compared to, for example a robot arm acting in the real world). One of the earliest examples was a program that learned to play checkers in the 1950s [98]. Expert backgammon players learned new strategies after watching how an RL agent, TD-Gammon, played the game in the 1990s [99]. And, in the 2010s, agents trained using RL methods demonstrated performance that matched or exceeded the best human players in the world at the boardgame Go [100, 101], a subset of games published for the Atari 2600 home video game console [7], and the online real time strategy video game StarCraft II [102]. This has lead to a popular view that RL is only good for games and toy problems, but there are recent reports of RL methods in use solving real practical problems including chip design [103] and data centre cooling [104].

While I provide an overview of the core concepts in this section, I direct the interested reader to Sutton and Barto [97] for a thorough introduction to RL, including a detailed history of the field, and to Arulkumaran *et al.* [105] for a recent review of DRL. Members of the operational research community less familiar with ML and neural networks may find the introduction to the topic in Boute *et al.* [106] a useful starting point.

Where not otherwise cited, the overview of RL in this section is based on the treatment in Sutton and Barto [97], supplemented by Kochenderfer *et al.* [107] and Lapan [108].

2.5.1 Markov decision processes

Markov decision processes (MDPs) provide a framework for formally describing a sequential decision making problem. MDPs were introduced in the field of optimal control by Bellman [109] and popularized in RL research by Watkins [110].

An MDP describes a scenario in which an **agent** interacts with an **environment**, which is everything in the problem that is external to the agent.

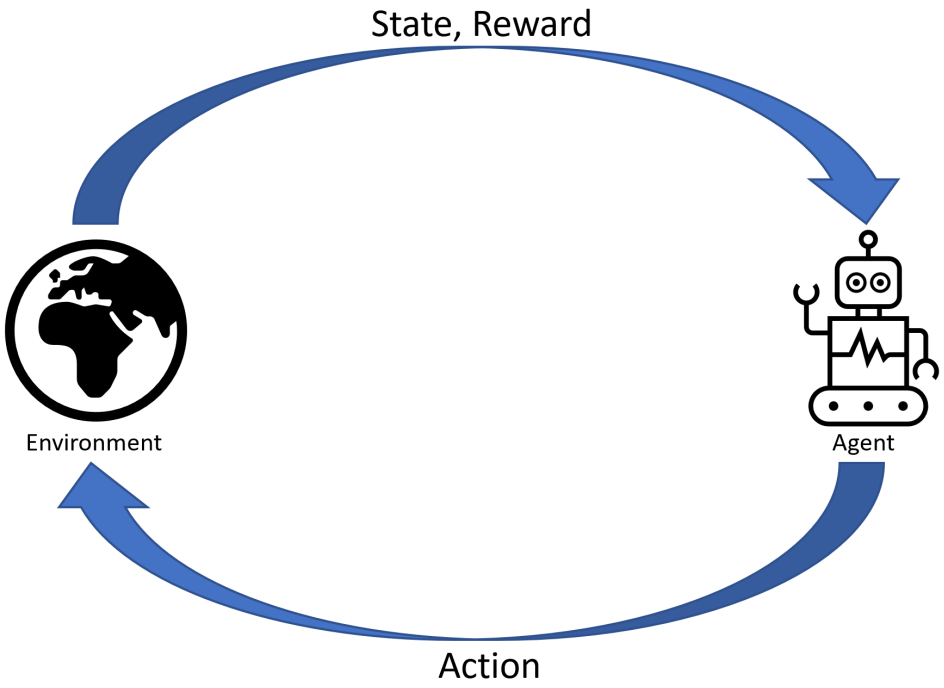


Figure 2.1: Interaction between the agent and environment in an MDP.

At discrete points in time, the agent observes the state of the environment (S_t) and selects an action (A_t). The environment will be updated in response to the action, and will also emit a **reward**, a numerical signal that relates to the quality of the agent's behaviour. At the next decision point, the agent will observe the reward (R_{t+1}) and the new state of the environment (S_{t+1}), and make its next action based on the state of the environment. If the interactions between the agent and the environment do not break down neatly into individual episodes (for example, a single game of checkers), then we introduce a **discount rate** that can be used to determine the present value of future rewards so that the sum of future discounted rewards is finite. This discount rate is analogous to a discount rate used to determine the net present value of an investment to account for the time value of money. The goal of the agent is to learn a **policy**, a function mapping states to actions, that will maximize the expected future discounted reward, or expected **return**, it achieves when interacting with the environment. I illustrate this set-up in Figure 2.1.

Recording the sequence of states, actions and reward from the agent interacting with the environment would lead to a trajectory, the opening items of which would

be $\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, ...\}$ [97, p. 48]. I follow the convention of Sutton and Barto [97] in which the reward emitted after A_t is taken is labelled as R_{t+1}.

An MDP can be defined with a tuple of five items:

- $\mathbb{S}$: the state space, the set of possible states of the environment, $s \in \mathbb{S}$
- $\mathbb{A}$: the set of possible actions that may be taken by the agent, $a \in \mathbb{A}$
- $T(s'|s, a)$: the state transition model which gives the probability of transitioning to state s' if starting in state s and taking action a.
- $R(s, a, s')$: the reward function which gives the expected immediate reward when transitioning to state s' if starting in state s and taking action a
- γ: the discount factor applied to future rewards, $\gamma \in (0, 1]$

The decision process is Markovian because the state transition model and the reward function obey the Markov principle: future states and rewards are conditionally independent of states S_0 to S_{t-1} given S_t. In this work I specifically consider stationary, finite MDPs in which the transition model and reward function do not change with time and sets of states, actions and rewards ($r \in \mathbb{Y}$) are all finite.

The agent learns a policy that returns a probability distribution over possible actions given the current state. A policy can be written $\pi(a|s)$, representing the probability of taking action a in state s. The policy may be deterministic, in which case the probability of the selected action is 1 and the probability for every other action is 0. A deterministic policy is often written as $\pi(s)$.

The **value** of a state given a particular policy π, $V^\pi(s)$, is the expected return of being in state s and following policy π.

The **state-action value** of a state-action pair given a particular policy π, $Q^\pi(s, a)$, is the expected return of being in state s, taking action a, and following policy π thereafter.

The **advantage** of an action in a particular state, given a particular policy, is $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. This is a measure of the quality of an action relative to other actions in the current state, because it is the difference in expected return

of taking action a in state s and then following policy π thereafter, and following policy π starting in state s.

For finite MDPs, an optimal policy, π^*, can be defined as a policy where for every state $s \in \mathbb{S}$ $V^{\pi^*}(s) \geq V^{\pi'}(s)$ for any policy π'. This means that the expected return from every state, following policy π^*, is at least as good as that from following any other policy. There may be more than one optimal policy, but they all share the same optimal value function, $V^{\pi^*}(s)$. The optimal state-action value function can be similarly defined: for every state $s \in \mathbb{S}$ and every action $a \in \mathbb{A}$, $Q^{\pi^*}(s,a) \geq Q^{\pi'}(s,a)$ for any policy π'. The optimal value function can be written in terms of the optimal state-action value function as shown in Equation 2.1.

$$V^{\pi^*}(s) = \max_{a \in \mathbb{A}} Q^{\pi^*}(s,a) \tag{2.1}$$

If the agent cannot observe the complete state then the problem can be modelled as a partially observable Markov decision process (POMDP). Two additional items must be defined in addition to the five elements described above: the set of possible observations ($\mathbb{O}$) and the observation model $O(o|s)$, which gives the conditional probability of observation o given that the environment is actually in state s. In practice, it is often not necessary to use this formalism and useful policies can be learned by augmenting the observation. One example is Mnih *et al.* [7] in which an individual observation, a single frame from an Atari game, does not include the complete state because the direction and speed of motion cannot be determined from a single frame. This issue is overcome by stacking multiple, sequential frames to form a single observation, without the need to represent the problem as a POMDP.

2.5.2 Reinforcement learning approaches

In this section I provide a high level overview of RL approaches and key terminology. An appropriate approach to a given problem will depend on a range of factors including prior knowledge of the environment, the size of the state and action spaces, and the safety and efficiency of collecting experience directly from

the environment.

2.5.2.1 Dynamic programming approaches

One of the key strands of research that led to what is now described as RL is work on optimal control and dynamic programming [97, pp.14-15], particularly the work of Richard Bellman [109]. Many researchers in operational research and optimal control use similar techniques without considering their work RL (e.g. the work discussed in Section 2.4.1 above), but these methods do offer a way to solve RL problems and are therefore can be considered to be RL methods [97, p. 15].

Value and state-action value functions for a given policy satisfy recursive relationships, called Bellman equations, which define the function in terms of the immediate reward and the value function at the next state. The Bellman optimality equations, that is the Bellman equations for the value function and state-action value function for an optimal policy π^*, are presented in Equations 2.2 and 2.3.

$$V^{\pi^*}(s) = \max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}, r \in \mathbb{\Psi}} p(s', r|s, a) \left[r + \gamma V^{\pi^*}(s') \right] \tag{2.2}$$

$$Q^{\pi^*}(s, a) = \sum_{s' \in \mathbb{S}, r \in \mathbb{\Psi}} p(s', r|s, a) \left[r + \gamma \max_{a' \in \mathbb{A}} Q^{\pi^*}(s', a') \right] \tag{2.3}$$

These equations can be used as the basis of iterative algorithms that are guaranteed to converge to the optimal value or state-action value function, from which the optimal policy can be extracted. **Value iteration** is an example of such a method, which directly adapts the Bellman optimality equation into an update rule, as shown in Equation 2.4 where j indicates the iteration number.

$$V_{j+1}(s) = \max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}, r \in \mathbb{\Psi}} p(s', r|s, a) \left[r + \gamma V_j(s') \right] \tag{2.4}$$

This update is performed for all states repeatedly in a loop until convergence (in practice, until the largest difference between successive estimates is below a tolerance). A deterministic policy, such that $\pi(s)$ gives the action to be taken in state s, can then be constructed that approximates the optimal policy using Equation

2.5.

$$\pi(s) = \arg\max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}, r \in \Psi} p(s', r|s, a) \left[r + \gamma V(s') \right] \tag{2.5}$$

Alternatively, a similar approach can be used for the state-action values, Q-value iteration, which requires additional memory to store all of the state-action values but does not require the one-step look ahead in Equation 2.5 to identify the policy. I use Q-value iteration in Chapter 3 and value iteration in Chapter 4.

Value iteration, and related dynamic programming approaches, can be considered 'exact' solutions: in the limit of infinite iterations they would converge to the optimal policy. However, there are two key limitations that mean they often cannot be used. The first is that they require full knowledge of the environment: the update requires an expectation over transitions and rewards which is only possible if the exact transition dynamics and reward function are known. The second is that, as I explain in Section 2.4.1, dynamic programming approaches require repeated operations over the whole state space, which grows exponentially with the number of feature variables in the state and therefore quickly becomes computationally intractable.

The first limitation can be addressed using methods that learn from samples of experience obtained by interacting with the environment, while the second has been addressed by approximating the value function or by directly approximating the policy function.

2.5.2.2 Learning from experience in a small state space

In a scenario where the transition model and/or reward model are not available for dynamic programming, value functions can be estimated based on experience collected from interacting with the environment. If the state space is small, the setting may be described as "tabular", because a table of entries can be maintained of the value function for each state (or, of the state-action value for each state-action pair).

Experience is collected by an agent following a behaviour policy in the

environment. One distinction is whether we are learning the value function for this behaviour policy (**on-policy learning**), or whether we are trying to learn the value function of the optimal policy based on experience collected by following the behaviour policy (**off-policy learning**).

Another distinction is whether we update our estimates of the value function using complete episodes of experience, **Monte Carlo** methods, or based on individual transitions using bootstrapping to incorporate information about what happened next: **temporal difference (TD) learning**.

A simple example of an on-policy Monte Carlo approach would be to collect a large number of episodes of experience in the environment. For every state visited in each episode, we would calculate the return (the sum of discounted future rewards) from the first time the state was visited in that episode. The value function for each state could then be estimated as the mean of the returns from that state over the episodes. This approach is straightforward, but it relies on the task being episodic and no learning can take place until an episode is completed.

TD learning, like dynamic programming, uses bootstrapping: estimates of the value function are updated in part based on the estimate of the value function for a future state. This means that, unlike Monte Carlo methods, they can learn online as experience is obtained rather than needed to wait for an entire episode to be completed. A popular method for temporal difference learning in a tabular setting is Q-learning [110], which is an off-policy method.

In Q-learning, the agent interacts with an environment following a behaviour policy, collecting tuples of experience (s, a, r, s'). The estimates of the state-action value function are then updated following Equation 2.6, where α is the learning rate controlling the contribution of each update.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a' \in \mathbb{A}} Q(s',a') - Q(s,a) \right] \tag{2.6}$$

The term in the square brackets is the TD error. The discounted future reward is estimated as the sampled immediate reward plus the discounted state-action value of the next state. Once the state-action value function has converged, the expected

value of this difference is zero.

State-action-reward-state-action (SARSA) is a method for on-policy TD learning. The key difference is that instead of including the action with the highest state-action value in the next state in the calculation of the TD error, it uses the actual action that was taken in the next state, a' and therefore requires experience tuples of the form (s, a, r, s', a') which give it its name.

SARSA and Q-learning are examples of TD(0) methods, because the TD error is calculated based on a single sampled transition and a bootstrapped value function from the next state. Monte Carlo and TD learning methods can be combined using n-step returns or eligibility traces to balance the contribution from each individual sampled episode and bootstrapping [97, p. 287].

2.5.2.3 Approximation and deep reinforcement learning

As the size of the state space grows there are two challenges to address. The first is the memory requirements for holding a tabular representation of the value function. The second is populating the entries of the table: in the methods above the entry for a state or state-action pair is only updated when it is visited. As the state space grows, we will often encounter states that we have never visited before. Therefore, we need to a way to recognize the similarity between states so that our knowledge about suitable behaviour can be updated based on experience of similar states that we have visited in the past [97, p196].

One approach to this is learning an approximate value (or state-action value) function. These approaches are often referred to as as approximate dynamic programming (ADP) in the operational research literature [111]. This could be a linear function with manual feature engineering but, as in many other fields of ML, this approach has fallen out of favour in recent years and the most common approach is to use deep neural networks. This is value-based DRL. A popular example of a value-based DRL method, that I use in this thesis, is Deep Q-Network (DQN) [7]. In DQN, a neural network is used to approximate the state-value action function and, for a given input state s, outputs an estimate of $Q(s, a)$ for every action. I present pesudocode for the specific DQN-based algorithm that I have used in this

thesis in Algorithm 1 in Section 3.4.6.

An alternative approach is to directly approximate the policy. These methods are commonly-referred to as policy gradient methods, because the policy parameters are updated by gradient ascent to maximize the performance. The policy function must therefore be differentiable with respect to its parameters. This requirement means that deep neural networks are a popular choice for approximating the policy, giving rise to policy-based DRL methods. The classic policy gradient algorithm, REINFORCE, uses Monte Carlo sampling to calculate the return from a given state-action pair and weights the updates by the return. This means that the probability of taking action a in state s will be increased by a large amount if the return was highly positive, and decreased by a large amount if the return was highly negative.

Pure policy gradient methods can be slow to learn due to the high variance of the Monte Carlo samples. Actor-critic methods combine value-based and policy-based methods. A popular actor-critic approach is the Advantage Actor-Critic (A2C) [112] which uses two neural networks: an actor network to approximate the policy and a critic network to approximate the value function. The value function is used to estimate the advantage of a state-action pair for each training example, which is used to weight the policy gradient updates instead of the Monte Carlo estimate of the return used in REINFORCE.

In a policy gradient methods, updates are being made to the parameters in parameter space, but the effect of the update happens in policy space. There is not necessarily a straightforward mapping between the two, and we should be cautious about making large updates. Policy gradient methods are on-policy. This means that after every update any previously collected experience must be discarded and new experience must be collected by following the new policy. This can lead to vicious circle when training policy gradient methods: a bad policy update can lead to the collection of bad experience samples, which will then be used to update the policy at the next step and lead to even poorer performance. Trust region methods, including Trust Region Policy Optimization (TRPO) [113] and Proximal Policy

Optimization (PPO) [114] have been developed which aim to limit the effect of any single update on the policy function and therefore avoid this performance collapse. I use PPO in this thesis and present pseudocode for the specific PPO-based algorithm in Algorithm 2 in Section 3.4.6.

2.5.2.4 Neuroevolutionary methods

The parameters of the neural networks used in the DRL methods including DQN and PPO are optimized using the backpropagation algorithm to estimate the gradient of the objective function with respect to each parameter. Neural network parameters can also be fit using evolutionary algorithms: the application of these optimization methods to the parameters of a neural network is called neuroevolution. Neuroevolution is conceptually similar to simulation optimization, described in Section 2.4, except that the parameters sought are the weights of an artificial neural network instead of the parameters of a heuristic policy.

Evolutionary algorithms are black-box optimization methods inspired by the process of evolution in nature. Black-box optimization methods are designed to solve problems where the functional form of the objective function is not known and so the derivative cannot be computed, but where is is possible to evaluate the performance of a candidate solution. Evolutionary algorithms maintain a population of candidate solutions, or a search distribution over candidate solutions, to an optimization problem, and iteratively evaluate the current candidates and update the population (or parameters of the search distribution) based on their performance ("fitness"). The population update may include selection (choosing candidate solutions to be retained or to reproduce based on their performance or a measure of diversity), mutation (randomly altering the properties of a parent solution), and crossover (combining the properties of more than one parent solution) [115].

Galván and Mooney [116] recently reviewed the application of neuroevolution to deep learning in general, while Bai *et al.* [115] reviewed applications of evolutionary computing to RL, including policy search using evolutionary strategies, genetic algorithms and genetic programming and highlighted the use of evolutionary computing for determining the architecture of the neural network

as well as optimizing the individual weights. These methods differ in how they encode a candidate solution, how subsequent generations of candidate solutions are generated, and how the strongest performers are identified. While the value and policy based RL methods described in Sections 2.5.2.3 explore in state and action space, evolutionary algorithms explore in the parameter space of the neural network representing the policy.

A key driver of recent work in this area was Salimans *et al.* [117], who demonstrated that an evolutionary strategy algorithm (now widely referred to as OpenAI ES), was competitive to DRL algorithms (DQN and Asynchronous Advantage Actor-Critic (A3C)) on a range of popular RL environments including MuJoCo tasks and Atari games. They noted that the black-box optimization approach has several advantages over widely used DRL approaches. By learning from a fitness function calculated based on whole episodes instead of individual steps, the approach does not need temporal discounting, does not require the estimation of value functions, and there is no disadvantage to long temporal gaps between actions and the corresponding rewards. Learning from the fitness of a whole episode makes OpenAI ES less sample efficient than popular DRL methods, but Salimans *et al.* [117] noted that this can be offset by lower computational requirements (because there is no need to perform backpropagation, or estimate value functions) and a parallel implementation can be used to reduce wall time to less than DRL competitors because the algorithm is well-suited to being run on a large number of parallel workers (minimal communication is required; just the scalar return of the episode). I adopt an implementation of OpenAI ES in Chapter 6 and present pseudocode describing this implementation in Algorithm 6 in Appendix J.

While the evolutionary strategy approach used by Salimans *et al.* [117] does not rely on backpropagation, it is a gradient based optimization method. It maintains a distribution over the neural network parameters, and these are updated at each iteration based on an estimate of the fitness function with respect to the parameters. Such *et al.* [118] showed that a simple genetic algorithm, which does not require

estimating gradients, could also achieve competitive performance on popular RL environments (compared to DQN, A3C and OpenAI ES), and could also be trained quickly and in parallel. I adopt an implementation of SimpleGA in Chapter 6 and present pseudocode describing this implementation in Algorithm 7 in Appendix J.

Both Salimans *et al.* [117] and Such *et al.* [118] adopted parallel CPU-based implementations to reduce the wall time required to find good policies. Parallel implementations can be used to both get more stable estimates of a stochastic reward function for single candidate solution by performing multiple rollouts, and to roll-out multiple candidate policies at the same time. The recent development of GPU-based RL environments [119, 120], and GPU-accelerated implementations of evolutionary algorithms [121, 122], enable these methods to be applied in parallel on GPUs, which have many more (less powerful) cores than CPUs and are specifically designed to apply to same operations to a large number of inputs in parallel. These methods can therefore now be used on a single, consumer-grade GPU instead of a large number of CPUs.

Beyond fitting parameters, evolutionary algorithms have been used in conjunction with other RL approaches to support hyperparameter tuning, exploration and reward shaping [115, 123, 124, 125].

2.5.2.5 Offline reinforcement learning

A common assumption when training an RL agent is that we will have access to the environment. This is key to the concept of learning through trial-and-error. In many areas where RL might be useful, however, it would not be practical or ethical to allow an agent to learn in the real environment. Healthcare is one such area.

If lots of historical data is available, then it is possible to learn policies based on the "behaviour policy" - the policy the person or other agent followed when the data was being collected. A simple way to do this is behavioural cloning, performing supervised learning on state-action pairs to learn a function that mirrors the behaviour policy.

The behaviour policy may not be optimal. Trying to improve on it requires considering counterfactual questions - what would the outcome have been if a

different action had been selected? Levine *et al.* [126] described how distributional shift (small errors compound, leading an agent into states that are not part of the training set and therefore where its policy will be poor) makes these counterfactual questions challenging and reviewed proposed methods for offline RL that address that issue.

The task I consider in this thesis, managing platelet inventory, is one for which it would not be ethical to allow an agent to learn in the real environment. Fortunately, the ordering and issuing actions that I consider can be assumed to only affect the stock holding of the hospital blood bank and not the wider hospital. It is unlikely, for example, that in general ordering decisions made in the hospital blood bank would affect whether a patient is admitted or whether their clinical team requests blood products for them. Therefore it is possible to simulate how different decisions would have affected stock levels, allowing an agent to safely explore the consequences of taking different actions from the same state.

2.5.3 Multi-agent reinforcement learning

In single-agent RL we assume that while environmental transitions are stochastic, there is no mechanism whereby the environment has agency to choose how it responds to the policy adopted by the agent. In many problems it may be necessary to consider other agents (which can be viewed as part of the environment from the perspective of each individual agent), each of which may change its policy to act competitively or co-operatively depending on their goals. Models of this process are the subject of game theory and are traditionally referred to as games. The RL methods described in the preceding sections can be applied naively, or used as the basis for more advanced methods that account for additional complexities, to find policies for one or more agents acting in an environment with one or more other agents [127]. DRL has been demonstrated to be effective in complex multi agent environments including those based on the computer game StarCraft II [102], city traffic control [128], and for voltage control in simulated power networks [129].

A general model for multi-agent games is the partially observable stochastic game (POSG). An MDP is a POSG with a single-agent and fully observable

states [127]. I adopt the agent-environment cycle (AEC) game model for a multi-agent environment introduced by Terry *et al.* [130], who showed that for every AEC game model there is an equivalent POSG model, and vice versa. This model was specifically designed to support a general programming interface for multi-agent environments. The state-action transitions of the agents are modelled deterministically, with the stochastic elements of the transitions handled using a separate "environment" agent, with a stochastic transition function between states. The environment agent does not take any actions. Individual agents may not have full access to the environment state (for example, they may only have access to local information), and therefore they must learn based on observations of the state as in the case of the POMDP described above in Section 2.5.1. The Python library PettingZoo was introduced by Terry *et al.* [130] for creating multi-agent environments based on the AEC game model.

An AEC game model is defined by a tuple of 11 items [130]:

- $\mathbb{S}$: the set of possible states of the environment, $s \in \mathbb{S}$.
- S_0: the initial state of the environment.
- N: The number of agents in the environment (not including the environment agent).
- $\mathbb{A}_i$: the set of possible actions for agent i, $a_i \in \mathbb{A}_i$.
- $T(s'|s, a_i)$: the state transition model for agent i. These state transitions are deterministic.
- $P(s'|s)$: the transition function for the environment.
- $\mathbb{Y}_i$: the set of rewards for agent i, which is assumed to be finite.
- $R_j(s, i, a_i, s', r)$ is the stochastic reward function for agent j and represents the probability of agent j receiving reward r when agent i takes action a_i while in state s and the environment transitions to state s'.
- $\mathbb{O}_i$: the set of possible observations for agent i, $o_i \in \mathbb{O}_i$

- $O_i(o|s)$: The observation function for agent i, the probability that agent i will observe ω while the environment is in state s.

- $\nu(s, i, a_i, j)$: the next agent function: the probability that agent j will be the next agent to act given that agent i has just taken action a_i when the environment was in state s.

For each episode the AEC game, the initial state is S_0. The environment agent is assumed to act first, and a stochastic step here may be used to determine which of the (non-environment) agents acts first, and the state that will be used to generate its observation. In each subsequent step, the next agent to act, agent i, makes observation $o_i \in \mathbb{O}_i$, and takes action $a_i \in \mathbb{A}_i$ based on its policy. The observation is generated based on the state, using observation function $O_i(o|s)$. If the current agent is the environment agent then the next state is sampled using the stochastic transition function $P(s'|s)$, and otherwise it comes from the deterministic transition function for agent i, $T(s'|s, a_i)$. Each agent receives reward $R_j(s, i, a_i, s', r)$ from the transition. The next agent to act, k, is determined by the next agent function $\nu(s, i, a_i, k)$.

When learning policies using multi-agent RL, there are additional challenges beyond those faced in a single-agent environment. Albrecht *et al.* [127] set out four main additional challenges: non-stationarity, equilibrium selection, multi-agent credit assignment and scaling to many agents. From the perspective of a single agent, the actions of other agents can be viewed as part of the environment and this will mean that the transition dynamics of the environment will appear to change over time as the policies of the other agents change, making learning much more challenging. There may exist multiple possible combinations of policies that form a Nash equilibrium, for which no agent can improve its reward by changing its own policy if other agents continue to follow the same policy, and additional criteria or communication may be required to avoid convergence to sub-optimal equilibria. Correctly assigning credit to actions is a challenge in single-agent RL (temporal credit assignment: understanding which of the agent's own actions led to the reward), and this is compounded in the cases of multi-agent RL by the fact that

it is not necessarily an agent's own actions that led to the current reward: it may be due to the actions of another agent or due to a specific combination of actions from multiple agents. Scaling to a large number of agents can lead to an exponential increase in the number of joint-actions and make some of the other problems worse: more agents with changing policies may increase the non-stationarity of the environment and make it more difficult to correctly assign credit to an agent's actions.

Two simple approaches to multi-agent RL are to reduce the problem to single-agent RL by central learning or independent learning [131] and apply standard RL methods [127]. These approaches address different challenges: central learning uses a single agent that selects an action at each step from the joint action space, mitigating the challenges of non-stationarity and multi-agent credit assignment but requires careful consideration of how to convert the joint reward into a single reward, leads to exponential growth in the size of the action space as the number of agents increases, and assumes full centralization rather than agents acting locally based on their own observations without necessarily having knowledge of what others agents can see of how they are acting. In independent learning, all of the other agents are assumed to form part of the environment. Unlike central learning, the action space is not affected by the number of agents and the agent acts locally based on its own observations and learns from its own reward but non-stationarity may lead to unstable learning and prevent the policy from converging.

More complex models often utilize the "centralized training with decentralized execution" paradigm [132], which involves sharing data during training to create policies that, once trained, can be run in a decentralized manner [127, 133]. Access to data about other the observations and actions of other agents during training can help to mitigate the challenges of non-stationarity and multi-agent credit assignment (in a similar way to central learning). These techniques include using a critic network that is conditioned on the joint trajectory of all the agents [134, 135], decomposing the value function to attribute the reward to each agent based on its contribution [136, 137], and learning how to communicate using the noisy

communication channels that would be available during decentralized execution [132].

See Albrecht *et al.* [127] for a modern introduction to multi-agent RL, including DRL approaches, and Gronauer and Diepold [138] for a survey of research on multi-agent DRL. Co-operative, rather than competitive, multi-agent DRL may be particularly relevant to the blood supply chain and OroojlooyJadid and Hajinezhad [133] recently reviewed work in this area.

2.5.4 Implementing reinforcement learning environments

In this thesis, I refer to software implementations of MDPs or, for multi-agent problems, AEC games, as an "environment" because they encapsulate the logic of the theoretical environment from Figure 2.1. The environments are implemented as Python classes, a set of instructions for creating an object with both data and methods, based on open-source implementations for generic environments and customized with additional logic to represent the specific scenarios. Open source Python libraries provide application programming interfaces (APIs) for working with RL environments. In the context of a software library, an API is the interface (e.g. classes, methods and functions) through which a developer interacts with the library. It provides a structured way for a developer to use the functionality of the library without needing to understand the details of the underlying implementation.

During the work on Chapter 3, the Python library OpenAI Gym [139] was the standard way to implement RL environments in Python, providing the `gym.Env` class to represent a generic MDP. During work on this thesis maintenance of the project was taken over by the Farama Foundation, the interface was modified, and the actively maintained version is now called Gymnasium [140].

Additionally, for the work in Chapters 4, 5 and 6 I developed GPU-accelerated RL environments based on the Python library gymnax [119], which provides a class representing a generic MDP implemented using the Python library JAX [141]. In Chapter 6, I took inspiration from the standard implementation for an AEC game provided by the Python library PettingZoo [130] to develop GPU-accelerated RL environments that could be used to jointly learn two policies.

I used these environments as a standard way to represent an MDP/AEC game and not exclusively for learning policies with RL methods. The environments were also used to evaluate policies computed, fit or developed using alternative methods.

2.5.5 Reinforcement learning in healthcare

Yu *et al.* [142] surveyed work using RL to address challenges in healthcare and found that most of the work was focused on dynamic treatment regimes: learning time-dependent treatment decisions for patients. The most well known of these is probably Komorowski *et al.* [143]: using RL to learn treatment strategies for intensive care patients with sepsis. Yu *et al.* [142] highlighted other applications including automated medical diagnosis, resource scheduling and allocation, drug discovery and health management. Due to the safety challenges involved, the majority of this work has been carried out on historical data, in a simulation, or (as in this thesis) some combination of the two.

Wang *et al.* [144] used RL to develop a decision support tool for recommending whether a patient in the intensive care unit should be given a PRBC, platelet or fresh frozen plasma transfusion using historic patient trajectories from critical care datasets.

One study that I found particularly helpful as I considered how to start work on this thesis and develop a custom RL environment was Allen and Monks [145], an example of using DRL for operational decision making in healthcare where the agent must decide how many staffed beds should be available in a simple simulated hospital.

2.6 Reinforcement learning for perishable inventory management

There has been increasing interest over the last few years in applying RL to problems from operational research. Balaji *et al.* [146] implemented a set of online stochastic optimization problems from the operational research literature, including the multi-period newsvendor problem with lead times and lost sales,

as a benchmark to evaluate the performance of RL on these problems. They found that policies trained using DRL could compete with or exceed standard baselines. OR-Gym [147], an open source library that provides RL environments for a set of operational research problems using the OpenAI Gym interface, includes these benchmark tasks and four additional operational research problems including another inventory management problem: a multi-echelon inventory management problem with stochastic consumer demand for a non-perishable product. Hubbs *et al.* [147] applied PPO [114] with the same network architecture to all of their new tasks and found that DRL can learn "*competent*" policies in all the tasks and outperformed the benchmarks in the more complex environments. The availability of these open source environments has the potential to facilitate progress in the field by providing a standard set of problems to which new approaches can be applied.

OR-Gym does not contain an environment for perishable inventory with a multiperiod lifetime (in the newsvendor model, the product is assumed to be a newspaper which has no value at the end of the day on which it is delivered), and there are relatively few papers looking at the application of RL to inventory management in general. Rolf *et al.* [148] and Yan *et al.* [149] both reviewed the application of RL to supply chain management generally in 2022. Yan *et al.* [149] observed that the majority of studies consider non-perishable inventory, Rolf *et al.* [148] noted that the majority of research to date had focused on modelling generic supply chains, and both studies identified that a key limitation for deployment was the fact that many organizations could not provide streaming data about their own inventory position which would be required to support real-time decision making, let alone that of suppliers or customers which could help improve co-ordination between different levels of the supply chain. Neither of these reviews identified examples of RL being used to find issuing policies for perishable products. While I have not identified any studies looking at learning replenishment policies for blood inventory in a hospital blood bank using DRL, there are a small number of recent studies looking at replenishment policies for perishable inventory, which I discuss below.

2.6.1 Replenishment policies for perishable products

Kara and Dogan [150] used two tabular RL methods, Q-learning [110] and SARSA [151], to find replenishment policies for a perishable product with a fixed lead time and stochastic demand. These policies were compared to heuristic replenishment policies with parameters fit using a genetic algorithm. They demonstrated the benefit of including information about the age of the stock in the state representation, and found that SARSA performed particularly strongly relative to the other methods when the demand was highly variable and the useful life of the product was short.

Sun *et al.* [152] used DQN to learn policies for a fresh-product retailer selling a single perishable product with stochastic demand, and ordering from a single wholesaler. This problem is analogous to a simplified hospital blood bank. The authors compared Q-learning to DQN, using a discretized state and action space for Q-learning because the full state and action spaces used in their DQN method would otherwise be computationally intractable. The authors found that DQN achieved consistently lower expiries than Q-learning (with the discretized state space) over a range of three different demand distributions, but did not provide a comparison to any standard replenishment heuristics. DQN was specifically designed to deal with large state spaces, and this study is an example where the ability to represent the large state and action spaces does have clear advantage over learning policies based on a discretized version.

In series of connected papers Sultana *et al.* [153] and Meisheri *et al.* [154, 155] investigated the use of DRL methods for a grocery supply chain with a large number (up to 1,000 [153]) of perishable products. Wastage was determined as a proportion of stock, rather than tracking the age profile of each product. The problem was framed as a multi-agent task, with separate agents placing replenishment orders for each product. The orders were then modified to comply with capacity constraints. This approach enabled the system to be used, without retraining, when products were added or removed [154]. In all three papers, the DRL methods gave better results than a heuristic policy with a target inventory

level for each product. Meisheri *et al.* [155] specifically considered the effect of forecast quality of DRL performance and found that performance increased as the correlation between a one-step ahead forecast and actual demand increased, matching a linear programming approach with future knowledge of all demand when the one-step ahead forecast was perfect.

Wang *et al.* [156] used two deep learning methods, DQN and A2C, to find policies that set both an order quantity and price at each timestep for a perishable product. Demand for the product was stochastic and depended on the current price. When the useful life was two periods and tabular Q-learning was tractable, both the DQN and A2C methods achieved higher mean profits. DQN and A2C also produced effective policies as the product lifetime increased to three and four, for which the authors reported that tabular Q-learning was "inefficient and impractical" due to the exponential increase in state space with useful life.

Abu Zwaida *et al.* [157] applied DQN to a simulation of a drug replenishment problem in a hospital, considering a product mix of up to 70 different perishable drugs subject to different levels of demand and with different storage requirements. As in other studies, the authors motivated the use of DRL by reference to the large state-action space. The DQN policy incurred lower ordering costs and suffered fewer shortages than three benchmark policies.

De Moor *et al.* [40] demonstrated the benefits of using potential-based reward shaping [158] when training a DQN model to make replenishment ordering decisions for perishable goods. A heuristic ordering policy, such as base stock or BSP-low-EW [42], was used as a 'teacher' to shape the reward used during the DQN training process. This enabled knowledge to be transferred from a simple policy that performed reasonably well to the DRL agent. Using a teacher stabilized the training process, leading to better policies than the unshaped version of DQN on average and making the training process less dependent on hyperparameter tuning. In one set of experiments, in which the BSP-low-EW teacher was not an optimal policy but outperformed DQN agents trained without reward shaping, the use of reward shaping led to a DQN policy that outperformed the teacher. This demonstrates the

potential for improved performance by using reward shaping with a strong heuristic.

Taken together, these studies suggest that DRL methods are a sensible approach to take with perishable inventory problems and deal effectively with the large state spaces needed to represent stock with different ages. However, they all use relatively simple simulated datasets and do not include some of the specific complexities that arise with the inventory management of blood products including periodic demand and substitution between products.

In parallel with my work on this thesis, other researchers have investigated managing multiple perishable products with no substitution [159, 160] and consumer-driven substitution [161] in the grocery supply chain, single perishable products in a pharmaceutical supply chain [162] and managing a single perishable product when both which supplier to order from and order quantity must be determined at each decision point [163].

2.6.2 Applications to blood product inventory

While I have not identified any studies that use DRL to find good policies for replenishing blood product inventory in a hospital blood bank, there are a small number of studies that formulate the management of blood product inventory as an MDP and use ADP, which can be considered a RL method, to find replenishment policies [164] or good issuing policies while keeping the replenishment policy fixed [63, 70].

Abdulwahab and Wahab [70] investigated policies for allocating platelets in a hospital blood bank following a similar setup to that proposed for PRBCs by Powell [111]. They found that a policy that incorporated information about the future, in the form of a piece-wise linear approximation of the value function, outperformed a myopic policy that just considered the reward from the next transition step. Dumkreiger [63] used several different methods to find policies for allocating PRBCs to meet patient demand including estimating Q-values based on the rollout of a base policy, approximate policy iteration using XGBoost [165] regression models to estimate Q-values and approximate policy iteration using an XGBoost classification model to model the policy directly. She found that direct policy

approximation using an XGBoost classification model achieved lower wastage due to expiration in both a high and low demand setting than the default policy based on current operational practice at the Mayo Clinic, Minnesota, USA. Both of these studies considered stochastic demand and the possibility of substitution between the eight different ABO/RhD blood types. While the reward function used by Dumkreiger [63] treats any acceptable blood type type between the unit and the patient as the same, the reward function used by Abdulwahab and Wahab [70] favours exact matches and penalizes the use of universal donor units[1] in patients who do not require them. Dumkreiger [63] also incorporated the fact that incoming inventory will not all be of the same age by sampling the age of incoming inventory from mixture distributions based on observed deliveries.

In parallel to my work on this thesis Abouee-Mehrizi *et al.* [164] used ADP, representing the value function as a linear combination basis functions, to find replenishment policies for platelets in a hospital blood bank. They considered two aspects of reality that make the problem more computational demanding: that platelet units do not arrive fresh and that the age distribution at arrival may depend on the order quantity. They were unable to compute the optimal policy using value iteration for a realistic maximum useful life of 5 days when considering these two challenges, due to the additional computational demands, and therefore investigated the performance of the ADP approach. The replenishment policies fit using ADP incurred lower levels of stock holding, expiries and shortages than were observed in reality when tested on held-out, data from a hospital in Canada and performed at least as well as policies computed using value iteration under the assumption that all stock arrives fresh over a range of notional cost settings. I consider this scenario, as originally presented in Mirjalili [166], in Chapter 4 as an example where GPU-accelerated value iteration can be used to compute the exact policy for large problems incorporating challenging aspects of reality that have recently been described as infeasible or impractical.

Additionally, in 2024 Mohamadi *et al.* [167] used A2C to learn a policy for

[1]Incorrectly stated in Abdulwahab and Wahab [70] be O- instead of AB+ for because the matching rules for PRBCs are used instead of those for plasma and platelets.

allocating the stock of PRBCs held at a regional blood centre between two hospitals, using a vendor-managed inventory approach where the supplier had access to the stock holding information and demand levels at the hospitals. The policy learned using A2C achieved higher service levels and lower wastage on simulated scenarios than the current practice in Iran, where each hospital places its own replenishment orders.

DRL has also been used to support decision making about other aspects of the blood supply chain. Li *et al.* [168] considered the problem of blood component preparation and used approximate dynamic programming with piece-wise linear function approximation to find policies for converting whole blood into different components given stochastic donations and stochastic demand while Altaf *et al.* [169] used DRL to optimize the placement of blood distribution centres and delivery routes from the distribution centres to hospitals.

2.7 Research directions

Research in the last few years has demonstrated that RL may be a good general purpose approach for different inventory problems [147, 170, 171]. In a 2022 review considering RL for inventory control, Boute *et al.* [106] suggested two areas for future research that are particularly applicable to blood inventory. The first is "*leveraging big data to enrich the state space*": EHRs provide a great deal of information about the current state of a hospital that is available in near real-time, and previous studies have already shown that this information can be used to generate better forecasts [61, 62] and improve ordering decisions [11]. The second is to address problems where "*multiple actions need to be taken simultaneously*": when hospital blood banks make orders they need to specify the number of units of each product type, rather than just a total number, and so addressing this problem would make the RL policy more suitable for real-world use.

The size of perishable inventory management problems for which value iteration is considered to be feasible does not appear to have changed very much over the last 40 years despite significant developments in computational power.

The updates for each state in one iteration of the value iteration algorithm can be computed in parallel, so taking advantage of parallel computation (either in CPUs or GPUs) may provide a way to extend value iteration to larger problems to better understand how both heuristic and approximate methods, including RL, perform as the problem size increases.

Additionally, Dumkreiger [63] identified that relatively little attention has been paid to issuing policies. It is common to use a First-in First-out (FIFO) or OUFO issuing policy but there may be situations where those policies lead to suboptimal results, particularly when managing multiple products with complex substitution relationships. Therefore, it would be beneficial to be able to compare different issuing policies, and to be able to jointly optimize replenishment and issuing policies.

2.8 Familiarization with real-world practice and data

As part of the on-boarding work for this thesis I shadowed biomedical scientists at the transfusion laboratory of our partner site, UCLH, to understand their daily workflow and inventory management practices. I also conducted a descriptive analysis of requests for blood products by clinicians and stock holding information for PRBCs and platelets. This work is presented in Appendix B. The data for platelets was subsequently used to calculate simulation inputs and to train and evaluate replenishment policies in Chapters 3 and 5, and to train a supervised learning model to predict whether requested platelet units would be transfused in Chapter 5.

Chapter 3

Platelet replenishment using reinforcement learning

3.1 Motivation

In this chapter I investigate whether DRL methods can be used to learn effective policies for platelet replenishment. One of the key challenges with platelet inventory management, in common with other perishable inventory problems, is the need to consider the age profile of the stock and the resulting large state space. DRL methods were specifically designed to learn policies for acting in large state spaces, but there have been limited applications in perishable inventory management and, to the best of my knowledge, none for the replenishment of blood products. Therefore, a first step to answering my broader research question is to test these methods on a simple platelet replenishment task and compare their performance to more established approaches.

I selected the scenario from Rajendran and Srinivas [45] because it is a recent study set in a hospital blood bank which evaluates multiple heuristic replenishment policies and included sufficient information to reimplement the scenario and the methods. A major advantage of starting with a problem that has been previously studied in the literature is that it provides an existing approach (or set of approaches) against which the performance of DRL methods can be compared. In the problem they address, the demand for platelets is stochastic and the demand distributions

incorporate the weekly periodicity in demand that is consistent with the UCLH data (see Appendix B) and descriptive analysis presented in other studies including Guan *et al.* [11] and Motamedi *et al.* [61].

Additionally, it is common to fit and evaluate platelet replenishment policies using simulated data, often using observed data to set the parameters of the demand probability distributions. Evaluating the policies on real, observed demand trajectories may provide a more realistic estimate of performance, in particular accounting for the fact that future demand patterns may differ from historical demand patterns. I therefore extended the analysis by fitting policies using observed demand data from our partner hospital, UCLH, and evaluating them on a subsequent year of demand data.

By implementing the scenario as an RL environment it is possible to both train RL agents and evaluate any policy that can be implemented with Python. I used open-source Python libraries to implement the RL environment and therefore other research groups will be able to build on this work and easily evaluate their own policies on the same problem. This may form part of a future benchmark for comparing different methods on the same problem, and avoid the repeated effort of building similar simulations that is common in the field.

3.2 Contribution statement

The main contributions of this chapter are:

- implementing an RL environment to represent a platelet inventory management problem in the literature, using a standard interface;
- applying standard DRL algorithms to learn platelet replenishment policies, and comparing their performance to heuristic policies applied to the same problem in the literature;
- developing a CPU-based parallel implementation of Q-value iteration to find the optimal replenishment policy and benchmark the performance of the DRL and heuristic policies on the simulated scenario;

- using real demand data from a UK hospital to perform temporal validation of the methods, estimating how they would have performed if trained on historical data and applied prospectively.

An extended abstract based on this work, including a figure based on Figure 3.1, was published in a special issue of *Blood* for work presented at the 65th Annual Meeting of the American Society of Hematology [172].

3.3 Background and related work

The key background and related work for this chapter is set out in Sections 2.4 to 2.6. As I describe in Section 2.4.4, current practice for platelet replenishment is often based on simple heuristic replenishment policies with parameters set based on expert knowledge and experience. In Section 2.4.1 I explain how replenishment policies have been determined in the literature including using dynamic programming approaches to compute the optimal policy for down-sized problems (Section 2.4.1.1), and fitting heuristic policy parameters using simulation optimization (2.4.1.2) or stochastic mixed integer linear programming (2.4.1.3).

I explain in Section 2.5.2 how DRL methods have been developed to deal with large state spaces, and provide an overview of research applying DRL to managing perishable inventory in Section 2.6. At the time this work was conducted, a small number of studies had investigated the use of RL [150] and DRL [152, 153, 154, 155] for perishable inventory management, all focused on retail supply chains. Sun *et al.* [152] did not include a comparison to any standard heuristic replenishment policies, while Sultana *et al.* [153] and Meisheri *et al.* [154, 155] considered a grocery supply chain with a large number of products and calculated wastage as a fixed proportion of stock holding instead of tracking the age profile of the stock. To the best of my knowledge, this chapter is the first attempt to model a platelet inventory management problem as an RL environment and fit replenishment policies using DRL methods.

3.4 Experiments using simulated data

3.4.1 Problem description

I adopt the platelet replenishment problem described by Rajendran and Srinivas [45], in which a hospital blood bank can place a daily replenishment order with a regional blood centre. This is a single-product, single-echelon, periodic-review perishable inventory problem. In this problem it is assumed that the single product can meet demand for any patient and therefore substitution is not considered.

Total daily demand from patients is simulated by using day-of-the-week specific Poisson distributions and filled using an OUFO issuing policy so that the oldest unexpired units in stock are issued first. Orders to the regional blood centre are placed at the end of a day, and received at the start of the next day. Normal daily orders incur both a variable order cost per unit, and a fixed order cost if an order is placed. An expensive rush order must be placed if there is a shortage of units. A wastage cost is imposed for units that expire before use and a holding cost is imposed for unexpired units held in stock at the end of each day. Any platelet unit is assumed to be capable of satisfying any demand. The settings of the simulation are set out in Table 3.1 - these are the settings of the baseline scenario used in Rajendran and Srinivas [45]. The daily platelet demand figures are originally from a study at Stanford Medical Centre, California, USA by Guan *et al.* [11] while the costs were originally obtained from a regional medical centre in Pennsylvania, USA by Rajendran and Ravindran [54].

The objective is to find an ordering policy that minimizes the expected ongoing cost of operations.

3.4.2 Constructing the reinforcement learning environment

I converted the scenario described above into an MDP and then constructed an RL environment to represent the problem using the OpenAI Gym [139] Python library.

To support comparison between different problems, I have adopted a single notation for all of the scenarios considered in this thesis instead of using the notation from the original research papers.

Parameter		Value
Mean of Poisson distribution for demand (units), μ^{τ}		
	Monday	37.5
	Tuesday	37.3
	Wednesday	39.2
	Thursday	37.8
	Friday	40.5
	Saturday	27.2
	Sunday	28.4
C_f Fixed order cost ($/shipment)		225
C_v Variable order cost ($/unit)		650
C_h Holding cost ($/units/day)		130
C_s Emergency procurement (shortage) cost ($/unit)		3,250
C_w Wastage cost ($/unit)		650
m Useful life on receipt (days)		3

Table 3.1: Key assumptions and inputs for the baseline platelet replenishment scenario, from Rajendran and Srinivas [45]

The state, S_t comprises two components: $\tau \in \{0, 1, ..., 6\}$, representing the day of the week and the units in stock at the end of the day $\underline{\mathrm{X}_t} = [X_{m-1,t}, X_{m-2,t}, ..., X_{1,t}]$ ordered by ascending age from left to right. In the baseline case where the useful life on receipt, m, is three days, there will be two values in the state vector relating to stock currently held: $X_{2,t}$ for stock received at the start of the current day and $X_{1,t}$ for stock received at the start of the previous day. Any stock that was received two days ago that has not been used before the end of day t expires and will be wasted. I assume that orders placed on the evening of day t are received on the morning of day $t + 1$ and therefore no orders in transit are included in the state.

The action, A_t, is the number of units ordered at the end of day t. These units are received and ready to be issued at the start of day $t + 1$. The actions are constrained to be positive integers, up to the maximum order value, $A_{\max}$: $A_t \in \mathbb{A} = \{0, 1, 2, ..., A_{\max}\}$.

In Equation 3.1 I set out how the elements of the state are updated for successive days, and in Equations 3.2 to 3.4 I set out how the number of wasted

units, the number of units held overnight, and the number of units that must be ordered on an emergency basis due to a shortage are calculated based on the demand that must be met during day t, D_t, when all stock arrives fresh. Units with a remaining useful life of 1 day at the start of a day will expire at the end of that day if unused.

$$\begin{aligned}
\tau_{t+1} &= (\tau_t + 1) \mod 7 \\
X_{j,t+1} &= \left[X_{j+1,t} - \left[D_{t+1} - \sum_{k=1}^{j} X_{k,t} \right]^+ \right]^+ \quad \forall j \in \{1, ..., m-2\} \\
X_{m-1,t+1} &= \left[A_t - \left[D_{t+1} - \sum_{j=1}^{m-1} X_{j,t} \right]^+ \right]^+
\end{aligned} \tag{3.1}$$

$$W_{t+1} = [X_{1,t} - D_{t+1}]^+ \tag{3.2}$$

$$B_{t+1} = [B_t + A_t - D_{t+1} - W_{t+1}]^+ \tag{3.3}$$

$$E_{t+1} = [D_{t+1} - B_t - A_t]^+ \tag{3.4}$$

The reward, R_t, is the negative total cost for the day, including the fixed and variable order cost incurred by the previous day's action, holding costs and wastage costs based on units still in stock at the end of the current day, and shortage costs based on the number of emergency rush orders required during the current day. It is set out in Equation 3.5.

$$R_{t+1} = -C_f \mathbb{1}_{A_t > 0} - C_v A_t - C_h B_{t+1} - C_w W_{t+1} - C_s E_{t+1} \tag{3.5}$$

The only stochastic element contributing to the transition probabilities, $p(S_{t+1}|S_t, A_t)$, is the demand D_{t+1}.

Note that in this scenario, where orders are placed at the end of the day, the demand for day t, D_t, precedes the observation of S_t, and the wastage W_t, holding

B_t and shortage E_t for day t are calculated before the observation of S_t. In later chapters, when the replenishment order is placed in the morning, demand D_t arises after the observation of state S_t.

In Figure 3.1 I illustrate two possible steps through the environment, showing how the state is updated and the reward is calculated. In one example there is sufficient inventory to meet demand and some units expire, in the other there is a shortage.

Due to the order of events in an MDP, the reward function is subtly different from the objective function in Rajendran and Srinivas [45]. In Equation 3.5 the ordering costs included in the reward from day t are those from day $t-1$, because the order quantity A_t is only known after the agent makes a decision having received S_t and R_t at the end of day t. This limitation does not exist in the stochastic mixed integer linear programming formulation in Rajendran and Srinivas [45], and therefore the ordering costs included in the cost function from day t are those from day t. This is a timing difference and, since I compare the models based on undiscounted reward, long term costs would be the same under both formulations.

I developed a custom Python class to represent the environment: a subclass of the `gym.Env` class from OpenAI Gym, called `PlateletBankGym`. By creating a subclass of `gym.Env`, an established standard format for representing a generic MDP can be extended with additional information to capture the specific logic of this scenario. Initialization arguments allow the environment to be customized by setting the maximum order quantity, the maximum useful life on receipt, the lead time, the distribution of remaining useful life on receipt, the value of the cost function components and whether the state breaks down the stock by age. Seeds can be provided to ensure that the random processes are reproducible, and there is an option to include information about whether the end of an episode was caused by truncation in the `info` dictionary returned after each step in the environment.

The `PlateletBankGym` must also be passed an instance of a custom `DemandProvider` class. At each step through the environment, the `DemandProvider` provides the demand for platelets and, optionally, additional information to

be included in S_t. The `DemandProvider` written for the baseline scenario, `PoissonDemandProviderSR`, generates demand by sampling from day-of-the-week specific Poisson distributions parameterized using the values in Table 3.1 and keeps track of the current weekday which is included in S_t. This modular approach makes it easy to alter how the demand is generated, by creating a new `DemandProvider` subclass, without needing to change any of the core logic of the rest of the environment. In the experiments using data from UCLH, I used a subclass of `DemandProvider` which sampled trajectories from a supplied comma separated value (CSV) file containing historical demand figures.

3.4.3 Hardware

The stochastic mixed integer linear programming experiments were conducted on CPU nodes of the UCL high performance computing cluster Myriad, using 8 CPU cores and 32GB random-access memory (RAM). The DRL and Q-value iteration experiments were conducted on a desktop computer running Ubuntu 20.04 LTS via Windows Subsystem for Linux on Windows 11 with an AMD Ryzen 9 5900X processor, 64GB RAM, and an Nvidia GeForce RTX 3060 GPU.

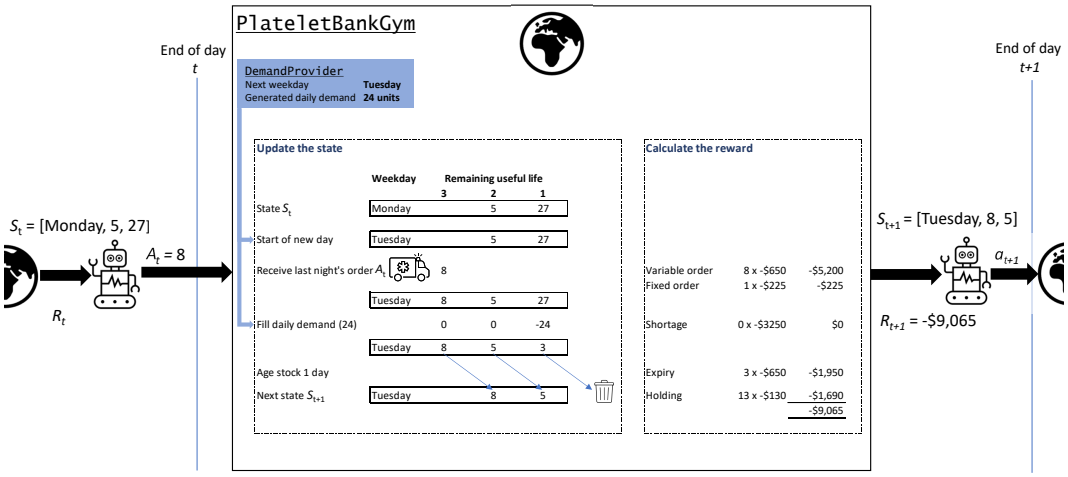
3.4.4 Code availability

The code supporting the work in this chapter (both the the experiments with simulated demand described in Section 3.4 and the experiments with real demand described in Section 3.5) is available at https://github.com/joefarrington/bloodbank_rl.

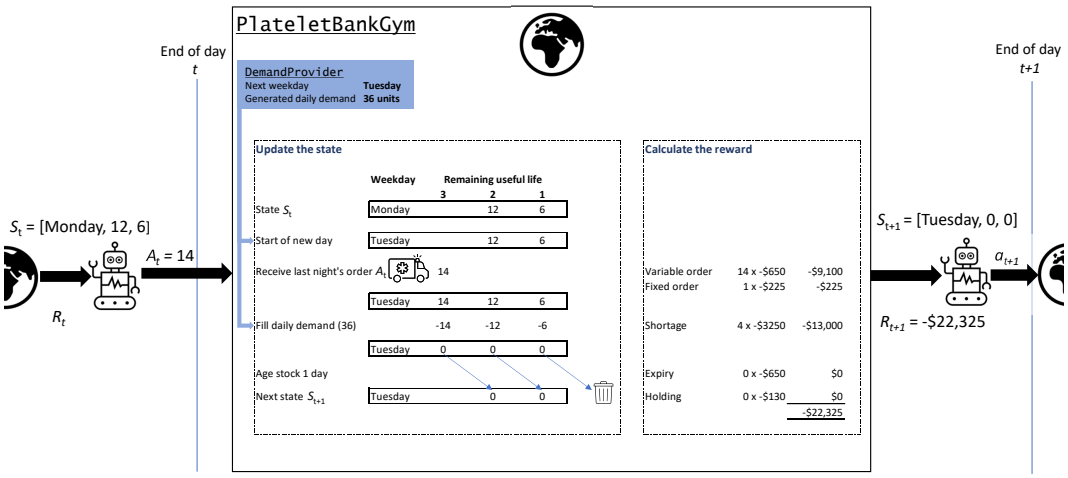
3.4.5 Heuristic policies

Rajendran and Srinivas [45] report the performance of four heuristic policies: $(\mathtt{s}, \mathtt{S})^{\dagger}$[1], $(\mathtt{s}, \mathtt{Q})$, $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$. The ordering rules for the four policies

[1] In this chapter, I use the definition of the $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy from Rajendran and Srinivas [45], in which $\mathtt{s} < \mathtt{S}$ and an order is only placed when the inventory position is less than $\mathtt{s}$. This means that an order quantity of 1 cannot be placed. In subsequent chapters where an $(\mathtt{s}, \mathtt{S})$ policy is used, I follow the definition from Snyder and Shen [28] in which an order can be placed when the inventory position is less than or equal to $\mathtt{s}$. This allows an order quantity of 1 to be placed. I use a superscript dagger to distinguish these policies.



(a) An example step through the RL environment where expiry and holding costs are incurred.



(b) An example step through the RL environment where shortage costs are incurred.

Figure 3.1: Example steps through the RL environment for platelet replenishment.

are set out in Equations 3.6 to 3.9.

$$(\mathtt{s}, \mathtt{S})^{\dagger} \qquad A_t = \begin{cases} 0 & \text{if } B_t \geq \mathtt{s} \\ \mathtt{S} - B_t & \text{if } B_t < \mathtt{s} \end{cases} \tag{3.6}$$

$$(\mathtt{s}, \mathtt{Q}) \qquad A_t = \begin{cases} 0 & \text{if } B_t \geq \mathtt{s} \\ \mathtt{Q} & \text{if } B_t < \mathtt{s} \end{cases} \tag{3.7}$$

$$(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q}) \qquad A_t = \begin{cases} 0 & \text{if } B_t \geq \mathtt{s} \\ \mathtt{Q} & \text{if } \alpha \leq B_t < \mathtt{s} \\ \mathtt{S} - B_t & \text{if } B_t < \alpha \end{cases} \tag{3.8}$$

$$(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q}) \qquad A_t = \begin{cases} 0 & \text{if } B_t \geq \mathtt{s} \\ \mathtt{S} - B_t & \text{if } \beta \leq B_t < \mathtt{s} \\ \mathtt{Q} & \text{if } B_t < \beta \end{cases} \tag{3.9}$$

All four of these heuristic replenishment policies are periodic review policies where the review period is 1 day. The $(\mathtt{s}, \mathtt{S})^{\dagger}$ and $(\mathtt{s}, \mathtt{Q})$ policies are standard heuristic policies from the operational research literature. In both cases an order is made if the stock level is below the reorder point $\mathtt{s}$ when the ordering decision is made. For the $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy, the order quantity is the difference between the order-up-to level $\mathtt{S}$ and the current inventory position. For the $(\mathtt{s}, \mathtt{Q})$ policy the order quantity is a fixed value $\mathtt{Q}$. The $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policies were developed by Rajendran and Srinivas [45]. In these policies α and β are additional thresholds, less than the reorder point $\mathtt{s}$, below which the method for determining the order quantity changes.

I intended to use the parameters reported in the original paper as the benchmark against which to compare DRL policies. The performance of the heuristic policies with these parameters as evaluated using my RL environment was not consistent with the performance reported by Rajendran and Srinivas [45] and, from a review of the costs, I determined that it may be possible to find better parameters for the

heuristic policies. I therefore decided to optimize the heuristic policy parameters myself, to ensure that I had a reliable benchmark against which to compare the performance of alternative approaches. I describe my work evaluating the parameters from the original paper using my RL environment, and discuss the discrepancies between the cost components, in Appendix C.

3.4.5.1 Methods

I reimplemented the stochastic mixed integer linear programming approach to fitting the parameters for the four heuristic policies described by Rajendran and Srinivas [45] and set out in Equations 3.6 to 3.9 above.

A mixed integer linear program is an optimization problem in which the objective function and the constraints are described by equations that are linear in the decision variables and at least one of the decision variables is constrained to be integer. If it is important account for stochasticity in some variables outside our control, this can be addressed by constructing a stochastic mixed integer linear programming model and simultaneously optimising over multiple possible scenarios, each with a randomly sampled trajectory for these stochastic variables. A subset of the decision variables, those that parameterize the policy of interest, are constrained to be the same across all the scenarios. The objective function is the expectation of the single-scenario objective function over the different scenarios: we want to find the set of policy parameters that has the lowest expected cost given the uncertainty.

In this task, it it important to consider stochasticity in the demand for platelet units, and find parameters for each of the four policies that minimize the expected cost given this uncertainty. As in Rajendran and Srinivas [45], I used 20 different scenarios, each with a different trajectory of daily demands sampled from the day-of-the-week specific Poisson distributions set out in Table 3.1. The same 20 scenarios are used to optimize each of the four different policies. Each scenario begins on a Monday and is 30 days long.

The objective function and constraints are set out in Appendix D. Except where noted below, these are reproduced from Rajendran and Srinivas [45] using their

notation. The constraints enforce the rules of the problem as described in Section 3.4.1: the number of units received on day $t + 1$ is the number ordered on day t, demand is filled using the oldest units first, and an order is placed if the inventory position is below the reorder threshold $\mathtt{s}_t$. Additionally, $\mathtt{s}_t$ must be strictly less than $\mathtt{S}_t$ and, in the policies that use them, α_t or β_t must be strictly less than $\mathtt{s}_t$.

I modified the constraints on the policy parameter decision variables ($\mathtt{s}_t$, $\mathtt{S}_t$, $\mathtt{Q}_t$, α_t and β_t) to make it explicit that I want to learn one parameter for each weekday instead of each timestep. This is because, as set out in Table 3.1, demand is based on day of the week.

Additionally, I introduced new binary decisions variables and corresponding constraints, based on those from Pauls-Worm *et al.* [173], to ensure that the OUFO issuing policy was correctly enforced. These modifications are described in detail in Appendix D.4.

I constructed the basic mixed integer linear program for a single scenario in Python using Pyomo [174, 175]. I used mpi-sppy [176] and the commercial solver Gurobi [177] to convert this to a stochastic programming problem with 20 scenarios and solve the extensive form. I allowed each optimization problem to run for a maximum of 36 hours.

I ran a series of tests on the output of the optimization process to check that for each policy and each scenario the output was consistent with my understanding of the problem set-up. This allowed me to confirm that the constraints had been correctly adapted from the mathematics of the paper to the Pyomo modelling syntax and, in the situation described above for the OUFO policy, that the constraints fully encapsulated the problem as described.

I simulated 1,000 trajectories, each 365 days long, of platelet demand using the day-of-the-week specific Poisson distribution means in Table 3.1. Each year starts on a Sunday evening with no inventory in stock. An agent for each of the four policies was applied to these trajectories, and the daily rewards (and reward components) were recorded. I used the same 1,000 demand trajectories, which I refer to as the "evaluation episodes", to evaluate the performance of alternative

policies in Sections 3.4.6 and 3.4.7 and Appendix C.

3.4.5.2 Results

I present the parameters found for each policy using the stochastic programming approach in Table 3.2, and the mean daily costs incurred by each policy parameterized using those values over the 1,000 evaluation episodes in Table 3.3. The daily cost (in total, and for each component) for an individual episode is the mean cost per day: the cost incurred divided by the the length of the episode. In this thesis, I refer to this metric as the "daily cost" for an individual episode, and use the term "mean daily cost" to refer to the mean of the metric over multiple episodes.

Policy	Parameter	**Weekday**						
		Mon	Tue	Wed	Thu	Fri	Sat	Sun
$(s, S)^\dagger$	s	27	27	19	49	23	21	24
	S	49	45	47	50	37	37	49
(s, Q)	s	45	27	30	34	38	27	28
	Q	69	32	43	45	41	29	90
(s, S, α, Q)	s	38	32	35	39	31	29	49
	S	43	45	46	48	33	33	82
	α	5	16	9	10	9	3	10
	Q	48	34	41	33	26	26	31
(s, S, β, Q)	s	53	43	22	28	33	18	24
	S	54	44	45	50	34	37	49
	β	13	12	10	11	5	7	6
	Q	44	41	45	45	35	31	46

Table 3.2: Heuristic replenishment policy parameters from my reimplementation of the stochastic programming methods in Rajendran and Srinivas [45]

Policy	**Mean daily cost**					
	Fixed	Variable	Holding	Shortage	Wastage	Total
$(s, S)^\dagger$	223	22,863	1,246	922	0	**25,254**
(s, Q)	157	22,787	3,002	2,332	122	28,401
(s, S, α, Q)	203	22,850	1,825	1,144	0	26,023
(s, S, β, Q)	224	22,826	1,257	1,108	0	25,414

The daily cost components and daily total cost were calculated for each episode. The mean of the daily cost components and daily total cost over the 1,000 evaluation episodes is reported. The lowest mean daily total cost is indicated in **bold**. The heuristic policy parameters are set out in Table 3.2.

Table 3.3: Mean daily costs incurred using heuristic replenishment policies with parameters fit using stochastic programming.

3.4.5.3 Discussion

There are large reductions in mean daily cost when comparing the policies using the parameters I found, in Table 3.3, to the policies using the parameters reported in Rajendran and Srinivas [45], in Table C.2. This is driven by lower shortage costs for all of the policies.

In Table 3.3 it is clear by inspecting the values for the mean daily fixed cost that the two best performing policies, $(\mathtt{s}, \mathtt{S})^{\dagger}$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$, now place an order almost every day, which is what I would expect from a good policy given the cost structure. This is caused by an increase in the values of the reorder threshold parameter, $\mathtt{s}$, which can observed across all of the policies.

Counterintuitively, the order quantity parameter $\mathtt{Q}$ for the $(\mathtt{s}, \mathtt{Q})$ policy is higher than the order-up-to level parameter $\mathtt{S}$ for the $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy for three of the seven weekdays and is particularly high on Sunday and Monday. This is likely to contribute to the comparatively large holding, shortage and wastage costs incurred by the $(\mathtt{s}, \mathtt{Q})$ policy.

Interestingly, while Rajendran and Srinivas [45] report that $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ consistently outperform the simpler policies across multiple settings, I find here that $(\mathtt{s}, \mathtt{S})^{\dagger}$ achieves the lowest mean daily cost. Both an $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and an $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policy are capable, through specific parameter settings, of matching the decisions of an $(\mathtt{s}, \mathtt{S})^{\dagger}$ or $(\mathtt{s}, \mathtt{Q})$ policy. Therefore, in principle, they should perform no worse than the better of those two policies. However, $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policies both require additional constraints and integer decision variables, making them more complex problems to solve. This can be observed in the original paper where the authors report than it took almost six times longer to solve for $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ than for $(\mathtt{s}, \mathtt{S})^{\dagger}$ and $(\mathtt{s}, \mathtt{Q})$. The additional constraints I introduced to enforce the OUFO issuing policy increase the difficulty of all the optimization problems.

Given sufficient additional time or computational resources I may have found parameters such that $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ achieved equal or lower mean daily total cost than $(\mathtt{s}, \mathtt{S})^{\dagger}$. It is also possible that, given the relatively small number

of scenarios considered, this could have led to 'overfitting' by the two policies with additional parameters: achieving a lower expected cost in the optimization problem by finding parameters that work well on the 20 scenarios used to fit the parameters, but not generalizing well to new demand trajectories sampled from the same distribution.

Overall, the policies with parameters I found by reimplementing the stochastic programming methods achieve lower mean daily total costs and make ordering decisions that more closely match my understanding of the problem based on the cost function. They therefore appear to be a more appropriate benchmark against which to compare the performance of my DRL models.

3.4.6 Deep reinforcement learning policies

DRL methods have been shown to perform well as a general approach for solving inventory management problems, in addition to a wide range of other tasks. Like the stochastic mixed integer linear programming approach, it is possible to learn a policy based on samples from the demand distribution. Unlike the stochastic mixed integer linear programming approaches, it is not necessary to specify a functional form for the policy in advance.

3.4.6.1 Methods

I trained policies using two common DRL approaches: DQN [178] and PPO [114] on my platelet replenishment RL environment using the implementations from the Python library Tianshou [179]. During training, the `DemandProvider` in the environment randomly generated demand values using the day-of-the-week specific Poisson means presented in Table 3.1. Unlike when fitting the stochastic programming models, I did not use a fixed number of demand trajectories sampled in advance: different demand trajectories were generated for each episode during training. The DRL approaches can efficiently learn from a larger number of example trajectories compared to stochastic programming methods, and dynamically sampling demand trajectories during training enabled me to fully leverage this advantage. This sampling process was controlled by a random seed to ensure

reproducibility.

The observation provided to the DRL agents at each step was based on S_t as set out in Section 3.4.2, but the integer representation of the weekday was replaced by a one-hot vector. A one-hot vector is used to represent categorical data using binary values: it a vector with a length equal to the number categories where all the values are set to zero except for the position corresponding to the category of interest, which is set to one. Here, for example, Monday was represented as $[1, 0, 0, 0, 0, 0, 0]$ and Wednesday was represented as $[0, 0, 1, 0, 0, 0, 0]$.

I selected DQN as the most widely used example of a DRL method that uses a neural network to estimate a value function, specifically the state-action values. To promote exploration by the agent, which is required to estimate the quality of different state-action pairs, I used ϵ-greedy exploration. At each step in the environment during training, the decision about whether to explore or not is taken using a sample from a Bernoulli distribution with probability of success $1 - \epsilon$. If the sample is successful, the action with the highest estimated Q-value for the current state is taken. If the sample fails the agent explores by selecting an action at random. The value of ϵ was reduced on a schedule throughout training. The training process for DQN is set out in Algorithm 1.

I wanted to compare the performance of a DRL method that estimates a value-function with a method that directly approximates a policy with a neural network. In initial experiments, I applied A2C [112] to the problem, but the results were relatively poor. I therefore used the PPO algorithm, which was recently found to learn successful policies on a range of problems from operational research by Hubbs *et al.* [147] using one common neural network architecture. PPO is similar to A2C in that it uses two neural networks, one to estimate the policy and another that is used to estimate the advantage of being in a state, but differs in that it uses Generalized Advantage Estimation (GAE) [180] to calculate the advantage based on the experience trajectories and uses an alternative objective function which is designed to limit policy updates to a "trusted" region. The goal of the change in objective function is to prevent performance collapse that can arise in on-policy

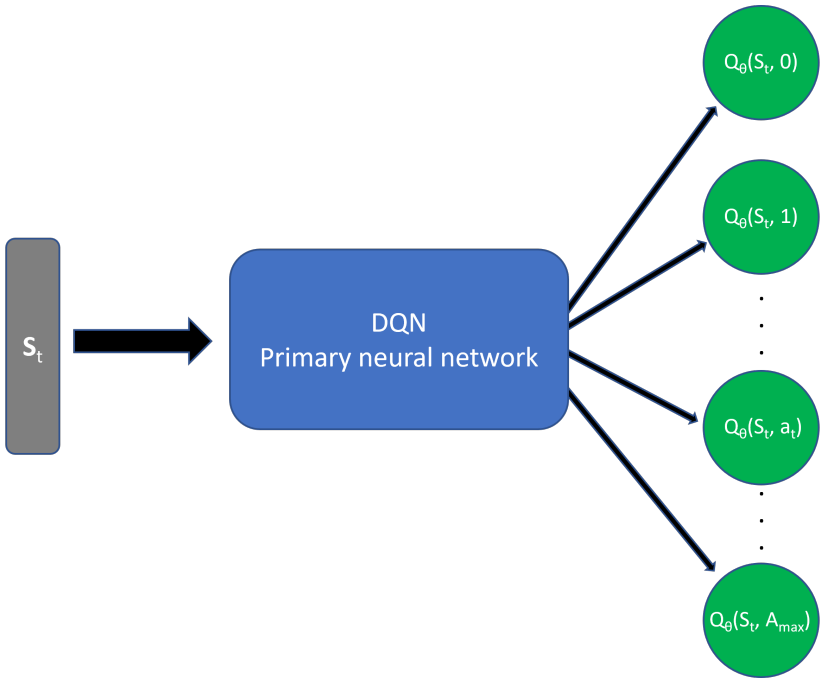
methods: a bad update to the policy can lead to the collection of bad experiences in the environment, which are then used to update the policy. During training, actions are sampled from a probability distribution parameterized by the output of the neural network. Exploration is encouraged by a term in the objective function that rewards a high entropy in this distribution. The training process for PPO is set out in Algorithm 2.

In order to initialize the output layers of the neural networks, I needed to specify the maximum order quantity, $A_{\max}$. This value was set to 60.

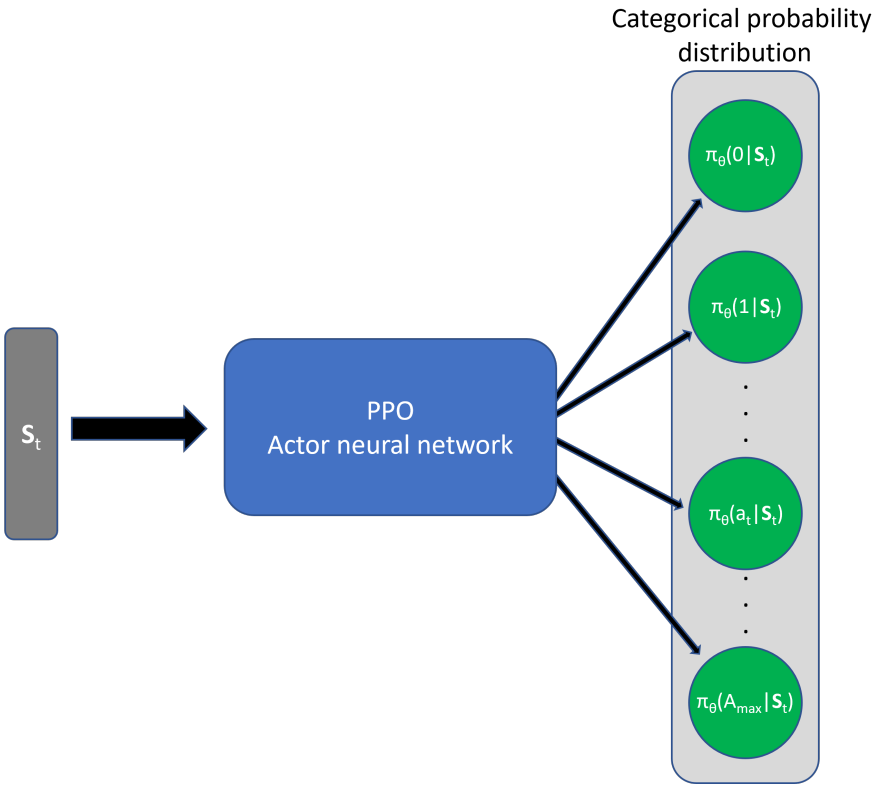
In Figure 3.2 I present a diagram showing the inputs and outputs from the neural networks used to determine the policy, either indirectly (in the case of DQN), or directly (in the case of PPO).

The training episodes were 365 days long. This represents an artificial truncation - there is actually nothing special about the terminal step and in reality the task would continue indefinitely. Conventionally, no bootstrapping is used when estimating the value of the terminal state because the are no future rewards. Pardo *et al.* [181] demonstrated that when the final step of an episode is not a "true" terminal state, but a truncation, the performance of policies can be improved by using bootstrapping based on the estimate of the value function for the state at the end of the episode. I incorporated code equivalent to `TimeLimit` wrapper from OpenAI Gym into my environment to indicate that my episodes are truncated and that rewards would be available after the final step. This enabled me to make use of the implementation of the partial-episode bootstrapping from Pardo *et al.* [181] in Tianshou. This is reflected in the pseudocode for Algorithm 1 and Algorithm 2.

For both methods, I periodically evaluated the performance of the current policy during training on separately seeded copies of the environment: validation environments. Model checkpoints were saved after each of these validation steps if the policy incurred a lower mean cost than on the previous validation step. I performed limited manual hyperparameter tuning based on the performance on the validation environments. After training, the policy associated with the the final model checkpoint saved from the training run was used to create an agent which



(a) Inputs and outputs for the primary network used in DQN.



(b) Inputs and outputs for the actor network used in PPO.

Figure 3.2: Inputs and outputs for the neural networks that are used to determine the policy in DQN and PPO.

was applied to the 1,000 evaluation episodes described in Section 3.4.5.1. When these policies took steps in the validation environments during training, and on the 1,000 evaluation episodes, I used deterministic versions of the DRL policies: always selecting action with highest state-action value for DQN or highest probability for PPO.

I used the same neural network architecture, a fully-connected network with three hidden layers of 128 units each, for the primary and target networks in DQN and for the actor and critic networks in PPO.

The hyperparameters and settings used to train the final DQN and PPO policies are set out in configuration files in Appendix E.

Algorithm 1 DQN with ϵ-greedy exploration and partial episode bootstrapping

Initialize replay buffer $\mathcal{D}$
Initialize primary network Q_θ and target network $Q'_{\theta'}$
Initialize discount rate γ
Initialize learning rate η
Initialize target network update frequency f
Initialize ϵ decay schedule C
Observe starting state s
for each iteration **do**
  for each collection step **do**
    Sample action $a = \begin{cases} \arg\max_a Q_\theta(s,a), & \text{with probability } 1-\epsilon \\ \text{random action}, & \text{with probability } \epsilon \end{cases}$
    Observe next state s', reward r and binary terminal indicator d
    Store experience tuple (s, a, r, s', d) in $\mathcal{D}$
    if d is True **then**
      Reset environment and observe new starting state s
    else
      Update the current state $s \leftarrow s'$
    end if
    Update epsilon $\epsilon \leftarrow C(\epsilon)$
  end for
  for each update step **do**
    Sample a random mini-batch from $\mathcal{D}$
    Calculate the loss function:
      $J(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{D})}\left[(r + \gamma Q'_{\theta'}(s', \arg\max_{a'} Q'_{\theta'}(s', a')) - Q_\theta(s,a))^2\right]$
    Update the primary network $\theta \leftarrow \theta - \eta\nabla_\theta J(\theta)$
  end for
  if iteration % f = 0 **then**
    Update the target network $\theta' \leftarrow \theta$
  end if
end for

Algorithm 2 PPO with partial episode bootstrapping

Initialize actor network π_θ and critic network $V_{\mathbf{w}}$
Initialize actor learning rate α
Initialize critic learning rate η
Initialize clipping threshold in actor loss function ϵ
Initialize discount rate γ
Initialize GAE decay factor λ
Initialize entropy exploration scaling factor β
Initialize trajectory size $|\tau|$
for each iteration **do**
  Initialize empty storage for trajectory τ
  Initialize empty storage to identify the last step for each episode U
  Initialize empty storage to identify steps that require bootstrapping M
  Observe starting state s_0
  for t in range(0, $|\tau|$) **do**
    Sample action a_t from $\pi_\theta(a_t|s_t)$
    Observe next state s'_t, reward r_{t+1}, terminal indicator d_t & truncation indicator c_t
    Append experience tuple $(s_t, a_t, r_{t+1}, s'_t, d_t)$ to trajectory τ
    $U_t = d_t \vee c_t$
    $M_t = \neg d_t \vee c_t$
    if d_t is True **then**
      Reset environment and observe new starting state s
      $s_{t+1} \leftarrow s$
    else
      Update the current state $s_{t+1} \leftarrow s'_t$
    end if
  end for
  for t in range($|\tau|$ - 1, 0, -1) **do**
    Calculate single step advantage, $\delta_t = r_{t+1} + M_t\left(\gamma V_{\mathbf{w}}(s'_t) - V_{\mathbf{w}}(s_t)\right)$
    Calculate advantage using GAE, $A_t^\pi = \delta_t + U_t\left(\gamma\lambda A_{t+1}^\pi\right)$
    Calculate critic target, $V_t^{\text{tar}} = V_{\mathbf{w}}(s_t) + A_t^\pi$
    Calculate likelihood of action under current policy, and detach gradients
      $\pi_{\theta_{old}}(a_t|s_t) = \pi_\theta(a_t|s_t)$
  end for
  for each update step **do**
    Calculate the critic loss function:
      $J(\mathbf{w}) = \mathbb{E}_t\left[\left(V_{\mathbf{w}}(s_t) - V_t^{\text{tar}}\right)^2\right]$
    Update the critic network $\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla_{\mathbf{w}}J(\mathbf{w})$
    Calculate the clipped actor loss function, including entropy bonus for exploration:
      $J(\theta) = -\mathbb{E}_t\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}}A_t^\pi, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}}, 1-\epsilon, 1+\epsilon\right)A_t^\pi\right)\right] +$
      $\beta\mathbb{E}_t\left[-\pi_\theta\left(a_t|s_t\right)\log\left(\pi_\theta\left(a_t|s_t\right)\right)\right]$
    Update the actor network $\theta \leftarrow \theta - \alpha\nabla_\theta J(\theta)$
  end for
end for

3.4.6.2 Results

I present the mean daily costs incurred by the policies learned using DQN and PPO over the 1,000 evaluation episodes, split by component, in Table 3.4.

Policy	Mean daily cost					
	Fixed	Variable	Holding	Shortage	Wastage	Total
DQN	225	22,986	1,618	317	0	25,146
PPO	225	22,979	1,480	340	0	**25,024**

The daily cost components and daily total cost were calculated for each episode. The mean of the daily cost components and daily total cost over the 1,000 evaluation episodes is reported. The lowest mean daily total cost is indicated in **bold**.

Table 3.4: Mean daily costs incurred using replenishment policies learned using DQN and PPO.

3.4.6.3 Discussion

Two different DRL methods learned policies that are competitive in terms of mean daily cost with the best policy found through stochastic programming, using minimal manual hyperparameter tuning. This is as expected, and other researchers have recently demonstrated the effectiveness of similar approaches on other replenishment problems for perishable inventory [40, 152, 156, 157].

To determine whether further work in the current simplified setting was likely to result in performance improvements, I sought to find the optimal policy for the MDP (Section 3.4.7) and then performed a more detailed comparison of the performance of the different policies from this section and Section 3.4.5, the results of which are presented in Section 3.4.8.

3.4.7 Q-value iteration policy

In Section 3.4.6 I demonstrated that DRL methods can find competitive policies for a simple platelet replenishment problem. After running these experiments, I decided to investigate whether I could find the optimal policy for the MDP in order to determine whether it would be worthwhile to consider alternative DRL approaches or additional hyperparameter tuning. In this section I find the optimal policy for the MDP using a dynamic programming approach: Q-value iteration.

3.4.7.1 Methods

I describe the different approaches to finding a policy using RL in Section 2.5.2. Dynamic programming techniques can be used to find the optimal policy for an MDP if the full specification of the MDP, including the state transition probabilities and reward function, is known and if the combined state-action space is sufficiently small for the algorithms to be computationally tractable [97, pp.74-90].

I used the Q-value iteration algorithm, adapted to take advantage of the periodic nature of the MDP in my problem, to find the optimal policy. In each iteration of the algorithm, I performed a sweep through every state and updated my estimate of the Q-value for that state using an iterative equation based on the Bellman optimality equation for state-action values, presented in Equation 3.10 [97, p.74].

$$Q_{j+1}(s,a) \leftarrow \sum_{s' \in \mathbb{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \max_{a \in \mathbb{A}} Q_j(s',a') \right] \tag{3.10}$$

In Equation 3.10 $Q_j(s,a)$ is my estimate of the state-action value of state s and action a after j sweeps through the state space. $p(s'|s,a)$ is the probability of ending up in state s' when starting is state s and taking action a. The reward for making that transition after taking action a is given by $R(s,a,s')$. This is the equivalent of the value iteration update presented in Equation 2.4, but for the state-action value function instead of the value function.

The MDP is periodic, following the days of the week in a cycle. This property can be used when determining whether the policy has converged. Q-value iteration is guaranteed to converge to the optimal state-action values for a finite MDP (or an infinite MDP with a discount factor) with an infinite number of iterations. With no periodicity, it is necessary to either set a fixed time limit or introduce a discount factor and test for convergence of the Q-value approximation on iteration j using Equation 3.11, with some small threshold $\epsilon > 0$. Introducing a discount factor into problem changes the ratio between the costs, and therefore I consider the

undiscounted case, with $\gamma = 1$, and seek to maximize the expected reward per week (which, in turn, minimizes the daily cost, which I report for easier with comparison with Rajendran and Srinivas [45]). In this setting, a convergence test based on that derived by Su and Deininger [182] for periodic MDPs can be used, which I set out in Equation 3.12. K is the period of the MDP, in this case seven for the seven days of the week.

$$\max_{s \in \mathbb{S}, a \in \mathbb{A}} |Q_j(s,a) - Q_{j-1}(s,a)| < \epsilon \tag{3.11}$$

$$\max_{s \in \mathbb{S}, a \in \mathbb{A}} |Q_j(s,a) - Q_{j-K}(s,a)| - \min_{s \in \mathbb{S}, a \in \mathbb{A}} |Q_j(s,a) - Q_{j-K}(s,a)| < \epsilon \tag{3.12}$$

These two conditions test different properties. The condition in Equation 3.11 is testing for convergence of the state-action values for each state-action pair, and when the condition is met the error in the estimate of the state-action value function (relative to the optimal state-action value function) is bounded by $\frac{\epsilon\gamma}{(1-\gamma)}$ [107]. The condition in Equation 3.12 doesn't provide any information about how close the current estimate state-action value function is to the optimal state-action value function. Instead, when it is met, future iterations may change the estimated value but won't (within some tolerance) change the ranking of actions within a given state and therefore the deterministic policy extracted based on the estimated state-action value function will not change.

I calculated the transition probabilities $p(s'|s,a)$ using the demand probability distribution of the following weekday. For each state s, I calculated the probability of observing each demand value, D' between 0 and $A_{\max}$, on the next weekday. I then looped through each valid combination of s, a and D' to determine the next state s' and the reward $R(s,a,s')$ if demand D' was observed. For any state s' that is not found to be a potential next state in this loop, $p(s'|s,a) = 0$. It is only possible to transition between state s and s' if the weekday of state s' follows the weekday of s. It is not, for example, possible to transition directly from a state where the

weekday is Monday to a state where the weekday is Thursday. This means that for a given state-action pair, I consider the transition to only $A_{\max} + 1$ states even though there are a total of $(7 \times A_{\max} \times A_{\max})$ states in the MDP.

Consistent with the DRL approach in Section 3.4.6, I set the maximum order quantity $A_{\max} = 60$. Additionally, because transition probabilities are only calculated for demand values between 0 and $A_{\max}$, the iteration procedure may not converge to the optimal policy if there is a significant probability mass for demand greater than $A_{\max}$. The highest mean demand is on Fridays, when the mean demand is 40.5 units. The cumulative distribution function for a Poisson distribution with $\mu = 40.5$ evaluated at 60 is greater than 0.998 and therefore the setting of $A_{\max}$ is not expected to have affected the optimal policy. To confirm this, I reperformed the analysis with $A_{\max} = 70$ (the cumulative distribution function for a Poisson distribution function with $\mu = 40.5$ evaluated at 70 is greater than 0.99999) and obtained the same policy.

I present the training algorithm for Q-value iteration in Algorithm 3.

The iterative update equation in Equation 3.10 is an embarrassingly parallel operation: it is easy to implement in parallel because each updated state-action value can be computed independently of the others. I therefore used the Python library joblib [183] to take advantage of parallel processing to compute the updates and reduce the wall time needed to run the algorithm, using an embarrassingly parallel for loop over the states. In Algorithm 3 I recompute the next states, probabilities and rewards on each iteration. In a serial implementation, these could be calculated once, at the start, and reused in each iteration. This was not possible in my simple parallel implementation due to memory constraints. My goal was not to write the most efficient parallel implementation of the algorithm, but to develop a version that would run in a reasonable wall time to provide a bound on cost incurred for the other methods.

To compare the performance of the policy identified using Q-value iteration with other policies, I created an agent that could take the tabular output of Q-values and, when state s is encountered in the environment, lookup the optimal action

$\arg\max_{a \in \mathbb{A}} Q(s, a)$. I applied this agent to the same 1,000 evaluation episodes used in the preceding sections and recorded the daily costs and cost components.

Algorithm 3 Q-value iteration for a periodic MDP

Initialize $Q(s, a) = Q_{old}(s, a) = 0 \quad \forall \quad a \in \mathbb{A}, s \in \mathbb{S}$
Initialize iteration counter $j = 0$
Initialize convergence threshold ϵ
Initialize binary convergence indicator, converged = False
Let $s' = T(s, a, D')$, a deterministic transition function given demand D'
Let s^k be the weekday component of state s
while not converged **do**
  for s in $\mathbb{S}$ **do**
    for a in $\mathbb{A}$ **do**
      Initialize $\mathbb{S}_{valid} = \emptyset$
      for D' in range(0, $A_{\max}$) **do**
        Calculate and store $s' = T(s, a, D')$
        Calculate and store $p(D'|s'^k)$
        Calculate and store $R(s, a, s')$
        $\mathbb{S}_{valid} \leftarrow \mathbb{S}_{valid} \cup s'$
      end for
      $Q(s, a) \leftarrow \sum_{s' \in \mathbb{S}_{valid}} p(D'|s'^k) \left[R(s, a, s') + \gamma \max_a Q_{old}(s', a')\right]$
    end for
  end for
  Save Q values from this iteration, $Q_j \leftarrow Q$
  $Q_{old} \leftarrow Q$
  if $j \geq 7$ **then**
    Load Q-values from a period ago, Q_{j-7}
    converged = $\max_{s,a} |Q_j(s, a) - Q_{j-7}(s, a)| - \min_{s,a} |Q_j(s, a) - Q_{j-7}(s, a)| < \epsilon$
  end if
  $j \leftarrow j + 1$
end while

3.4.7.2 Results

I present the mean daily costs incurred by the policy learning using Q-value iteration over the 1,000 evaluation episodes, split by component, in Table 3.5.

Policy	Mean daily cost					
	Fixed	Variable	Holding	Shortage	Wastage	Total
Q-value iteration	225	22,961	1,337	423	0	24,946

The daily cost components and daily total cost were calculated for each episode. The mean of the daily cost components and daily total cost over the 1,000 evaluation episodes is reported.

Table 3.5: Mean daily costs incurred by the replenishment policy computed using Q-value iteration.

From inspection of the Q-values, the policy can be represented as an $(\mathtt{s}, \mathtt{S})^\dagger$ policy with the parameters presented in Table 3.6.

Policy	Parameter	Weekday						
		Mon	Tue	Wed	Thu	Fri	Sat	Sun
$(\mathtt{s},\mathtt{S})^{\dagger}$	$\mathtt{s}$	45	47	46	49	34	35	46
	$\mathtt{S}$	48	50	48	51	36	38	48

Table 3.6: Policy parameters for the $(\mathtt{s},\mathtt{S})^{\dagger}$ policy equivalent to that computed using Q-value iteration

3.4.7.3 Discussion

The policy learned by Q-value iteration incurs the lowest mean daily cost on the 1,000 evaluation episodes of any of the policies tested in this section, which is consistent with my expectation that this method should identify the optimal policy. The mean daily cost incurred by the policy learned using PPO was only 0.3% higher than that incurred by the policy learned by Q-value iteration. Therefore my current DRL approaches are able to learn policies that are close to optimal for this problem and I would not expect to be able to achieve a meaningful reduction in cost by using an alternative DRL approach or performing additional hyperparameter tuning.

The policy learned by Q-value iteration can be represented as an $(\mathtt{s},\mathtt{S})^{\dagger}$ policy, which in part explains the strong performance of the $(\mathtt{s},\mathtt{S})^{\dagger}$ reported in Section 3.4.7. This structure is consistent with the finding of Blake *et al.* [37] that the age profile didn't factor into the ordering decision for platelets if an order could be placed every day. The results may differ if the simplifying assumption that all of the units have the same remaining useful life on arrival is relaxed. Comparing the $(\mathtt{s},\mathtt{S})^{\dagger}$ parameters fit using stochastic programming in Table 3.2 with those that represent the Q-value iteration policy in Table 3.6, the order-up-to level parameters $\mathtt{S}$ are similar except for Tuesday. With the exception of Thursday, the reorder point parameters $\mathtt{s}$ fit using stochastic programming are consistently lower than those found using Q-iteration. The stochastic programming parameters were fit using only a small number of possible demand trajectories and therefore these values likely represent the minimum values to ensure that order are made almost every day in those sampled trajectories and may mean that, on future unseen data, orders are not placed when it would be advantageous to do so.

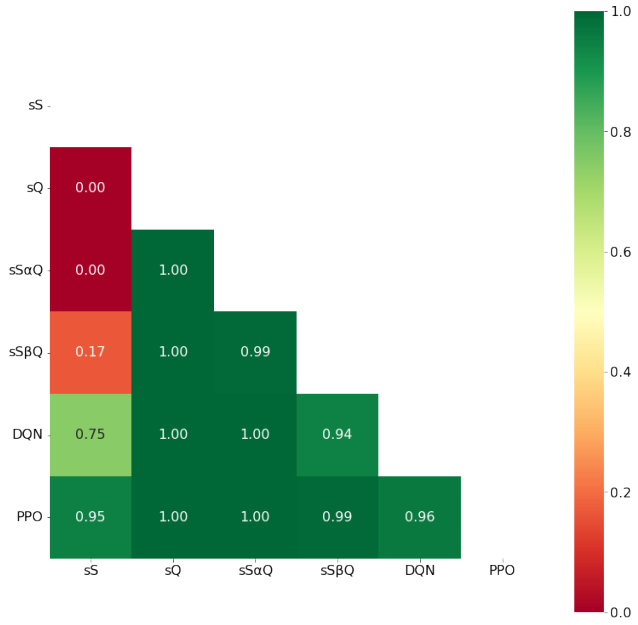
3.4.8 Policy comparison

Policy	Mean daily cost						Mean KPIs		
	Fixed	Variable	Holding	Shortage	Wastage	Total	Service level (%)	Days with shortage (%)	Wastage (%)
Perfect foresight	225	23,027	0	0	0	23,252	100.0	0.0	0.0
Q-value iteration	225	22,961	1,337	423	0	24,946	99.6	4.1	0.0
$(\mathtt{s}, \mathtt{S})^{\dagger}$	223	22,863	1,246	922	0	25,254	99.1	6.5	0.0
$(\mathtt{s}, \mathtt{Q})$	157	22,787	3,002	2,332	122	28,401	97.8	10.6	0.5
$(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$	203	22,850	1,825	1,144	0	26,023	99.0	8.5	0.0
$(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$	224	22,826	1,257	1,108	0	25,414	99.0	8.0	0.0
DQN	225	22,986	1,618	317	0	25,146	99.6	2.8	0.0
PPO	225	22,979	1,480	340	0	**25,024**	99.7	3.2	0.0

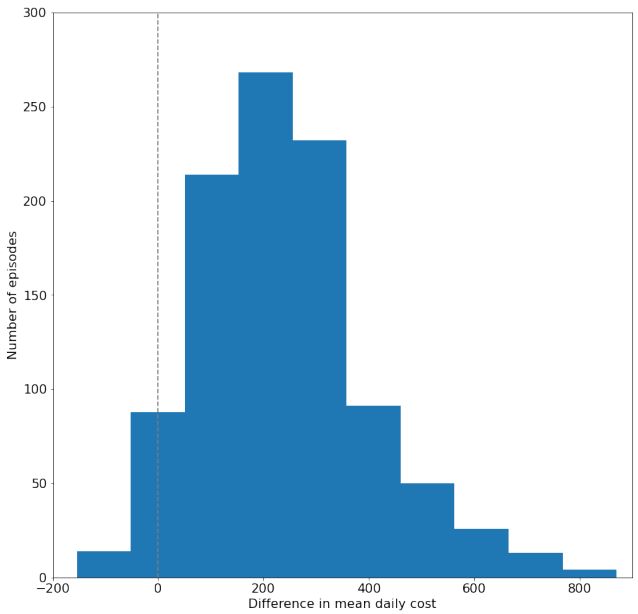
The daily cost components, daily total cost and KPIs were calculated for each episode. The mean of the daily cost components, daily total cost and KPIs over the 1,000 evaluation episodes is reported. The lowest mean daily total cost given by an approximate method without perfect foresight is indicated in **bold**.

Table 3.7: Mean daily costs incurred and associated KPIs given by the Q-value iteration, heuristic, and DRL policies.

In this section I compare the performance of the policies from Sections 3.4.5, 3.4.6 and 3.4.7 on the same 1,000 evaluation episodes. I present the mean daily costs for these policies, originally reported above in Tables 3.3, 3.4 and 3.5 in Table 3.7. I also include a benchmark policy with perfect foresight: it is assumed to order exactly the quantity required to meet the demand on the following day and therefore incurs no holding, shortage or wastage costs. I separate out both Q-value iteration and the perfect foresight policy from the other policies in Table 3.7 because they both have access to information that would not be available in reality: either the full transition probability distribution (Q-value iteration), or perfect knowledge of future demand before an order is placed (perfect foresight). Both of these policies provide useful context for the others that I present: Q-value iteration provides the optimal policy given the uncertainty and so is a lower bound on the mean cost I would expect the other models to incur. The policy with perfect foresight provides a lower bound on the cost if perfect future knowledge were available and therefore difference in performance between perfect foresight and Q-value iteration represents the possible benefit of a short-term forecast in this scenario. The implementation of a perfect foresight policy here is limited to the current cost structure, where the fixed order cost is much lower than the holding cost that would be incurred by holding a number of units to meet the mean daily demand overnight.



(a) Heatmap showing the proportion of the 1,000 evaluation episodes in which the daily cost of the policy on the vertical axis was lower than that of the policy on the horizontal axis.



(b) Distribution of the difference in daily cost between $(\mathtt{s}, \mathtt{S})^{\dagger}$ and PPO policies on the 1,000 evaluation episodes.

Figure 3.3: Relative performance of the heuristic and DRL replenishment policies.

I also include the mean of three KPIs over the 1,000 evaluation episodes in Table 3.7. The service level is the percentage of demand that was filled using platelet units in stock; the days with shortage is the percentage of days in which demand exceeded units in stock and an emergency order had to be placed; and the wastage rate is the number of wasted units as a proportion of the total units received through routine daily orders. All of the policies except $(\mathtt{s}, \mathtt{Q})$ achieve no waste and a service level over 99%. The DQN model achieved the lowest proportion of days with a shortage, followed by the PPO model.

In Figure 3.3a I present paired-sample comparisons between policies, showing the proportion of the 1,000 evaluation episodes in which the policy on the row label incurred a lower mean cost than the model on the column label. It is clear from this figure that the two DRL approaches consistently outperformed the policies fit using stochastic programming. Of the two DRL approaches, PPO achieved a lower daily cost than DQN in 96% of the evaluation episodes. Figure 3.3b shows the distribution of differences in daily cost on the evaluation episodes for the best performing stochastic programming method, $(\mathtt{s},\mathtt{S})^{\dagger}$, and PPO. The mean difference is approximately 1% of the mean daily cost. The mean daily cost for PPO is 8% higher than the perfect foresight policy.

In Section 3.4.7, I observed that the optimal policy computed using Q-value iteration can be represented by an $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy, and therefore does not depend on age information about the stock. This suggests that the advantage of the DRL policies in this particular scenario is not in their ability to incorporate knowledge about the age of the inventory into the ordering decision. Instead, it is likely to be because the DRL methods are able to efficiently learn from a much larger number of samples and therefore have a better implicit representation of the uncertainty in the environment. An alternative method for fitting the heuristic policies parameters, that can efficiently use a large number of sampled demand trajectories, may give better performance and I consider this in Chapter 4.

One important consideration is how straightforward these methods are for future researchers. There are an increasing number of open-source libraries making

DRL methods relatively easily accessible. If this trend continues, it may well be more straightforward for researchers to set up their problem as a simulation that can be wrapped in an RL environment and use these RL libraries, as I have done, than to describe the problem using linear constraints and use stochastic mixed integer programming. This approach could also be applied to a much wider range of problems that cannot be described using linear constraints.

3.5 Application to demand data from UCLH

In Section 3.4 I show that DRL methods can learn effective policies for ordering platelets in a simplified setting where demand is generated using day-of-the-week specific Poisson distributions. In this section, I investigated whether these approaches can be straightforwardly applied to real demand trajectories from a hospital blood bank and evaluate their performance.

3.5.1 Methods

I extracted daily demand figures for adult platelet units for the period 1 January 2015 to 31 December 2017, inclusive, from an extract of data from the transfusion laboratory information system Bank Manager. Units were included in the demand figures if the unit fate was "Transfused" or "Transfusion Assumed", and were assigned to calendar days based on the day on which the request was required to be filled because this reflects the point at which they would need to be issued by the hospital blood bank. For additional detail on the data extracted from Bank Manager, including descriptive analysis, see Appendix B.

With the exception of the demand trajectories, I used the same scenario and assumptions described in Section 3.4.1. The parameters for each policy were fit using demand data from 2015 and 2016.

Q-value iteration requires knowledge of the full transition probability distribution for the MDP, which is not available for the real-life demand data. Both stochastic programming and the DRL methods can be used to find policies based on sampled trajectories, and the real-life observed data can be considered a sample from the underlying distribution. Therefore in this section I only consider

the stochastic programming and DRL approaches from Section 3.4

The same hardware was used as for the corresponding methods with simulated demand trajectories, as set out in Section 3.4.3.

3.5.1.1 Stochastic programming

I followed the approach described in Section 3.4.5 above, fitting parameters for the four policies described by Rajendran and Srinivas [45], and set out in Equations 3.6 to 3.9 above, using stochastic programming implemented using the Python libraries Pyomo [174, 175] and mpi-sppy [176], and the commercial solver Gurobi [177].

Instead of using 20 randomly sampled scenarios, the demand trajectories were 20 consecutive 30-day periods all starting on a Monday, beginning with the first Monday in 2015. The starting inventory for each scenario was 25 units with a remaining useful life of two days, based on the mean daily demand observed in 2015 and 2016. All other settings were the same as those used for the experiments in Section 3.4.5.

3.5.1.2 Deep reinforcement learning

I followed the approach described in Section 3.4.6 above, training DQN and PPO models using the Python library Tianshou [179].

Instead of using a `DemandProvider` that randomly generated 365-day long trajectories for model fitting, I used a `DemandProvider` that sampled 90-day long trajectories from the real demand data for 2015 and 2016. 90 days was selected as middle ground between having long episodes to reduce the impact of initial conditions and having enough variety in the number of different trajectories that could be sampled from a two year period. All other settings were the same as those used for the experiments in Section 3.4.6.

3.5.1.3 Evaluation

Once fit, an agent to represent each class was created and applied to an RL environment as described in Section 3.4.2. For evaluation, the RL environment was instantiated with a `DemandProvider` class that provided the real demand figures from 2017 as the demand trajectory. The daily rewards (and reward components)

from this single episode were recorded for each policy.

3.5.2 Results

I present the parameters found for each policy using the stochastic programming approach in Table 3.8. I present the mean daily costs incurred and KPIs achieved by each policy parameterized using those values, and two DRL policies, on real UCLH demand data from 2017 in Table 3.9.

Policy	Parameter	Weekday						
		Mon	Tue	Wed	Thu	Fri	Sat	Sun
$(s, S)^{\dagger}$	s	34	24	21	37	22	17	29
	S	61	35	38	40	28	46	60
(s, Q)	s	20	19	31	27	21	16	15
	Q	53	36	27	32	24	14	27
(s, S, α, Q)	s	38	20	20	39	19	15	36
	S	39	33	40	40	26	27	37
	α	11	13	2	18	7	11	12
	Q	24	12	26	11	12	9	21
(s, S, β, Q)	s	38	29	23	39	19	17	21
	S	39	36	33	40	25	26	36
	β	1	8	1	1	1	2	7
	Q	50	23	41	35	34	27	38

Table 3.8: Heuristic replenishment policy parameters from my reimplementation of the stochastic programming methods from Rajendran and Srinivas [45] applied to real UCLH demand data from 2015 and 2016

Policy	Daily cost						KPIs		
	Fixed	Variable	Holding	Shortage	Wastage	Total	Service level (%)	Days with shortage (%)	Wastage (%)
Perfect foresight	225	15,933	0	0	0	16,158	100.0	0.0	0.0
$(s, S)^{\dagger}$	182	15,985	2,171	151	32	18,522	99.8	2.5	0.2
(s, Q)	171	15,374	1,531	3,214	84	20,374	96.0	20.0	0.5
(s, S, α, Q)	218	15,698	1,117	1,255	0	18,289	98.4	8.2	0.0
(s, S, β, Q)	219	15,840	1,200	543	2	17,804	99.3	6.0	0.0
DQN	225	15,942	1,685	98	0	17,950	99.9	1.1	0.0
PPO	225	15,940	1,429	116	0	**17,710**	99.9	2.7	0.0

The daily cost components, daily total cost and KPIs were calculated for a single episode using real observed daily demand for one calendar year, 2017. The lowest daily total cost given by an approximate method without perfect foresight is indicated in **bold**.

Table 3.9: Daily costs incurred and associated KPIs given by the heuristic and deep reinforcement learning policies on real UCLH demand data from 2017

3.5.3 Discussion

These results show that the DRL policies were also able to learn effective replenishment policies on real demand trajectories, and that the policies generalized well to a new period. PPO achieved the lowest mean daily cost, with the joint-highest service level and joint-lowest wastage rate.

Unlike in the evaluation based on simulated data, in this experiment $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ perform better than the $(\mathtt{s}, \mathtt{S})^\dagger$ policy in terms of daily cost, although they both experience shortages on a higher number of days. $(\mathtt{s}, \mathtt{Q})$ remains the worst-performing policy, incurring the highest daily cost, the lowest service level and the highest wastage rate. The policy used by UCLH in current practice is a standing order policy, similar to $(\mathtt{s}, \mathtt{Q})$ but with the reorder point $\mathtt{s}$ effectively set to infinity.

It may be possible to improve the performance of the DRL models by performing hyperparameter tuning for this task, rather than using the hyperparameters that were found to work well in Section 3.4.6. However, the purpose of these experiments was not to find the best possible performing policy but to investigate whether the DRL methods were competitive on real demand data, as they were on simulated data.

The main limitation of this work is that the results cannot be reasonably compared with real performance over the period because, other than the demand data, the other settings of the scenario have been taken from the work in Section 3.4 and therefore are based on simplifying assumptions, e.g. that all arriving inventory is fresh and that demand for a unit can be met with any unit in stock, which make the task considerably easier.

3.6 Conclusion

In this chapter I have established the effectiveness of DRL for learning platelet replenishment policies. Using simulated demand data, the replenishment policy fit by PPO achieved lower costs than the four heuristic policies with parameters fit using stochastic mixed integer linear programming. PPO also achieved the lowest

daily cost when trained and temporally validated on real demand data from a UK hospital.

Using a CPU-based implementation of Q-value iteration, I was able to find the optimal policy for the case with simulated demand data. This policy was used as an additional benchmark, and PPO performed near-optimally. Additionally, I identified that the optimal policy could be represented as an $(s,S)^{\dagger}$ policy, but that the stochastic mixed integer linear programming has identified sub-optimal parameters. An alternative method to fitting such heuristic policy parameters, that can be fit using a larger number of demand trajectories, may improve the performance of the heuristic policies.

I implemented the scenario as an RL environment using a standard Python library. This environment can be used by other researchers to test new methods on the same problem, and expanded to incorporate additional aspects of the real problem.

Chapter 4

GPU-accelerated value iteration and simulation for perishable inventory management

4.1 Motivation

In Chapter 3, I computed the optimal policy for a simple platelet replenishment scenario introduced by Rajendran and Srinivas [45] using a dynamic programming approach (specifically: Q-value iteration) and used this policy as a benchmark for the performance of heuristic and DRL policies. In this chapter, I investigate whether recent advances in GPU hardware and software increase the size of perishable inventory replenishment problems for which it is practical and feasible to compute the optimal policy.

As I explain in Section 2.4.1.1, the need to consider the age profile of stock limits the size of perishable inventory problems for which dynamic programming approaches are feasible, due to the "curse of dimensionality". However, the common understanding in the operational research literature about the size of problems for which it can be used does not appear to have changed meaningfully since the 1980s despite significant advances in computational power - particularly GPUs. The parallel processing capabilities of GPUs are well-suited to dynamic programming. In addition to situations where the optimal policy can be found

and used in practice, extending the range of problems for which it is feasible and practical to find the optimal policy could support research into new heuristics and approximate approaches, including DRL, by providing performance benchmarks on larger, more realistic, problems than has previously been possible.

Additionally, I consider in this chapter the application of GPU-accelerated computing to simulation optimization to fit parameters for heuristic benchmark policies. In Chapter 3, I found that while the optimal replenishment policy for the case with simulated demand data could be represented as an `(s,S)`† policy, stochastic mixed integer linear programming did not identify the corresponding parameters within the permitted run time of 36 hours. The parallel processing capabilities of GPUs support an extensive search of possible parameters for heuristic policies, and a reduction in the sampling error for the estimate of the objective function by running a large number of simulated rollouts for each combination of parameters proposed by the search strategy.

One of the reasons for the limited adoption of GPU-accelerated computing among operational researchers compared to machine learning researchers may be the perceived barrier to entry, and therefore I focused on developing an accessible approach using high-level software libraries, which require minimal GPU-specific knowledge, and consumer-grade hardware.

4.2 Contribution statement

The main contributions of this chapter are:

- establishing that value iteration can be used to compute optimal perishable inventory replenishment policies, for problems where value iteration was recently described as computationally infeasible or impractical, using consumer-grade GPU hardware;
- providing an open-source, general purpose implementation of value iteration for MDPs using the Python library JAX, and four examples of how it can be customized to solve specific perishable inventory problems without in-depth knowledge of GPU-specific programming approaches and frameworks; and

- demonstrating that simulation optimization for perishable inventory management can also be effectively run in parallel using JAX, particularly for larger problems where policies that perform well can be identified in a fraction of the time required to run value iteration.

The extension of value iteration to larger problem instances is summarized in Table 4.1.

		Maximum useful life m				
	Problem features	2	3	4	5	8
A	Lead time > 1	●	✱	✱	✱	○
B	Substitution between products	●	✱	○	○	○
C	Not all arrivals fresh, freshness $\sim$ order quantity, periodic demand	○	●	○	✱	✖

Key
- ● Value iteration feasible for all experiments in the original study.
- ✱ My method extends value iteration to experiments that were considered infeasible or impractical in the original study.
- ✖ All experiments infeasible in the original study and with my method.
- ○ Setting not considered in the original study.

Table 4.1: Summary of my contribution extending value iteration to larger problems with a longer maximum useful life.

At the time of writing, in June 2024, an article based on this chapter is under review at *Annals of Operations Research*. A preprint is available on *arXiv* [184].

4.3 Background and related work

Early theoretical work demonstrated that optimal policies for perishable inventory replenishment could be computed using dynamic programming [38, 39]. In 1982 Nahmias [6] observed that the need to consider the age profile of the stock made dynamic programming approaches impractical for problems where the maximum useful life of the product was more than two periods, due to the "curse of dimensionality". More recently, despite advances in computational power,

researchers have stated that value iteration remains infeasible or impractical when the maximum useful life of the product is longer than two or three periods or when additional complexities (e.g. substitution between products or stochastic remaining useful life on arrival) are introduced [40, 71, 166].

The prevalent view in the perishable inventory literature about the scale of problems for which value iteration is feasible appears to neglect the recent developments in GPU hardware, and in software libraries that make it possible for researchers and practitioners to take advantage of GPU capabilities without detailed knowledge of GPU hardware and GPU-specific programming approaches. The first mainstream software framework to support general computing tasks on GPUs using a common general purpose programming language was Nvidia's Compute Unified Device Architecture (CUDA) platform [185], which launched in 2007.

A wide range of sequential decision making problems can be modelled as MDPs and solved using value iteration. Jóhannsson [186] demonstrated that value iteration could be effectively run in parallel on a GPU soon after the introduction of CUDA. Subsequent research has evaluated the performance of GPU-accelerated value iteration on problems from economics and finance [187, 188, 189, 190, 191] and route-finding and navigation [192, 193, 194, 195]. I have only identified a single study that applied this approach to an inventory management problem: Ortega *et al.* [196] implemented a custom value iteration algorithm in CUDA to find replenishment policies for a subset of perishable inventory problems originally described by Hendrix *et al.* [71].

Previous studies have reported impressive reductions in wall time achieved by running value iteration on GPU. The GPU-accelerated method developed by Ortega *et al.* [196] was up to 11.7× faster than a sequential CPU-based method written in C. Despite this, the approach has not been widely adopted. The majority of the aforementioned studies focus on implementing value iteration using CUDA or OpenCL, a multi-platform alternative to CUDA, and comparing the performance of a GPU-accelerated method with a CPU-based method, instead seeking to use GPU-acceleration to solve problems for which value iteration is

otherwise impractical or infeasible. A similar approach has been used to develop GPU-accelerated methods for other operational research problems [197, 198, 199]. One of the main barriers to entry for other researchers may be the perceived difficulty of GPU programming. Writing efficient code using CUDA or OpenCL requires careful consideration of memory access, balancing resource usage when mapping parallel processes to the hardware, and interaction between the CPU and GPU [200]. There have been efforts to make GPU-accelerated value iteration more accessible. Jóhannsson [186] created a solver framework using his CUDA implementation of value iteration as a back-end, but this does not appear to be publicly available. More recently, Kirkby [190] created a toolkit in MATLAB to solve infinite horizon value iteration problems which automatically uses a GPU when available and appropriate. This toolkit requires the state-transition matrix to be provided as an input, which is not practical for some of the larger problems I consider due to memory limitations.

One of the areas in which GPUs have since had a major impact is the field of deep learning. This impact has led to, and in turn been supported by [201], the development of higher-level software libraries including TensorFlow [202], PyTorch [203] and JAX [141]. These libraries provide comparatively simple Python APIs to support easy experimentation and developer productivity, while utilising highly optimized CUDA code "under-the-hood" to exploit the parallel processing capabilities of GPUs.

My approach is broadly similar to that of Duarte *et al.* [189] and Sargent and Stachurski [204] who used machine learning frameworks to implement GPU-accelerated value iteration for economics models. A key observation made by Duarte *et al.* [189] is that their TensorFlow implementation is an order of magnitude faster than their custom CUDA C++ implementation on a GPU. This demonstrates that the comparative accessibility provided by a machine learning framework need not come at the cost of poorer performance. TensorFlow, like PyTorch and JAX, translates the high level API instructions into highly-optimized CUDA code. Experts in GPU programming may be able achieve better performance

than a machine learning framework by working at the level of CUDA or OpenCL. Researchers without that expertise are likely to both save development time and achieve performance benefits by working at a higher level of abstraction using a machine learning framework and relying on it to make best use of the available hardware.

Due to the computational challenges of using dynamic programming methods to find policies for perishable inventory management, research has focused on approximate solutions. One straightforward way to make a dynamic programming approach more computationally tractable is to reduce the size of the state space by aggregating stock items into batches. The solution to the down-sized dynamic program can then be factored up to give an approximate solution to the original problem [44, 49]. An alternative approach is to use a heuristic policy with a small number of parameters, such as a base stock policy. Research in this area has concentrated on both identifying suitable structures for heuristic policies (see Nahmias [43] for an early example, and Haijema and Minner [42] for a recent example), and finding suitable parameters for those policies in specific situations - commonly using stochastic mixed integer linear programming [36, 53, 54] or simulation optimization [46, 48]. Recently, RL methods have also been used to find approximate polices for managing perishable inventory [40, 150, 152, 162].

Of these other approaches to the problem, I focus on simulation optimization in addition to value iteration because GPU-accelerated simulation is feasible using available software libraries but not yet widely adopted. Applied research on specific mixed integer linear programs often relies on commercial solver software such as IBM ILOG CPLEX Optimization Studio or Gurobi Optimizer which do not currently support GPU-acceleration. Adapting mixed integer programming solution strategies to suit the architecture of GPUs is the subject of active research [205]. The use of GPU-acceleration supported by machine learning frameworks is already widespread in RL research - my RL experiments in Chapter 3 used PyTorch.

Simulation optimization can be used to solve optimization problems where the objective function cannot be computed exactly, but can be estimated using

simulation. Sampling error in the objective function can be reduced by running simulations for a longer simulated period, or by running additional simulations. The relevance of parallel computing to simulation optimization is well recognized [206, 207], but I have identified few simulation optimization studies using GPUs to run multiple simulations in parallel. In inventory management, Srimool *et al.* [208] exhaustively evaluated the possible order quantities for a newsvendor problem using parallel simulations on GPU. More recently, Lau and Srinivasan [209] used simulation optimization to solve a chemical process monitoring problem using GPU-acceleration for both the simulation and the metaheuristic search process that proposed candidate solutions. Similar to the value iteration studies discussed above, both of these projects used custom CUDA code which may explain the limited subsequent adoption despite the established reductions in wall time relative to CPU baselines. Following recent work in the RL community [120, 121] I implemented my simulators using the Python library gymnax [119], which enabled me to write my simulation operations using JAX and readily evaluate each of numerous policies on thousands of simulated years in parallel on GPU.

4.4 Methods

4.4.1 Scenarios

I considered three perishable inventory management scenarios recently described in the literature for which value iteration was reported as infeasible or impractical for at least some experimental settings, and which include elements relevant to the management of blood product inventory. I also revisited the platelet replenishment scenario from Chapter 3 [45] to compare the heuristic policy parameters fit using the GPU-accelerated simulation optimization approach I developed in this chapter to those fit using stochastic mixed integer programming in Section 3.4.5.

The three new scenarios are all periodic review, single-echelon perishable inventory problems with a fixed, known delivery lead time L. Scenario A, from De Moor *et al.* [40], is a straightforward perishable inventory replenishment problem but for some experimental settings the lead time, L, is greater than one

period and therefore it is necessary to consider inventory in-transit when placing an order. This may be relevant at higher echelons of the blood supply chain and UCLH transfusion laboratory staff suggested a longer lead time may be necessary if there was more variation in the order quantities placed by a hospital blood bank due to the use of daily forecasts and more complicated replenishment policies. Scenario B is the two product scenario described by Hendrix *et al.* [71] which adds the complexity of substitution between perishable products. Substitution is an important aspect of managing blood product inventory, where compatibility between the blood groups of the donor and the recipient is critical (as set out in Section 2.2). Scenario C, from Mirjalili [166], models the management of platelets in a hospital blood bank and adds two complicating factors: periodic patterns of demand, and uncertainty in the remaining useful life of products on arrival, which may depend on the order quantity. In every scenario demand is stochastic, unmet demand is assumed to be lost, and units in stock with a remaining useful life of one period are assumed to expire at the end of the day. Except in Scenario C, the products have a fixed, known useful life m and are all assumed to arrive fresh. I summarize the key differences between the scenarios in Table 4.2.

		Problem features						**Reward function components**					
	Source	Products	Lead time > 1	Substitution	Not all arrivals fresh	Freshness $\sim$ order quantity	Periodic demand	Variable ordering	Fixed ordering	Wastage	Shortage	Holding	Revenue
A	De Moor *et al.* [40]	1	✓					✓		✓	✓	✓	
B	Hendrix *et al.* [71]	2		✓				✓					✓
C	Mirjalili [166]	1			✓	✓	✓		✓	✓	✓	✓	

Table 4.2: Summary of the key differences between scenarios A, B and C.

De Moor *et al.* [40], Hendrix *et al.* [71] and Mirjalili [166] each used different notation to describe their work. In an effort to aid the reader in understanding the similarities and differences between the scenarios I have adopted a single notation which I apply to all three scenarios. In Appendix F I present the key equations describing each scenario in this notation.

All of the scenarios are defined as MDPs (see Section 2.5.1) and in this chapter I only consider deterministic replenishment policies, $a = \pi(s)$.

4.4.2 JAX

For the experiments in this chapter, I used the open source Python library JAX [141] to implement both value iteration and the simulators for both simulation optimization and policy evaluation. JAX has a NumPy-style API that will be familiar to many researchers, and performs just-in-time (JIT) compilation to run workloads efficiently on hardware accelerators including GPUs. JAX provides composable transformations of Python functions which make it straightforward to apply functions in parallel over arrays of input data. This is ideal for dynamic programming methods like value iteration: each iteration requires many independent updates which can be performed in parallel, and the up-front computational cost of JIT compilation can be amortized over many repeats of the compiled operation for each iteration.

4.4.3 Value iteration

I describe value iteration, a dynamic programming approach for computing the optimal policy of a finite MDP, in 2.5.2.1. The optimal value function is estimated by using the Bellman optimality equation as an update operation:

$$V_{j+1}(s) = \max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}, r \in \Psi} p(s', r|s, a) \left[r + \gamma V_j(s') \right] \tag{4.1}$$

For a finite MDP this operation will, in the limit of infinite iterations, converge to the optimal value function. The optimal policy can be extracted from the value function using a one-step ahead search:

$$\pi(s) = \arg\max_{a \in \mathbb{A}} \sum_{s' \in \mathbb{S}, r \in \Psi} p(s', r|s, a) \left[r + \gamma V(s') \right] \tag{4.2}$$

Similar to the approach of Hendrix *et al.* [71], I used a deterministic transition function, $(s', r) = T(s, a, \omega)$, where $\omega \in \Omega$ is a possible realization of the stochastic element(s) of the transition. For a specific state-action pair, (s, a), and

a specific random outcome, ω, the next state and the reward can be calculated deterministically. In the most straightforward example I consider, Scenario A, the only uncertainty is in the daily demand, and therefore Ω is the set of possible values that demand may take in any period. Under this formulation, the value iteration update equation can be rewritten as:

$$V_{j+1}(s) = \max_{a \in \mathbb{A}} \sum_{\omega \in \Omega} P(\omega|s,a) \left[r_\omega + \gamma V_j(s'_\omega) \right], \text{where } (r_\omega, s'_\omega) = T(s,a,\omega) \quad (4.3)$$

and the optimal policy can be extracted using the equation:

$$\pi(s) = \arg\max_{a \in \mathbb{A}} \sum_{\omega \in \Omega} P(\omega|s,a) \left[r_\omega + \gamma V(s'_\omega) \right] \quad (4.4)$$

where $P(\omega|s,a) = \text{Prob}\left(\Omega_t = \omega | S_t = s, A_t = a\right)$ is the probability of random outcome ω having observed state $S_t = s$ and then taken action $A_t = a$. Ω_t represents the stochastic elements of the transition that occur between the observation of state S_t and the observation of state S_{t+1}.

Since it is not possible to run an infinite number of iterations, a convergence test is needed to determine when to stop value iteration and extract the policy. The main output of interest is the policy, and not the value function itself, and therefore in certain cases the number of iterations required can be reduced by stopping when further updates to the value function will not change the policy. I describe the convergence test used for each scenario in the corresponding section below.

In Chapter 3, I used a CPU-based implementation of Q-value iteration to compute the optimal policy for the platelet replenishment scenario described by Rajendran and Srinivas [45]. Q-value iteration follows the same basic updating process as value iteration, but the estimates of all the state-action values (rather than just the value for best action) are retained at each step. This makes it easier to extract the final policy, but has greater memory requirements. Memory is a limiting factor in the GPU-accelerated approaches I consider in this chapter, and therefore I used value iteration here.

I implemented value iteration using a custom Python class, `ValueIterationRunner`, which defines the common functionality required to run value iteration and extract the optimal policy using the approach in Equations 4.3 and 4.4. The base `ValueIterationRunner` class includes eight placeholders for methods which must be defined for a specific scenario. For each scenario I defined a subclass of `ValueIterationRunner`, replacing the placeholder methods with custom functions that:

- return a list of all possible states as tuples;
- return an array that maps from a state to its index in the list of all possible states;
- return an array of all possible actions;
- return an array of all possible random outcomes;
- return the immediate reward and next state following the deterministic transition function given a state, action and random outcome;
- return an array with the probability of each random outcome given a state-action pair;
- return an initial estimate of the value function; and
- test for convergence of the value iteration procedure.

The `ValueIterationRunner` class could be easily adapted to solve new problems by creating a new subclass and replacing the placeholder entries for these eight methods.

A naive implementation of value iteration following Equation 4.3 would require a nested for-loop over every state, every action, and every random outcome. Every iteration, j, requires an outer loop over every state but the updates for each state can be computed independently using the estimates of the value function from the previous iteration $j - 1$. This independence means that, during each iteration, the value function updates for each state can be computed in parallel instead of in

sequence. At the end of each iteration, the results of the parallel updates can be combined into a complete updated estimate of the value function, which is then distributed as an input for the next iteration.

The independence of the updates, each of which involves performing the same operations on different inputs, is ideal for parallelization on GPUs. GPUs design was historically driven by the needs of gaming, where rendering images at high frames rates requires performing a large number of floating-point operations per second with high levels of memory access. This led to a throughput-oriented design, where GPUs excel at handling a large number of threads simultaneously, each working on a different part of the input data. When a thread experiences latency, such as waiting for memory access, the GPU can switch to another thread, keeping the hardware fully utilized. GPUs are especially efficient when groups of threads, called warps, execute the same instruction at the same time. This minimizes control overhead, as the same instruction can be issued to all threads in a warp simultaneously. It also speeds up memory access by allowing threads to share their memory requests, efficiently fetching shared regions of an array that multiple threads need, reducing the total number of times the input data in memory needs to be queried [210].

JAX provides two main function transformations that facilitate running functions in parallel: vectorizing map (vmap) and parallel map (pmap). Both of these transformations create new functions that map the original function over specified axes of the input, enabling the original function to be applied to a large number of inputs in parallel. The key difference is that vmap provides vectorization, so the operations happen on the same device (e.g. the same GPU), while pmap supports single-program, multiple-data parallelism and runs the same operation for different inputs on separate (but identical) devices at the same time.

An important feature of vmap and pmap is that they are composable and therefore can be readily nested. For the basic value iteration update, set out in Algorithm 4, nested vmap operations over states, actions, and random outcomes are used instead of using nested loops. This is only feasible if there is sufficient GPU

memory to update all of the states simultaneously. For larger instances I grouped the states into batches for which the update can be performed simultaneously and performed the update for one batch of states at a time. To enable multiple identical devices to be used where available, my implementation automatically detected the number of available of devices and used pmap to map the update function for multiple batches of states over the available devices.

Functions transformed by vmap and pmap are automatically JIT-compiled with XLA, a domain-specific compiler for linear algebra. JAX traces the function the first time it is run, and the traced function is compiled using XLA into optimized code for the available devices. This compilation requires fixed-sized arrays, as XLA generates optimized machine code for specific array dimensions, to avoid incurring the cost of recompilation for varying input shapes. In the case of pmap, this means that all arrays distributed across devices must have the same shape [211].

To meet the requirement for fixed sized arrays I padded the array of states so that it could be reshaped to an array with dimensions (*number of devices*, *number of batches*, *maximum batch size*, *number of elements in state*). Each device received an array with dimensions (*number of batches*, *maximum batch size*, *number of elements in state*), performed a loop over the leading dimension, and calculated the update one batch of states at a time. The same process was used to extract the policy in parallel at the end of value iteration.

I used double-precision (64-bit) numbers when running value iteration, instead of the single-precision (32-bit) numbers that JAX uses by default because, in preliminary experiments with Scenario A which required a large number of iterations for convergence, I found that convergence was not always stable.

I report the wall time required to run each value iteration experiment. The reported times include JIT compilation time, writing checkpoints and writing final outputs, including the policy, because I believe this represents a realistic use case.

4.4.4 Simulation of the Markov decision processes

I created a simulator to represent each scenario. The simulators have two purposes: firstly, to fit parameters for heuristic replenishment policies using simulation

Algorithm 4 Value iteration using vmap

Initialize array of all states s: $\mathbb{S}$
Initialize array of all actions a: $\mathbb{A}$
Initialize array of all random outcomes ω: $\mathbb{\Omega}$
Initialize initial estimate of value function: $V_0(s) \quad \forall s \in \mathbb{S}$
Initialize discount factor: γ
Initialize iteration counter: $j = 0$
Define deterministic transition function which returns next state and reward: $T(s, a, \omega)$

Perform value iteration
while not converged **do**
  $j \leftarrow j + 1$
  vmap over $s \in \mathbb{S}$
    vmap over $a \in \mathbb{A}$
      vmap over $\omega \in \mathbb{\Omega}$
        $(s'_\omega, r_\omega) \leftarrow T(s, a, \omega)$
      $Q_j(s, a) \leftarrow \sum_\omega P(\omega|s, a) \left[r_\omega + \gamma V_{j-1}(s'_\omega)\right]$
    $V_j(s) \leftarrow \max_a Q_j(s, a)$
  Test for convergence
end while

Extract the policy, $\pi(s) \approx \pi^*(s)$
vmap over $s \in \mathbb{S}$
  vmap over $a \in \mathbb{A}$
    vmap over $\omega \in \mathbb{\Omega}$
      $(s'_\omega, r_\omega) \leftarrow T(s, a, \omega)$
    $Q_{j+1}(s, a) \leftarrow \sum_\omega P(\omega|s, a) \left[r_\omega + \gamma V_j(s'_\omega)\right]$
  $\pi(s) \leftarrow \arg\max_a Q_{j+1}(s, a)$

optimization and, secondly, to evaluate the performance of the policies produced by value iteration and simulation optimization based on the return and three KPIs: service level, wastage and holding. The service level is the percentage of demand that was met over a simulated rollout, wastage is the proportion of units received that expired over a simulated rollout and holding is the mean number of units in stock at the end of each day during a simulated rollout.

Each simulator is an RL environment written using the Python library gymnax [119], which is based on JAX. This provides a standard interface for working with MDPs, while enabling many simulations to be run in parallel on a GPU using vmap.

For a single policy, vmap can be used to run a simulated rollout over multiple random seeds. Multiple sets of parameters can be simultaneously evaluated for the same heuristic policy on a shared set of random seeds by nesting vmapped functions. I note that it would be straightforward to use RL software libraries to learn policies for these scenarios using these environments.

I selected different heuristic policies for the different scenarios from the literature, considering which (if any) heuristic was used in the original study and the structure of each problem. I describe the heuristic policy used for each scenario in the corresponding section below. All of the heuristic policies use an order-up-to level parameter `S`, and some have an additional reorder point parameter `s`.

I used the Python library Optuna [212] to suggest parameters for the heuristic policies. For heuristic policies with a single parameter, I used Optuna's grid sampler to evaluate all feasible values simultaneously in parallel. For policies with multiple parameters, I used a genetic algorithm, Optuna's Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [213] sampler, to search the parameter space.

For each suggested set of parameters, I ran 4,000 episodes, each 365 days long after a warm-up period of 100 days. When using the grid sampler, I selected the parameter value with the highest mean return from the single parallel run. When using the NSGA-II sampler I evaluated 50 sets of parameters in parallel, corresponding to a single generation of the genetic algorithm, and ranked them by mean return. I terminated the NSGA-II search procedure when the best combination of parameters had not changed for five generations, or when a total of 100 generations had been completed.

The NSGA-II search algorithm is designed for multi-objective optimization and therefore candidate solutions are selected to generate offspring and/or be carried forward to the next generation based on whether they dominate other candidate solutions and occupy less crowded areas of the solution space to maintain diversity. Despite being designed for multi-objective problems, in preliminary experiments I found NSGA-II to be effective compared to alternatives provided by Optuna because it was able to quickly suggest the next batch of candidate solutions for

parallel evaluation.

For each scenario I compared the performance of the value iteration policy and best heuristic policy identified using simulation optimization on 10,000 simulated episodes, each 365 days long following a warm-up period of 100 days. I report the mean and standard deviation of the return and, in Appendix G, the service level, wastage and stock holding over these episodes. For each rollout, the return is the discounted sum of rewards from the end of the warm-up period until the end of the simulation. The components of the reward function for Scenarios A, B and C are summarized in Table 4.2 and the reward functions are set out in Appendix F.1, F.2 and F.3 respectively. The standard deviation of the return and the KPIs shows the effect of the stochasticity (due to random demand, random willingness to accept substitution and/or random useful life on arrival) in each scenario.

4.4.5 Reproducibility

There are two key reproducibility considerations for this chapter: firstly, accurately implementing the scenarios described in previous studies and, secondly, ensuring that others are able to reproduce my own experiments.

I compared outputs from my value iteration and simulation optimization methods to outputs from the original studies, and these checks are included as automated tests in my publicly available GitHub repository. De Moor *et al.* [40] made their code available on GitHub and fully specified the optimal and heuristic policies for two experiments in their paper which I used to test my implementation of Scenario A. For Scenario B, I compared the best parameters for heuristic policies and mean daily reward values to those reported in Hendrix *et al.* [71], and performed additional comparisons to the output of a MATLAB implementation of their value iteration method that the authors kindly made available to me. Mirjalili [166] plotted value iteration policies for a subset of his experiments, and he kindly provided me with the underlying data for those plots so that I could confirm the policies from my implementation of Scenario C matched those he had reported.

My code is available on GitHub, and is based on open-source software libraries. My GitHub repository includes a Google Colab notebook that can be

used to reproduce my experiments using a free, cloud-based GPU, avoiding local hardware requirements or configuration challenges. Note that the type of GPU allocated to a session in Colab is not guaranteed and there are service limits that restrict the maximum continuous running time. Experiments may be restarted from a checkpoint if a session terminates before the experiment is completed.

4.4.6 Hardware

All experiments were conducted on a desktop computer running Ubuntu 20.04 LTS via Windows Subsystem for Linux on Windows 11 with an AMD Ryzen 9 5900X processor, 64GB RAM, and an Nvidia GeForce RTX 3060 GPU. The Nvidia GeForce RTX 3060 is a consumer-grade GPU that, at the time of writing in June 2024, can be purchased for less than £300 in the UK [214].

To demonstrate the potential benefits of more powerful data-centre grade GPU devices, and how my approach can be easily scaled to utilize multiple GPUs, I additionally ran value iteration for one large problem case of Scenario B using one, two or four Nvidia A100 40GB GPUs on the UCL high-performance computing cluster Myriad.

4.4.7 Code availability

The code supporting the work in this chapter is available at: `https://github.com/joefarrington/viso_jax`

4.5 Scenario A: lead time may be greater than one period

4.5.1 Problem description

De Moor *et al.* [40] described a single-product, single-echelon, periodic review perishable inventory replenishment problem and investigated whether using heuristic replenishment policies to shape the reward function can improve the performance of DRL methods.

At the start of each day t the agent observes the state S_t, the current inventory in stock (split by remaining useful life) and in-transit (split by period ordered), and

places a replenishment order $A_t \in \{0, 1, ..., A_{\max}\}$. Demand is filled from available stock following either a FIFO or Last-in First-out (LIFO) issuing policy. At the end of the day, the state is updated to reflect the ageing of stock and the reward, R_{t+1}, is calculated. The reward function comprises four components: a holding cost per unit in stock at the end of the period (C_h), a variable ordering cost per unit (C_v), a shortage cost per unit of unmet demand (C_s) and a wastage cost per unit that perishes at the end of the period (C_w). The order placed on day $t - (L - 1)$ is received immediately prior to the start of day $t + 1$, and is included in the stock element of the state S_{t+1}.

The stochastic element in the transition is the daily demand D, $\mathbb{\Omega} = \{0, 1, ..., \infty\}$, in the problem described by De Moor *et al.* [40]. The state transition and the reward are deterministic given a state-action pair and the realization of the daily demand. Daily demand is modelled by a gamma distribution with mean μ and coefficient of variation $\frac{\mu}{\sigma}$, and rounded to the nearest integer. I truncated the demand distribution at $D_{\max} \gg \mu + 5\sigma$, such that $\mathbb{\Omega} = \{0, 1, ..., D_{\max}\}$, for the purposes of implementation.

The initial value function $V_0(s)$ was initialized at zero for every state. De Moor *et al.* [40] did not specify a particular convergence test for their value iteration experiments. The problem is not periodic and includes a discount factor, and I therefore used a standard convergence test for the value function [97] as set out in Appendix F.1.

De Moor *et al.* [40] considered products with a maximum useful life m of two, three, four or five periods, and evaluated eight different experimental settings for each value of m. For a product with $m = 2$, they found the optimal policy using value iteration, and used this as a benchmark for their DRL policies. For larger values of m, they instead used a heuristic policy as the benchmark on grounds of computational feasibility. The experiments for each value of m evaluate different combinations of lead time L, wastage cost C_w, and issuing policy. I demonstrate that, using JAX and a consumer-grade GPU, it is feasible to obtain the optimal policy for all of the experimental settings, up to and including a maximum useful

life m of five periods, and report the wall time required to run value iteration for each experiment.

I compared the policy from value iteration with a standard base stock policy, parameterized by order-up-to level $\mathtt{S}$, such that the order quantity on day t, given total current stock (on hand and in-transit) I_t is:

$$A_t = [\mathtt{S} - I_t]^+ \tag{4.5}$$

I evaluated the mean return for each value of $\mathtt{S} \in \{0, ..., A_{\max}\}$ using the Optuna grid sampler. I compared the base stock policy that achieves the highest mean return, characterized by parameter $\mathtt{S}_{\text{best}}$, to the value iteration policy.

See Appendix F.1 for additional information about Scenario A.

4.5.2 Results

In Table 4.3 I present the wall time in seconds required to run value iteration and simulation optimization for each experimental setting. I also present the mean and standard deviation of the return obtained when using value iteration and best heuristic policies on 10,000 simulated episodes, each 365 days long following a warm-up period of 100 days.

The wall times reported in Table 4.3 show that, using my approach, the largest cases, with $m = 5$ and $L = 2$, can be solved using value iteration in under 20 minutes. The running time for simulation optimization is approximately constant at two seconds over the different problem sizes. This is consistent with the fact that the parameter space for the heuristic base stock policy is the same for each experimental setting, because the maximum order quantity $A_{\max}$ does not change.

As expected, higher mean returns were given by a FIFO issuing policy compared to a LIFO issuing policy and as m increases due to lower wastage. The optimality gap is consistently higher for experiments with a longer lead time, suggesting that the age profile of the stock is more important when lead times are longer.

See Appendix G.1 for the best parameters for the heuristic policy and KPIs for

each experiment. In the cases with the largest optimality gaps, where $m = L = 2$, the improvement is driven by an improved service level at similar levels of stock holding.

								Value iteration			**Simulation optimization**			
m	Exp	L	C_w	Issuing policy	$\lvert\mathbb{S}\rvert$	$\lvert\mathbb{A}\rvert$	$\lvert\Omega\rvert$	WT (s)	Return *Mean*	*(s.d.)*	WT (s)	Return *Mean*	*(s.d.)*	Optimality gap (%)
2	1	1	7	LIFO	121	11	101	5	-1,553	(61)	2	-1,565	(62)	0.80
	2	1	7	FIFO	121	11	101	4	-1,457	(59)	2	-1,474	(56)	1.20
	3	1	10	LIFO	121	11	101	5	-1,571	(61)	2	-1,581	(62)	0.64
	4	1	10	FIFO	121	11	101	5	-1,463	(60)	2	-1,485	(61)	1.46
	5	2	7	LIFO	1,331	11	101	5	-1,551	(62)	2	-1,590	(64)	2.49
	6	2	7	FIFO	1,331	11	101	5	-1,461	(58)	2	-1,495	(60)	2.31
	7	2	10	LIFO	1,331	11	101	6	-1,569	(61)	2	-1,606	(64)	2.35
	8	2	10	FIFO	1,331	11	101	5	-1,469	(59)	2	-1,504	(60)	2.41
3	1	1	7	LIFO	1,331	11	101	5	-1,490	(58)	2	-1,500	(59)	0.71
	2	1	7	FIFO	1,331	11	101	5	-1,424	(56)	2	-1,435	(52)	0.74
	3	1	10	LIFO	1,331	11	101	5	-1,498	(61)	2	-1,512	(58)	0.90
	4	1	10	FIFO	1,331	11	101	5	-1,425	(55)	2	-1,436	(52)	0.82
	5	2	7	LIFO	14,641	11	101	13	-1,513	(61)	2	-1,533	(61)	1.32
	6	2	7	FIFO	14,641	11	101	13	-1,435	(56)	2	-1,456	(58)	1.42
	7	2	10	LIFO	14,641	11	101	13	-1,526	(60)	2	-1,544	(61)	1.16
	8	2	10	FIFO	14,641	11	101	13	-1,437	(56)	2	-1,457	(58)	1.42
4	1	1	7	LIFO	14,641	11	101	14	-1,459	(56)	2	-1,476	(54)	1.15
	2	1	7	FIFO	14,641	11	101	14	-1,422	(56)	2	-1,430	(52)	0.54
	3	1	10	LIFO	14,641	11	101	14	-1,465	(56)	2	-1,481	(60)	1.08
	4	1	10	FIFO	14,641	11	101	14	-1,422	(56)	2	-1,430	(52)	0.54
	5	2	7	LIFO	161,051	11	101	111	-1,480	(59)	2	-1,496	(59)	1.07
	6	2	7	FIFO	161,051	11	101	110	-1,432	(55)	2	-1,453	(58)	1.44
	7	2	10	LIFO	161,051	11	101	110	-1,489	(59)	2	-1,505	(58)	1.07
	8	2	10	FIFO	161,051	11	101	109	-1,432	(55)	2	-1,453	(58)	1.44
5	1	1	7	LIFO	161,051	11	101	114	-1,443	(55)	2	-1,454	(55)	0.73
	2	1	7	FIFO	161,051	11	101	113	-1,422	(56)	2	-1,430	(52)	0.54
	3	1	10	LIFO	161,051	11	101	114	-1,446	(56)	2	-1,460	(55)	0.94
	4	1	10	FIFO	161,051	11	101	114	-1,422	(56)	2	-1,430	(52)	0.54
	5	2	7	LIFO	1,771,561	11	101	1,191	-1,463	(58)	2	-1,480	(60)	1.22
	6	2	7	FIFO	1,771,561	11	101	1,185	-1,432	(55)	2	-1,453	(58)	1.44
	7	2	10	LIFO	1,771,561	11	101	1,188	-1,467	(58)	2	-1,484	(59)	1.15
	8	2	10	FIFO	1,771,561	11	101	1,190	-1,432	(55)	2	-1,453	(58)	1.44

The return was calculated for each episode and the mean and standard deviation of the return over the 10,000 evaluation episodes is reported. The optimality gap represents the percentage difference in mean return between the value iteration policy and the heuristic policy with parameters fit using simulation optimization. The wall time (WT) is the time taken to fit the policy. The longest wall times, for value iteration when $m = 5$ and $L = 2$, are approximately 20 minutes. Value iteration was considered intractable for experiments where $m > 2$ in the original study.

Table 4.3: Results on Scenario A for all of the experimental settings from De Moor *et al.* [40].

4.6 Scenario B: substitution between two perishable products

4.6.1 Problem description

Hendrix *et al.* [71] applied value iteration and simulation optimization to fit replenishment policies for two perishable inventory problems: a single-product scenario that is similar to Scenario A and a scenario with two products and the potential for substitution which I consider here as Scenario B. In Scenario B there are two perishable products, product A and product B, with the same fixed, known useful life m. Some customers who want product B are willing to accept product A instead if product B is out of stock. The lead time $L = 1$ and therefore there is no in-transit component to the state.

At the start of each day t, the agent observes state S_t, the current inventory of each product in stock split by remaining useful life, and places a replenishment order. The action consists of two elements, one order for each product: $A_t = [A_t^a, A_t^b]$ where $A_t^a \in \{0, 1, ..., A_{\max}^a\}$ and $A_t^b \in \{0, 1, ..., A_{\max}^b\}$. Demand for day t is sampled from independent Poisson distributions for each product, parameterized respectively by mean demand μ^a and μ^b, and is initially filled for each product independently using a FIFO issuing policy. Some customers with unmet demand for product B may be willing to accept product A instead. The substitution demand is sampled from a binomial distribution, with a probability of accepting substitution ρ and a number of trials equal to the unmet demand for product B. After demand for product A has been filled as far as possible, demand for product B willing to accept product A is filled by any remaining units of product A using a FIFO issuing policy. At the end of the day, the state is updated to reflect the ageing of stock, and the reward, R_{t+1} is calculated. The reward function comprises revenue per unit sold (C_r^a, C_r^b) and variable order cost (C_v^a, C_v^b) for each product. The order placed on day t is received immediately prior to the start of day $t + 1$ and is included in the stock element of state S_{t+1}.

The daily demand and willingness to accept substitution are both stochastic.

I capture the effect of both by considering the stochastic element in the transition to be the number of units issued for each product type: H^a and H^b. The state transition and the reward are deterministic given a state-action pair and the number of units issued of product A and of product B. The set of possible realizations of the stochastic elements is:

$$\Omega = \{(h^a, h^b)\} \qquad h^a \in \{0, 1, ..., H^a_{\max} = mA^a_{\max}\} \qquad h^b \in \{0, 1, ..., H^b_{\max} = mA^b_{\max}\} \tag{4.6}$$

The initial value function, $V_0(s)$, is set to the expected sales revenue for state s with I^a units of product A and I^b units of product B in stock. This is an infinite horizon problem with no discount factor, and therefore I used the convergence test specified in Hendrix *et al.* [71], which stops value iteration when the value of each state is changing by approximately the same amount on each iteration. If the value of every state is changing by the same amount there will be no further changes to the best action for each state, indicating a stable estimate of the optimal policy.

Hendrix *et al.* [71] considered products with a maximum useful life m of two and three periods and evaluated two experimental settings for each value of m. For experiments 1 and 2 with $m = 3$, where the maximum order quantities were set based on the newsvendor model, they reported that it was not possible to complete value iteration within one week. They therefore repeated the two experiments for $m = 3$ with lower values of $A^a_{\max}$ and $A^b_{\max}$. With this adjustment, one of the cases could be completed within 80 hours, while the other could still not be solved within a week. I demonstrate that using JAX and a consumer-grade GPU it is feasible to obtain the optimal policy for all of these settings and report the wall time required to run value iteration for each experiment. Additionally, to investigate how my method can benefit from more powerful GPUs, and how it scales to multiple GPUs, I report the wall times for running the largest problem on one, two and four Nvidia A100 40GB GPUs.

Separately, I consider the experimental settings used by Ortega *et al.* [196] to

evaluate their GPU-accelerated method, in which value iteration was always run for 100 iterations instead of to convergence. The different experiments evaluate mean daily demands between five and seven with maximum order quantities based on the newsvendor model.

In each case, I compared the performance of the policy computed using value iteration with the performance of the modified base stock policy used by Hendrix *et al.* [71], based on the work of Haijema and Minner [42], which has an order-up-to level parameter for each product: $\mathtt{S}^a$ and $\mathtt{S}^b$. The order quantity for each product is determined considering only the on hand inventory of that product and includes an adjustment for expected waste. The order quantity on day t, given total stock on hand I_t^a and I_t^a and stock that expires at the end of the current period $X_{1,t}^a$ and $X_{1,t}^b$, is:

$$A_t = \left[A_t^a, A_t^b\right] = \left[\left[\mathtt{S}^a - I_t^a + \left[X_{1,t}^a - \mu^a\right]^+\right]^+, \left[\mathtt{S}^b - I_t^b + \left[X_{1,t}^b - \mu^b\right]^+\right]^+\right] \quad (4.7)$$

There are two parameters, and I used Optuna's NSGA-II sampler to search the parameter space $\mathtt{S}^a \in \{0, 1, ..., \mathtt{S}^a_{\max} = 2A^a_{\max}\}$ and $\mathtt{S}^b \in \{0, 1, ..., \mathtt{S}^b_{\max} = 2A^b_{\max}\}$. I considered values of the order-up-to level up to twice the maximum order quantity used for value iteration because Hendrix *et al.* [71] reported best values of $\mathtt{S}$ that were higher than the values of $A_{\max}$ specified for value iteration for some of their experiments. I compared the modified base stock policy that achieved the highest mean return, characterized by the pair of parameters $\left(\mathtt{S}^a, \mathtt{S}^b\right)_{\text{best}}$, to the value iteration policy.

See Appendix F.2 for additional information about Scenario B.

4.6.2 Results

I present results for the experimental settings for the two product scenario from Hendrix *et al.* [71] in Table 4.4. The wall times in Table 4.4 show that, using my method, value iteration can be used to find the optimal policy for all four settings of the two product scenario with $m = 3$ in under 3.2 hours. Hendrix *et al.* [71] reported

that, for $m = 3$, value iteration did not converge within a week for experiments 1, 2 and 3 using a MATLAB implementation and experiment 4 converged in 80 hours. My implementation of experiment 4, running on a consumer-grade GPU, converges in just over two minutes: more than 2000$\times$ faster.

I present results for the four experimental settings, P1 to P4, from Ortega *et al.* [196] in Table 4.5. The wall times for my approach are at least six times faster than those reported by Ortega *et al.* [196] for all four settings. It is not possible to conclude on the relative performance of my method and the GPU-accelerated method from Ortega *et al.* [196] without running both implementations on the same hardware and accounting for the difference between up-front and JIT compilation. However, the results suggest that my method is at least competitive with a custom CUDA implementation of value iteration for the two product case while requiring less specialist knowledge of GPU programming.

Simulation optimization scales well to larger problems, with wall times less than one minute for all of the experimental settings. The optimality gap is never greater than 1%, and reduces as both mean demand and the maximum useful life increase. This suggests that there is a limited advantage to making ordering decisions based on the stock of both products, compared to making independent decisions for each product using a simple heuristic policy, under the reward function and substitution process proposed by Hendrix *et al.* [71].

Figure 4.1 illustrates the clear benefits of both more powerful GPUs, and of using multiple GPUs. Using a single Nvidia A100 40GB GPU, experiment 1 when $m = 3$ can be run in 4,838s: 2.4$\times$ faster than the Nvidia RTX 3060 in my local machine. The A100 40GB has more GPU RAM and more CUDA cores than the RTX 3060 [215], 40GB vs 16GB and 6,912 vs 3,584 respectively, which means that it can update the value function for a larger number of states simultaneously. Using two A100 40GB GPUs is 1.8$\times$ faster than one, and using four A100 40GB GPUs is 2.8$\times$ faster than one, demonstrating how the wall time can further reduced and how larger problems can be solved with additional computational resources using exactly the same code. The wall time does not scale linearly with the number

of GPUs because not all of the operations are conducted in parallel on GPU - for example at the end of each iteration the updated values need to be aggregated and transferred to each device, and the values are transferred back to CPU in order to save a checkpoint.

See Appendix G.2 for the best parameters for the heuristic policy and KPIs for each experiment. In the experiments with the largest optimality gaps, when $m = 2$, fewer units of product B are ordered under the optimal policy resulting in a lower service level but also less wastage and less holding for product B.

									Value iteration			**Simulation optimization**			
m	Exp	μ^a	μ^b	$A^a_{\max}$	$A^b_{\max}$	$\lvert\mathbb{S}\rvert$	$\lvert\mathbb{A}\rvert$	$\lvert\Omega\rvert$	WT (s)	Return *Mean*	*(s.d.)*	WT (s)	Return *Mean*	*(s.d.)*	Optimality gap (%)
2	1	5	5	10	10	14,641	121	441	5	1,644	(33)	24	1,632	(34)	0.70
	2	7	3	14	6	11,025	105	377	4	1,650	(33)	23	1,639	(34)	0.67
3	1	5	5	15	15	16,777,216	256	2,116	11,496	1,761	(32)	33	1,758	(32)	0.16
	2	7	3	21	9	10,648,000	220	1,792	4,013	1,762	(32)	44	1,759	(32)	0.18
	3	5	5	13	13	7,529,536	196	1,600	3,058	1,761	(32)	32	1,758	(32)	0.16
	4	7	3	20	4	1,157,625	105	793	134	1,762	(32)	43	1,759	(32)	0.17

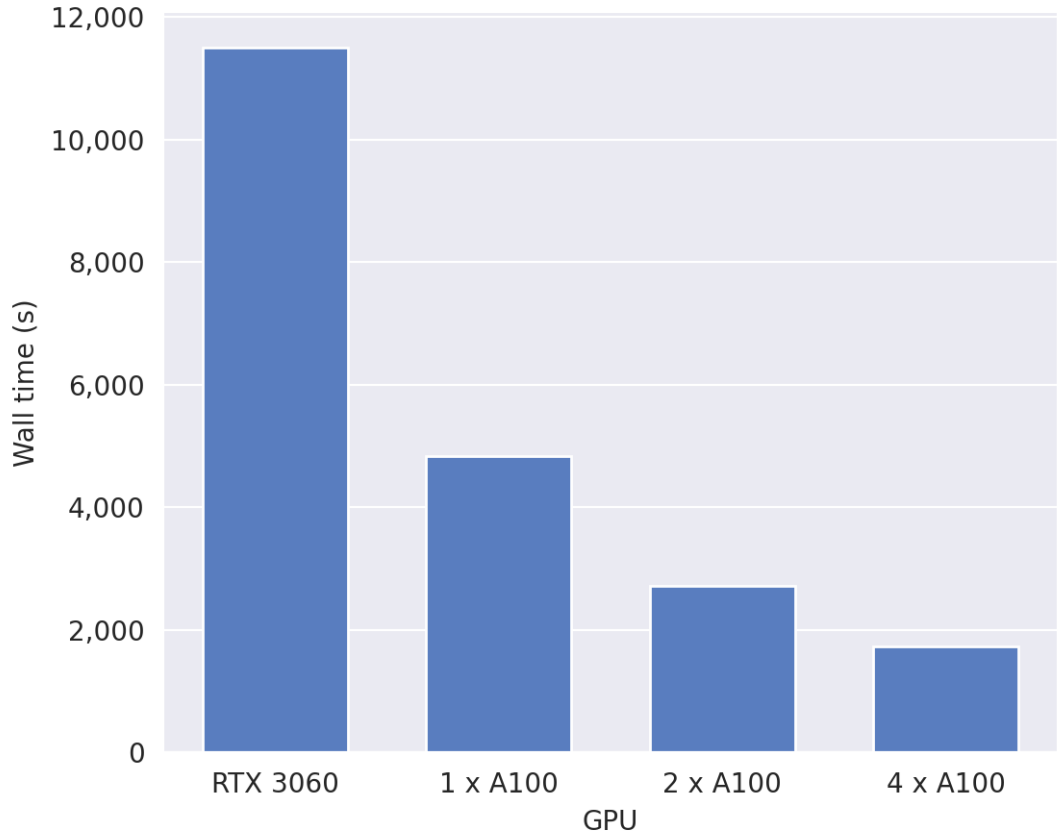
The return was calculated for each episode and the mean and standard deviation of the return over the 10,000 evaluation episodes is reported. The optimality gap represents the percentage difference in mean return between the value iteration policy and the heuristic policy with parameters fit using simulation optimization. The wall time (WT) is the time taken to fit the policy. The longest wall time, for value iteration on experiment 1 when $m = 3$, is approximately 3.2 hours. Value iteration could not be completed within a week for experiments 1, 2 and 3, and required 80 hours for experiment 4, when $m = 3$ in Hendrix *et al.* [71].

Table 4.4: Results on Scenario B for all of the experimental settings from Hendrix *et al.* [71].

									Value iteration			**Simulation optimization**			
m	Exp	μ^a	μ^b	$A^a_{\max}$	$A^b_{\max}$	$\lvert\mathbb{S}\rvert$	$\lvert\mathbb{A}\rvert$	$\lvert\Omega\rvert$	WT (s)	Return *Mean*	*(s.d.)*	WT (s)	Return *Mean*	*(s.d.)*	Optimality gap (%)
2	P1	5	5	10	10	14,641	121	441	11	1,644	(33)	25	1,632	(34)	0.70
	P2	5	6	10	12	20,449	143	525	18	1,826	(35)	31	1,816	(34)	0.58
	P3	6	6	12	12	28,561	169	625	27	2,011	(36)	31	2,000	(37)	0.55
	P4	7	7	13	13	38,416	196	729	56	2,379	(39)	29	2,368	(40)	0.46

Value iteration was run for 100 iterations for each experiment instead of to convergence. Aside from this, experiment P1 is the same as experiment 1 with $m = 2$ in Table 4.4. The return was calculated for each episode and the mean and standard deviation of the return over the 10,000 evaluation episodes is reported. The optimality gap represents the percentage difference in mean return between the value iteration policy and the heuristic policy with parameters fit using simulation optimization. The wall time (WT) is the time taken to fit the policy. The longest wall time, for value iteration on experiment P4, is approximately one minute. Value iteration was tractable for all of these settings in the original study but wall times using my method are at least six times faster than those reported by Ortega *et al.* [196]. This improvement may be at least partially attributable to hardware differences.

Table 4.5: Results on Scenario B for all of the experimental settings used by Ortega *et al.* [196] to test their GPU-accelerated implementation of value iteration.



The Nvidia GeForce RTX 3060 is a consumer-grade GPU. The Nvidia A100 40GB is a data-centre grade GPU. JAX enables my implementation to be run on multiple identical GPUs without any code changes.

Figure 4.1: Comparison of the wall time required to run value iteration for experiment 1 with $m = 3$ for Scenario B using the same code and different GPUs.

4.7 Scenario C: periodic demand and uncertain useful life on arrival

4.7.1 Problem description

Mirjalili [166] described a perishable inventory problem that models the management of platelets in a hospital blood bank. There are two problem features not included in Scenarios A or B. Firstly, the demand is periodic, with an independent demand distribution for each day of the week. Secondly, the remaining useful life of products on arrival is uncertain, and this uncertainty may be exogenous or endogenous (depending on the order quantity). The lead time L is assumed to be zero and therefore there is no in-transit component to the state.

At the start of each day t the agent observes state S_t, which specifies the

day of the week and the current inventory in stock split by remaining useful life, and places a replenishment order $A_t \in \{0, 1, ..., A_{\max}\}$. This order is assumed to arrive instantly. The remaining useful life of the units on arrival is sampled from a multinomial distribution, the parameters of which may depend on the order quantity A_t. Demand for day t, D_t is sampled from a truncated negative binomial distribution and filled from available stock using an OUFO policy. At the end of the day, the state is updated to reflect the ageing of stock and the reward, R_{t+1} is calculated. The reward function comprises four components: a holding cost per unit in stock at the end of the period (C_h), a shortage cost per unit of unmet demand (C_s), a wastage cost per unit that perishes at the end of the period (C_w) and a fixed ordering cost (C_f). Unlike Scenario A which also includes a holding cost, the holding cost is charged on units that expire at the end of the period.

To reduce the number of possible states, I consider a limited case of this problem in which there is a maximum capacity of $A_{\max}$ for stock of each age. If, when an order is received, the sum of units in stock and units received with k days of remaining useful life is greater than $A_{\max}$ then I assume the excess units are not accepted at delivery. The stock level with k days of remaining useful life is therefore at most $A_{\max}$ when demand is sampled. This constraint is not explicitly stated in the original work, but a capacity constraint is necessary for consistency with the calculation of the total number of states in Mirjalili [166]. Note that there are alternative ways to apply the constraint (e.g. by discarding excess units at the end of each day along with wastage) and these may have different optimal policies.

The stochastic elements in the transition are the daily demand, D, and the age profile of the units received to fill the order placed at the start of the day: $\underline{Y} = [Y_m, Y_{m-1}, ..., Y_1]$. The state transition and the reward are deterministic given a state-action pair, the daily demand, and the age profile of the units received. The set of possible realizations of the stochastic elements is:

$$\begin{aligned} \Omega = \{(d, \underline{y})\} \quad & d \in \{0, 1, ..., D_{\max}\} \\ & y_i \in \{0, 1, ..., A_{\max}\}, \quad \forall i \in \{1, 2, ..., m\} \\ & \sum_{i=1}^{m} y_i \leq A_{\max} \end{aligned} \tag{4.8}$$

The initial value function $V_0(s)$ was initialized at zero for every state. Mirjalili [166] did not specify a particular convergence test for his value iteration experiments. The problem is periodic, with a discount factor, and therefore I use a convergence test based on those described in Su and Deininger [182] which stops value iteration when the undiscounted change in the value function over a period (in this case, seven days) is approximately the same for every state. As in Scenario B, when the change in value is approximately the same for every state there will be no further changes to the best action for every state, and hence, the estimated optimal policy is stable.

Mirjalili [166] considered products with a maximum useful life of three, five or eight periods and stated that, due to the large state space, value iteration was intractable for this problem when $m \geq 5$. I was able to run value iteration when $m = 5$, but not when $m = 8$. For each value of m, Mirjalili [166] investigated five different settings for the distribution of useful life on arrival (one where the uncertainty was exogenous, and four where the uncertainty was endogenous). For each of these five settings, he evaluated six combinations of C_f and C_w. My objective was to demonstrate the feasibility of my approach and therefore, given the large number of experiments and long wall times when $m = 5$, I ran two experiments for each value of m: one where the uncertainty in useful life on arrival was exogenous and one where it was endogenous. For $m = 5$, I selected the settings from Mirjalili [166] that are based on real, observed data from a network of hospitals in Ontario, Canada instead of the additional settings created for sensitivity analysis. I report the wall time required to run value iteration for each experiment.

I compared the policy from value iteration with an $(\mathtt{s}, \mathtt{S})$ policy. Mirjalili

[166] did not fit heuristic policies, but suggested (s, S) as an example of a suitable heuristic policy for future work: the addition of a fixed ordering cost to the reward function means that it may be beneficial to include the reorder point parameter s to avoid uneconomically small orders. I fit one pair of s and S for each day of the week, a total of 14 parameters. The order quantity on day t, given that the day of the week is τ and the total current stock on hand is I_t is:

$$A_t = \begin{cases} [\mathtt{S}^\tau - I_t]^+ & \text{if } I_t \leq \mathtt{s}^\tau \\ 0 & \text{if } I_t > \mathtt{s}^\tau \end{cases} \tag{4.9}$$

where $(\mathtt{s}^\tau, \mathtt{S}^\tau)$ is the pair of parameters for day of the week τ.

I used Optuna's NSGA-II sampler to search for combinations of $\mathtt{s}^\tau \in \{0, 1, ..., \mathtt{s}_{\max} = A_{\max}\}$ and $\mathtt{S}^\tau \in \{0, 1, ..., \mathtt{S}_{\max} = A_{\max}\}$ $\forall \tau \in \{0, 1, .., 6\}$. This heuristic policy has a hard constraint that $\mathtt{s}^\tau < \mathtt{S}^\tau$ $\forall \tau \in \{0, 1, .., 6\}$. Optuna does not support using hard constraints to restrict the search space, so I enforced the constraint by only allowing a non-zero order to be placed if the constraint was met. I compared the heuristic policy that achieved the highest mean return, characterized by parameters $((\mathtt{s}^0, \mathtt{S}^0), ..., (\mathtt{s}^6, \mathtt{S}^6))_{\text{best}}$, to the value iteration policy.

See Appendix F.3 for additional information about Scenario C. This scenario was taken from Chapter 6 of Mirjalili [166], but a preprint based on that chapter has since been released [164].

4.7.2 Results

In Table 4.6 I present the results for the experimental settings from Mirjalili [166] that I selected, two for each value of m. Using my method, it is possible to find the optimal policy using value iteration for $m = 3$ and $m = 5$ while accounting for uncertainty in useful life on arrival. The experiments where $m = 5$ represent a real world problem: Mirjalili [166] fit the parameters for the demand distribution and distribution of useful life on arrival to observed data from a network of hospitals in Ontario, Canada. This is an important application of my value iteration method, demonstrating that it can be used to find optimal policies for problems of

a realistic size. The alternative experimental settings evaluated by Mirjalili [166] but not repeated here have the same numbers of states, actions and possible random outcomes and therefore I would expect the wall times to be of a similar order as corresponding experiments reported in Table 4.6.

I was unable to complete value iteration when $m = 8$. This problem has over 12.6 billion possible states, even with the restriction that I placed on the maximum stock holding of each age, and over 65 million possible random outcomes. It was not feasible to store the state array in the memory of my local machine, let alone run value iteration. However, I was able to fit a heuristic policy using simulation optimization in less than 20 minutes.

The simulation optimization experiments for this scenario take longer than those of the other scenarios, between five and 20 minutes. This is due to the large number of possible combinations of parameters, because the heuristic policy requires seven pairs of parameters $(\mathtt{s}, \mathtt{S})$, one for each weekday. Optuna does not support restricting the search space based on the constraint that $\mathtt{s}^\tau < \mathtt{S}^\tau \ \forall \tau \in \{0, 1, .., 6\}$ and therefore the size of the search space for each experiment is $(A_{\max} + 1)^{14} = 3.2 \times 10^{18}$, compared to only 11 possible parameters for the base stock policy used for Scenario A and fewer than 1,000 possible combinations of parameters for even the largest settings from Scenario B. The heuristic policies perform well, with a maximum optimality gap of 1.22%. See Appendix G.3 for the best parameters for the heuristic policy and KPIs for each experiment. For the experiments with the highest optimality gaps, when $m = 5$, the optimal policy achieves better performance when the distribution of remaining useful life on arrival is exogenous due to a higher service level at the cost of more wastage and higher stock holding. Conversely, when the distribution of remaining useful life on arrival is endogenous, the optimal policy achieves a lower service level, but with lower wastage and stock holding.

In Figure 4.2 I draw together the results from Scenario C with those from the preceding scenarios, and plot the optimality gap between the heuristic policy that achieved the highest mean return and the value iteration policy against the wall time

m	Exp	Uncertainty in useful life	$\lvert\mathbb{S}\rvert$	$\lvert\mathbb{A}\rvert$	$\lvert\Omega\rvert$	Value iteration			Simulation optimization			
						WT (s)	Return *Mean*	*(s.d.)*	WT (s)	Return *Mean*	*(s.d.)*	Optimality gap (%)
3	1	Exogenous	3,087	21	37,191	15	-410	(62)	507	-411	(63)	0.26
	2	Endogenous	3,087	21	37,191	17	-349	(53)	305	-352	(55)	1.04
5	1	Exogenous	1,361,367	21	1,115,730	178,078	-312	(46)	514	-313	(50)	0.34
	2	Endogenous	1,361,367	21	1,115,730	178,023	-312	(47)	393	-315	(46)	1.22
8	1	Exogenous	12,607,619,787	21	65,270,205	—	—	—	618	-293	(42)	—
	2	Endogenous	12,607,619,787	21	65,270,205	—	—	—	972	-297	(43)	—

The return was calculated for each episode and the mean and standard deviation of the return over the 10,000 evaluation episodes is reported. The optimality gap represents the percentage difference in mean return between the value iteration policy and the heuristic policy with parameters fit using simulation optimization. The wall time (WT) is the time taken to fit the policy. The longest wall time, for value iteration on experiment 1 when $m = 5$, is approximately 49.5 hours. Value iteration was considered intractable for the experiments where $m \geq 5$ in the original study. I was able to use value iteration when $m = 5$, but not when $m = 8$.

Table 4.6: Results on Scenario C for a subset of the experimental settings from Mirjalili [166]: two examples for each value of maximum useful life m.

required for value iteration.

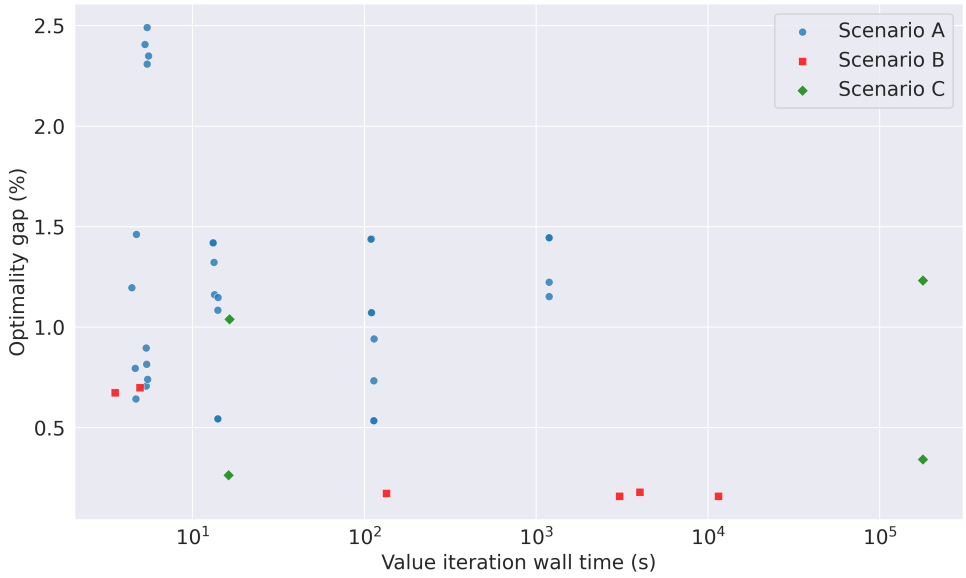


Figure 4.2: The optimality gap between the best heuristic policy and the value iteration policy plotted against the wall time required to run value iteration for the experiments from Scenarios A, B and C.

4.8 Scenario from Chapter 3: periodic demand

4.8.1 Problem description

In Chapter 3 I adapted the platelet replenishment scenario described by Rajendran and Srinivas [45] into an RL environment, and compared the performance

of heuristic policies with parameters fit using stochastic mixed integer linear programming, DRL policies, and the optimal policy computed using Q-value iteration. Rajendran and Srinivas [45] made no comment about the feasibility of computing the optimal policy using a dynamic programming approach and therefore this scenario was not included in the initial analysis for this chapter. However, I note in Section 3.4.7.3 that while the optimal policy takes the form of an $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy (and could also be represented by either an $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ or a $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policy), the optimal parameters were not identified using stochastic mixed integer linear programming. I therefore decided to investigate whether my GPU-accelerated approach to simulation optimization would perform better, and also repeated the value iteration using the new GPU-accelerated implementation in order to compare the wall time and confirm that both implementations gave the same policies.

The problem is described in Section 3.4.1. This scenario is single-product, single-echelon, periodic review inventory replenishment problem with fixed useful life $m = 3$ and fixed lead time $L = 0$.

The stochastic element in the transition is the daily demand D, $\mathbb{\Omega} = \{0, 1, ..., \infty\}$. The state transition and the reward are deterministic given a state-action pair and the realization of the daily demand. Daily demand is modelled by a Poisson distribution, with a mean for each day of the week. As in the Q-value iteration approach in Chapter 3, I truncated the demand distribution at $D_{\max} = 60$, such that $\mathbb{\Omega} = \{0, 1, ..., 60\}$, for the purposes of implementation.

The initial value function $V_0(s)$ was initialized at zero for every state. The problem is periodic and does not include a discount factor, and I therefore used the same convergence test as for Scenario C.

I compared the policy from value iteration with each of the four heuristic policies from Rajendran and Srinivas [45]: $(\mathtt{s}, \mathtt{S})^{\dagger}$, $(\mathtt{s}, \mathtt{Q})$, $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ which are set out in Equations 3.6 to 3.9 respectively.

There are 14 parameters each to fit for the $(\mathtt{s}, \mathtt{S})^{\dagger}$ and $(\mathtt{s}, \mathtt{Q})$ policies, and 28 parameters each to fit for the $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policies. Each

parameter could take a value between 0 and 60, consistent with the maximum order quantity allowed in Chapter 3. I used Optuna's NSGA-II sampler to search the parameter space and compared the version of each policy that achieved the highest return to the value iteration policy, on the 1,000 evaluation episodes used in the corresponding experiments in Section 3.4, using the OpenAI gym-based RL environment developed in Chapter 3. In this scenario, for consistency with the experiments in Section 3.4, I did not use a warm-up period. Due to the larger number of heuristic policy parameters for certain heuristic policies considered for this scenario, I terminated the NSGA-II search procedure when the best combination of parameters had not changed for 20 generations, or when 200 generations had been completed.

There are hard constraints on these heuristic policies:

$$\mathtt{s}^\tau < \mathtt{S}^\tau \quad \forall \tau \in \{0, 1, ..., 6\} \tag{4.10}$$

$$\alpha^\tau < \mathtt{s}^\tau \quad \forall \tau \in \{0, 1, ..., 6\} \tag{4.11}$$

$$\beta^\tau < \mathtt{s}^\tau \quad \forall \tau \in \{0, 1, ..., 6\} \tag{4.12}$$

I enforced the constraints on the heuristic policy for Scenario C by only allowing an order of zero units to be placed if the constraint was violated. When applying the same method with the same settings, I found that NSGA-II did not find feasible values for the $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ and $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policies, likely due to the much larger number of parameters. Therefore, for this scenario, I enforced the constraints by clipping the values of $\mathtt{s}$, α and β:

$$\mathtt{s} = \min(\mathtt{s}^\tau, \mathtt{S}^\tau - 1) \tag{4.13}$$

$$\alpha = \min(\alpha^\tau, \mathtt{s}^\tau - 1) \tag{4.14}$$

$$\beta = \min(\beta^\tau, \mathtt{s}^\tau - 1) \tag{4.15}$$

4.8.2 Results

As expected, the policy obtained using GPU-accelerated value iteration was the same as the obtained using Q-value iteration in Section 3.4.7. The GPU-accelerated value iteration ran in 4s (including extracting and saving the optimal policy), compared with over 6 hours for the CPU-based Q-value iteration approach. Both of these experiments were performed on the same desktop workstation (see Section 4.4.6). I adopted a relatively naive parallelization strategy for the Q-valuation iteration: as described in Section 3.4.7.1 the code was only optimized to the point where it produced a result in less than a day. The difference in wall time between the two approaches could almost certainly be reduced by additional optimization of the CPU-based approach.

I set out the parameters for the heuristic policies fit using simulation optimization in Table 4.7. In Table 4.8 I present the performance the four heuristic policies with parameters fit using simulation optimization, in the same format as Table 3.7. In Table 4.9 I compared the optimality gap between the optimal policy (computed using Q-value iteration and value iteration) and the heuristic policy parameters fit using stochastic mixed integer linear programming and GPU-accelerated simulation optimization.

The heuristic policy parameters fit using GPU-accelerated simulation optimization achieved lower mean daily cost on the evaluation episodes than the corresponding parameters fit using stochastic mixed integer linear programming for each of the four heuristic policies. For the three heuristic policies capable of representing the optimal policy (so, not $(\mathtt{s}, \mathtt{Q})$), the largest optimality gap was 0.04%. Under the parameters reported in Table 4.7, the $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policy effectively reduces to an $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy due to the low values of the parameter β for each weekday.

The wall time for GPU-accelerated simulation optimization ranged between 103s for the $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy and 237s for the $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policy. GPU-accelerated simulation optimization therefore gave better results for all four policies in under 4 minutes than allowing the CPU-based stochastic mixed integer linear programming

solver to run for 36 hours. This is also substantially shorter than the wall times reported by Rajendran and Srinivas [45] for stochastic mixed integer linear programming, without the additional constraints I describe in Appendix D, of between 10 minutes and 1 hour. This approach, using consumer-grade GPU hardware, can therefore achieve better results in less time.

Policy	Parameter	Weekday						
		Mon	Tue	Wed	Thu	Fri	Sat	Sun
$(s, S)^{\dagger}$	s	41	41	41	38	35	31	39
	S	48	50	47	51	36	38	49
(s, Q)	s	39	41	38	42	28	28	36
	Q	41	40	39	44	27	30	40
(s, S, α, Q)	s	42	41	37	50	33	37	47
	S	48	50	48	51	36	38	48
	α	41	40	32	49	31	36	28
	Q	25	6	17	6	5	43	20
(s, S, β, Q)	s	41	47	47	46	33	28	47
	S	48	50	48	51	36	37	48
	β	0	0	1	1	0	1	0
	Q	7	49	48	51	49	38	45

Table 4.7: Heuristic replenishment policy parameters identified using GPU-accelerated simulation optimization for the scenario from Chapter 3

Policy	Mean daily cost						Mean KPIs		
	Fixed	Variable	Holding	Shortage	Wastage	Total	Service level (%)	Days with shortage (%)	Wastage rate (%)
Value iteration	225	22,961	1,337	423	0	24,946	99.6	4.1	0.0
$(s, S)^{\dagger}$	225	22,961	1,338	433	0	24,957	99.6	4.2	0.0
(s, Q)	210	22,861	2,121	976	2	26,171	99.2	7.1	0.0
(s, S, α, Q)	225	22,961	1,337	423	0	24,946	99.6	4.1	0.0
(s, S, β, Q)	225	22,957	1,320	445	0	24,947	99.6	4.3	0.0

The daily cost components, daily total cost and KPIs were calculated for each episode. The mean of the daily cost components, daily total cost and KPIs over the 1,000 evaluation episodes is reported.

Table 4.8: Mean daily costs incurred and associated KPIs given by the four heuristic replenishment policies with parameters fit using GPU-accelerated simulation optimization.

4.9 Discussion

I have found JAX to provide an effective way to expand the scale of perishable inventory problems for which value iteration is tractable, using only consumer-grade hardware. By developing a faster, parallel implementation of value iteration I have

Policy	Fit using stochastic programming		Fit using simulation optimization		
	Mean daily cost	Optimality gap (%)	Mean daily cost	Optimality gap (%)	Δ optimality gap
$(s, S)^{\dagger}$	25,254	1.23	24,957	0.04	1.19pp
(s, Q)	28,401	13.85	26,171	4.91	8.94pp
(s, S, α, Q)	26,023	4.32	24,946	0.00	4.32pp
(s, S, β, Q)	25,414	1.88	24,947	0.00	1.87pp

The daily total cost was calculated for each episode, and the mean over the 1,000 evaluation episodes is reported. The optimality gap represents the percentage difference in mean return between the value iteration policy and the heuristic policy, with parameters fit using either stochastic programming or simulation optimization. The difference between the optimality gaps is reported in percentage points. The policy parameters fit using stochastic programming are reported in Table 3.2 in Chapter 3 and the policy parameters fit using simulation optimization are reported in Table 4.7.

Table 4.9: Comparison between the performance of each heuristic replenishment policy using parameters fit using stochastic programming in Chapter 3 and parameters fit using simulation optimization.

been able to see further: obtaining the optimal policies for large problems that were previously considered infeasible or impractical. A key benefit of finding these policies is being able to properly quantify the performance of heuristic and approximate policies, in terms of the optimality gap and difference in KPIs, on larger problem instances. Additionally, using these metrics, it is possible to investigate how the relative performance of heuristic and approximate policies scales with properties that influence the problem size. This may support the development of new heuristics, and determining the utility of RL and other approximate methods.

Expanding the range of problems to which value iteration can be applied is not just a matter of considering settings with greater demand, more products, or products with a longer useful life. It also enables the incorporation of complexity that might otherwise be neglected due to its effect on the computational tractability such as substitution and endogenous uncertainty in the useful life of products on arrival. An "optimal" policy fit using value iteration is optimal for the situation as modelled, but may not perform well in practice if the model neglects challenging aspects of the real problem. For example, Mirjalili [166] reported large optimality gaps, with an average of 51%, when policies computed using value iteration under the assumption that all stock arrived fresh were applied to a scenario with endogenous uncertainty in useful life.

The optimality gap in the experiments on Scenarios A, B and C was never

larger than 2.5%, and in the experiments from Scenario A and Scenario B the optimality gap decreased as the demand and/or maximum useful life of the product increased. This is encouraging because it suggests that in some circumstances where the problem size remains too large for value iteration there may actually be little to gain by using the optimal policy over one of these heuristic policies. In Scenario C, the optimality gaps increased as the maximum useful life increased, suggesting that the heuristic policies may be less effective on larger problems when the stock is not always fresh on arrival.

Additionally, I found that my GPU-accelerated approach to simulation optimization identified better parameters for the four heuristic policies from Rajendran and Srinivas [45] than the stochastic mixed integer linear programming approach described in that paper and reimplemented in Chapter 3. By running the simulations in parallel on GPU, the simulation optimization approach was able to evaluate the performance of a set of candidate heuristic parameters on 4,000 demand trajectories (each 365 days long), instead of the 20, 30-day long demand trajectories used for the stochastic programming, during the fitting process. Increasing the number or length of scenarios for stochastic mixed integer linear programming is challenging due to the need for additional decision variables, and the computational time required to solve the equivalent deterministic problems grows exponentially with the number of scenarios [216]. Another advantage of simulation optimization is the flexibility: there is no need to frame the problem in terms of linear constraints.

One limitation of this work is that I have not compared the performance of the GPU-accelerated methods implemented in JAX against a CPU-based approach. As I discuss in Section 4.3, previous studies have established the performance advantages of a GPU-based approach to value iteration over a CPU-based approach for problems on which both are feasible. My primary aim was therefore to develop an accessible method for solving problems that had been described in recent studies as impractical or infeasible using CPU-based approaches. Accepting the claims of prior work, the expected outcome of CPU-benchmarking for the largest problems

would be to expend a large amount of computational time just to establish some time period within which these problems could not be solved on CPU. For the smaller problem sizes, a remaining challenge would be selecting the most appropriate benchmark: the studies from which the scenarios were drawn used a range of different languages (MATLAB, Python, C) and I would expect the performance under these different languages (and, particularly for Python, the choice of libraries) to vary significantly. I therefore chose to focus on comparing the performance metrics and wall time of value iteration and simulation optimization approaches, and report wall times for realistic use cases (e.g. including JIT compilation and recording checkpoints) on affordable hardware which I believe will be most relevant to researchers considering applying this computational approach on their own problems.

A second limitation is that in my simulation optimization experiments I only used GPUs to run the simulated policy rollouts. The heuristic search methods for proposing the next sets of candidate parameters are CPU-based. I did not find Optuna's NSGA-II sampler to be a bottleneck, but during preliminary experiments I found that some alternative methods took longer to propose the next set of candidate parameters than was required to evaluate them on simulated rollouts. In future work, the optimization process suggesting parameters could also be run on GPU similar to the work of Lau and Srinivasan [209] and recent work using evolutionary strategies on GPUs to search for neural network parameters [121]. The gymnax-based simulators would also be well suited to ranking and selection methods because it would be straightforward to run a small number of rollouts for a large number of possible parameters in parallel and then, at a second stage, run a large number of rollouts for the most competitive parameters in parallel to obtain more accurate estimates of their performance.

One of the main contributions of this work is to demonstrate an accessible way of using GPUs to accelerate value iteration and simulation optimization and therefore solve larger problems that are closer to those faced in reality. On the software side, I implemented my approach using the relatively high-level JAX

API and relied on the XLA compiler to efficiently utilize GPU hardware. On the hardware side, I primarily report results on a consumer-grade GPU, and make available a Google Colab notebook so that my experiments can be reproduced at no cost using cloud-based computational resources. However, a significant strength of JAX is support for easily distributing a workload over multiple identical GPU devices using the pmap function transformation and I discuss in Section 4.6 how additional devices can be used to further reduce the wall time and potentially make even larger problems tractable. Modern cloud computing platforms provide on-demand access to data-centre grade GPUs, including the A100 40GB GPU I used to run the scaling experiments in Section 4.6. At the time of writing in June 2024, a single A100 40GB GPU is available on-demand for $3.67 per hour and four A100 40GB GPUs are available for $14.68 per hour through Google Cloud Platform [217]. This may provide a cost-effective way for research teams without access to local high-performance computing resources to investigate problems that are too large for freely available or consumer-grade GPU hardware.

Future hardware development will make value iteration feasible for even larger problems. In addition to future generations of GPUs, one promising direction is Field Programmable Gate Arrays (FPGAs): integrated circuits that can be reprogrammed to customize the hardware to implement a specific algorithm, including value iteration [218]. Customising the hardware currently requires specialist knowledge but, just as machine learning frameworks and higher-level tools have made GPU programming more accessible, Peri [218] suggests that FPGA compilers able to translate high level code into customized circuit designs may facilitate wider adoption.

As is appropriate given the topic of this thesis, I have focused on perishable inventory management with problem features that are relevant to managing blood product inventory, but my computational approach has much wider applicability. For each scenario I created a custom subclass of my base `ValueIterationRunner` class and a custom subclass of the gymnax RL environment to simulate the MDP, each with methods to implement the scenario-specific logic. This approach could

be followed for other problems that can be modelled as an MDP and where finding the optimal policy for larger settings of interest has recently been described as infeasible or impractical [219, 220, 221]. For other problems, where realistic settings are likely to remain infeasible for the foreseeable future, GPU-accelerated value iteration may be useful for understanding the scaling properties of proposed heuristic policies on a wider range of example problems [222, 223]. More broadly, I believe that JAX (and other software libraries originally developed to support deep learning including PyTorch and TensorFlow) offers an efficient way for researchers to run large workloads in parallel on relatively affordable GPU hardware which may support research on a range of operational research problems.

4.10 Conclusion

In this chapter I have shown how a JAX-based approach can expand the range of perishable inventory management problems for which value iteration is practical and feasible, using only consumer-grade GPU hardware. JAX and similar software libraries provide a way for researchers without extensive experience of GPU programming to take advantage of the parallel processing capabilities of modern GPUs.

I also developed GPU-accelerated simulators for each scenario, in the form of JAX-based RL environments, and demonstrated how these can be used to quickly fit the parameters of heuristic policies by simultaneously evaluating many sets of policy parameters on thousands of simulated rollouts in parallel. By reducing the wall time required to run value iteration and simulation optimization, these methods can support research into larger problems, both in terms of scale and the incorporation of aspects of reality that increase the computational complexity. The ability to find optimal policies using value iteration may provide a valuable benchmark for the evaluation of new heuristic and approximate methods, helping efforts to make the best use of scarce resources and reduce wastage.

This chapter is focused on perishable inventory problems with features relevant to the management of blood products but I believe that the methods,

and the underlying principle of using software developed by the machine learning community to parallelize workloads on GPU, may be applicable to many other problems in operational research and have made the code publicly available to support future work.

Chapter 5

Reducing platelet wastage with a machine learning-guided issuing policy

5.1 Motivation

A widespread assumption in the literature is that all platelet units issued to patients are transfused, but my discussions with the transfusion laboratory staff at UCLH, and a review of the data from the laboratory information management system, revealed that it is common for units to be returned to the transfusion laboratory after being issued. This has the potential to lead to wastage when following an OUFO issuing policy, because there may not be sufficient time to reissue returned units before they expire. In this chapter I therefore investigated how hospital blood banks can modify their platelet issuing policies to account for these discrepancies between clinician requests and subsequent transfusions.

It may be possible to predict, based on data about the patient and the request, whether a patient will receive a transfusion when platelets are requested for them. If so, a predictive model could be used to guide a novel issuing policy: issuing the youngest unit for predicted returns and the oldest unit otherwise. In turn, this policy could help to ensure that more returned units are reissued, and ultimately transfused.

Given the challenges involved in accessing healthcare data to start model

development, and the uncertain relationship between good predictive performance and utility in practice, I adopted a simulation-first approach to understand the potential impact of this ML-guided issuing policy on KPIs including wastage and service level, at different levels of simulated predictive performance, before training the predictive model.

5.2 Contribution statement

The main contributions of this chapter are:

- developing a platelet inventory management workflow that includes returns (an important aspect of the real problem at UCLH previously neglected in the literature);
- proposing a novel, ML-guided policy for issuing platelets when returns may occur;
- describing an approach to determine the potential operational utility of predictive ML models in healthcare prior to model development;
- investigating how the impact of the proposed policy on KPIs such as service level and wastage varies with the predictive quality of the model and properties of the workflow; and
- training and temporally validating the first, to the best of my knowledge, supervised learning model to predict platelet returns using both classification metrics and KPIs estimated with the simulated workflow.

5.3 Background and related work

Dumkreiger [63] highlighted the lack of research on issuing policies compared to replenishment policies in the blood product inventory literature. Most studies use an OUFO policy [9, 34, 224] which is intuitively optimal [225] and recommended best practice for blood products [96, 226]. This observation is true for the wider perishable inventory literature, in which most studies use a FIFO issuing policy if,

as in a hospital blood bank, the products are selected by the supplier [6, 29, 227, 228, 229, 230]. More complex policies have been utilized in studies considering a preference for fresher units for a subset of patients [48, 231, 232], the potential for substitution between blood groups if a direct match is not available [36, 47, 55], uncontrolled replenishment [233] and reducing the mean age of transfused blood [234]. Dumkreiger [63] and Abdulwahab and Wahab [70] used approximate dynamic programming to learn custom issuing policies for PRBCs and platelets respectively based on the patient's blood type and current stock holding, but neither considered returns or the likelihood of use. An OUFO policy may not be optimal when some units are returned: an older unit issued to a patient less likely to be transfused may, if returned, expire before it can be used. In preliminary work for this chapter, I found that 8% of platelet units issued in 2015 and 2016 by the hospital blood bank at our partner site, UCLH, were not transfused.

The challenge of platelet returns has not, as far as I am aware, been considered in the blood product inventory management literature and was not discussed in a recent systematic review of efforts to reduce platelet wastage [1]. Older work on hospital blood bank PRBC inventory management included the related concept of assigning units to a sub-inventory for a specific patient following a physical cross-match, with units returned to the main inventory if not used within a specified period [72, 74, 235, 236]. The development of electronic cross-matching and remote issue for PRBCs [237] has reduced the need for assigned sub-inventories (except for patients with special transfusion requirements) and they have not been included in more recent modelling studies [36, 47, 63]. Previous studies have also considered the return of units to a regional blood centre for reissue to a hospital with higher demand [238, 239, 240, 241, 242], but this practice is not permitted in many high-income countries [9] and is not considered here. Product returns have been extensively studied in other contexts [243, 244], but forecasting efforts have focused mainly on aggregate return volumes [245, 246, 247, 248] and recent work predicting returns at the individual level for e-commerce clothing retailers [249, 250, 251, 252] was not used to inform issuing decisions. For non-perishable

products, the main concern would be the costs of processing returns rather than wastage.

I propose a novel, ML-based issuing policy that incorporates the possibility of returns: issue the youngest unit if a model predicts that the request will not result in transfusion, and the oldest unit otherwise. With a sufficiently good predictive model this policy should increase the likelihood that, if a unit is returned, it can be reissued to another patient before it expires. The UK Blood Stocks Management Scheme recommends issuing older stock for patients who are more likely to require a transfusion, and using fresher blood products as stand-by stock for patients who may receive a transfusion [226]. The closest related work I have identified is a working paper by Brodheim and Prastacos [253], who proposed a method to assign older PRBC units to sub-inventories for patients who were most likely to receive a transfusion, based on the quantity requested, at a time when physical cross-matching was required. I focus on platelets, and use an ML model trained on a wide range of features to predict whether requested units will be transfused.

Shah *et al.* [254] observed that conventional model evaluation, using performance metrics such as sensitivity, specificity and the area under the Receiver Operating Characteristic curve (AUROC), measure the quality of an ML model's predictions but provide limited information about the impact of those predictions on patient care and costs. ML models achieving high scores on performance metrics have been found to achieve limited or no clinical benefit in randomized controlled trials [255]. It is therefore critical to consider the clinical workflow in which the ML model would be deployed, including changes to current practice which may be enabled by the predictions [256]. Recent work has addressed this challenge by incorporating trained models into a simulated workflow to estimate the potential impact of the model in terms of KPIs of interest to decision makers [257, 258], but significant work is often required to get access to the data required in order to begin the ML model development process in healthcare [259]. I therefore adopted a simulation-first approach: simulating the predictions of an hypothetical ML model in a simulated workflow to understand the effect of different levels of predictive

model performance on KPIs, and to determine if even a perfect predictive model (PPM) would have a beneficial impact. This could be considered a method to estimate the value of information provided by a prediction with a specified level of quality, by comparing the KPIs of the system with the prediction to those without the prediction [260]. In cases where the workflow is well understood, and where it is possible to develop a simulator based on data that is more readily accessible (for example, aggregated management information), this approach could provide evidence at an early stage about the utility of a model in practice and help prioritize data access requests and model development efforts.

Simulation has been widely used within the blood product inventory management literature to evaluate benefits of changes in workflow or procedures [72, 73, 74, 75, 79, 80, 81], different replenishment policies [44, 45, 46, 49] and, recently, the potential benefits of demand forecasting models using real-time data from EHRs [11, 61, 62, 92, 94] in terms of KPIs *after* models have been trained. Within the broader supply chain literature, researchers have investigated the effect of different levels of error in a demand forecast on KPIs [261, 262, 263] and Meisheri *et al.* [155] investigated the impact of forecast quality on the performance of replenishment policies trained with DRL in a grocery supply chain. Dumkreiger [63] simulated PRBC demand forecasting models with different specified levels of error to determine whether better forecasts could reduce wastage and shortages - the only work I have identified investigating the potential benefits of a predictive model to blood product management prior to model development.

5.4 Simulation-first experiments

5.4.1 Methods

5.4.1.1 Setting

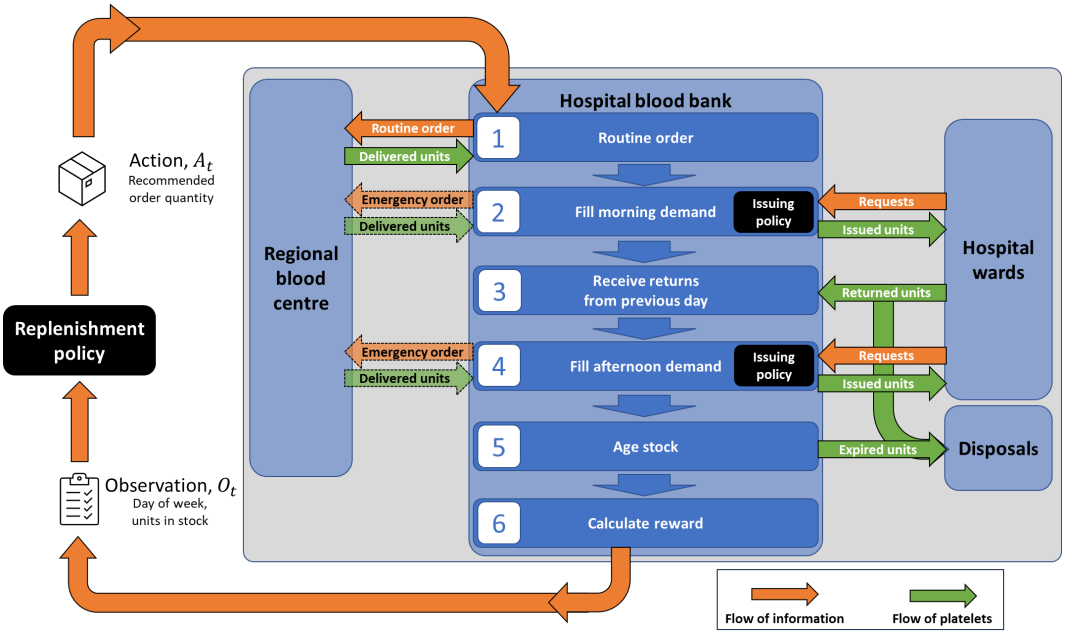
The data used in this chapter comes from our partner hospital trust, UCLH, and its associated transfusion laboratory. I describe the inventory management practices at UCLH in Appendix B.

Data was extracted for the period 2015 to 2017 inclusive, from Bank Manager

(the transfusion laboratory information system) and the UCLH Archive Data Store. See Appendix B for descriptive analysis of the data from Bank Manager. During this period, 1.6% of the platelet units received by UCLH were wasted. This figure is much lower than both the 10-20% reported by Flint *et al.* [1], and the 4.5% reported by the UK Blood Stocks Management Scheme for 2017/2018 [2].

The problem I explore in this chapter, the return of issued platelet units, was inspired by my conversations with staff at UCLH and the transfusion laboratory. I used data from UCLH and the transfusion laboratory to estimate realistic values of key input parameters. As a first effort to incorporate the concept of returns, the simulated workflow for this chapter builds on previous platelet inventory management simulations [45, 166].

5.4.1.2 Simulation



Emergency orders are only placed at stages 2 and 4 if there is insufficient stock to meet demand, and are made for one request at a time.

Figure 5.1: The order of events in one step of the simulated workflow for platelet inventory management including the possibility of returns, corresponding to one day.

I modelled the platelet inventory management task as a periodic review, single echelon perishable inventory problem with a fixed, known delivery lead time $L = 0$ [28]. Demand is stochastic, unmet demand is filled by placing an emergency order

which incurs a penalty, and units in stock with a remaining useful life of one period are assumed to expire at the end of the day. Platelets have a fixed, known useful life m but do not always arrive fresh. The problem includes a reverse flow of stock to represent the return of units that were requested and issued, but not transfused, on the previous day. I included slippage in the system to model potential problems with handling or storage such that not all units returned before expiry can be reissued [264].

The replenishment problem is framed as an MDP (see Section 2.5.1), and implemented as a RL environment using the Python library gymnax [119]. The simulation I developed of the workflow for managing platelets in a hospital blood bank is illustrated at Figure 5.1, showing the six stages that occur each day. A routine order is placed each morning based on the replenishment policy and is assumed to arrive immediately from the regional blood centre. Requests for units from hospital wards are filled from available stock following the issuing policy during the morning and afternoon, and emergency orders are placed in the event of a shortage. Units issued but not transfused on the previous day are returned at midday. Returned units that have expired since issue and units subject to 'slippage' (those not stored or handled appropriately while outside the blood bank) are discarded while other returned units are available to fill afternoon demand. Stock is aged at the end of day and expired units held in stock are discarded. Demand is split into morning and afternoon components because, on a given day, morning demand must be filled before unused units from the previous day are returned to the blood bank and are available for reissue. See Appendix H.1 for the full details of the simulated workflow.

Demand follows a Poisson distribution and I assumed that half of the demand arises in the morning (at stage 2) and half the the demand arises in the afternoon (at stage 4). The four key inputs to the simulation are the mean daily demand $\mu_\tau, \forall \tau \in \{0, 1, ..., 6\}$; the return rate ρ; the slippage rate ϕ; and the parameters for the multinomial distribution used to model the remaining useful life on arrival for each weekday $\underline{\Delta}_\tau, \forall \tau \in \{0, 1, ..., 6\}$. I estimated values for each of these

inputs using data from the UCLH transfusion laboratory information system Bank Manager for the years 2015 and 2016. The maximum useful life on arrival, m, is 5 days, in line with the majority of units received by the UCLH transfusion laboratory during this period. See Appendix H.3 for a detailed description of how these inputs were estimated from the underlying data. I also considered an alternative distribution of remaining useful life on arrival, with $m = 3$, previously reported by Rajendran and Ravindran [54].

The reward function comprises five components: a holding cost per unit in stock at the end of the period ($C_h = 130$), a shortage cost per unit of unmet demand ($C_s = 3,250$), a wastage cost per unit that perishes or is lost to slippage ($C_w = 650$), a variable ordering cost per unit purchased ($C_v = 650$) and a fixed ordering cost which is incurred when $A_t > 0$ ($C_f = 225$). These elements of the reward function are based on those used by Rajendran and Ravindran [54], gathered from a regional medical centre in Pennsylvania, USA, and were also used in Chapter 3. I use a discount factor $\gamma = 1$ in all of the experiments and therefore report the daily cost (the negative mean daily reward) instead of the return G for an episode. With $\gamma = 1$ these measures of performance are directly proportional, and I consider the daily cost easier to interpret.

5.4.1.3 Replenishment policy

The action taken in the MDP is the daily order quantity. In this chapter I do not focus on replenishment policies and therefore consider two straightforward, heuristic replenishment policies: a standing order policy and an $(\mathtt{s}, \mathtt{S})$ policy. The standing order policy is similar to the existing replenishment policy at UCLH (excluding additional orders later in the day): a fixed number of units are ordered each day. In an $(\mathtt{s}, \mathtt{S})$, an order is placed to bring stock up to the order-up-to level, $\mathtt{S}$, if the current stock level is less than or equal to the re-order point $\mathtt{s}$. I found an $(\mathtt{s}, \mathtt{S})$ policy with a pair of parameters for each day of the week to be near optimal in Chapter 4 when considering a platelet replenishment scenario described by Mirjalili [166], in which demand also depended on the day of the week and not all units arrived fresh, and therefore adopted the same policy here. These policies are unlikely to be

optimal because they do not take into account the age profile of the stock [41], or any information about the units issued on the previous day that will be returned, but they are widely used and provide good baselines, for which parameters can be fit in a reasonable amount of time, that enabled me to explore my main interest: the potential of an ML-guided issuing policy.

Following the approach described in Section 4.7, I fit the parameters for the replenishment policies using simulation optimization, using the Python library Optuna to suggest candidate sets of parameters. See Appendix H.2 for additional information about the replenishment policies and how I fit the parameters.

5.4.1.4 Issuing policy

The proposed issuing policy assumes an ML model exists that makes a binary prediction for each request: will the requested platelets be returned to the hospital blood bank, or will they be transfused? If the model predicts that at least one unit in the request will be returned, the freshest unit(s) are issued. Otherwise, the oldest unit(s) are issued. I call this policy Youngest Unit for Predicted Returns (YUPR). At UCLH in 2015 and 2016 over 90% of requests were for a single unit and, therefore, in the simulation-first experiments I make the assumption that the total demand comprises only requests for a single unit. In this case the freshest units are issued if the ML model predicts that the unit will be returned and the oldest units are issued otherwise.

The effectiveness of this approach depends on the quality of the predictive model. Two metrics for assessing the performance of a binary classification model are the sensitivity (the proportion of requests where the platelets were returned that the model predicted would be returned) and the specificity (the proportion of requests where the platelets were transfused that the model predicted would be transfused). If the model had a sensitivity = 0.0 and a specificity = 1.0, it would predict that all requested units would be transfused and the YUPR policy would be equivalent to an unmodified OUFO policy. Alternatively, if the model had a sensitivity = 1.0 and a specificity = 0.0, it would predict that all requested units would be returned and the YUPR policy would be equivalent to an unmodified

Youngest Unit First-out (YUFO) policy.

In a conventional ML workflow, these metrics would be calculated on a portion of the dataset held-out from training, with known outcomes, to estimate the generalization performance of the model. Here, for the simulation-first approach, it is necessary to simulate both the true labels (i.e. will the requested unit be returned or transfused?) and the predicted label. The true label for each request is sampled from a binomial distribution with a probability of success equal to the return rate ρ. To generate the predicted label, I specified levels of sensitivity (α) and specificity (β) and simulated the predictions that would be made by a model with that level of performance. If the sampled true label was 1 (and therefore the unit would not be transfused), the predicted label was set to 1 if a sample drawn from a uniform distribution between 0 and 1 was less than α and 0 otherwise. If the sampled true label was 0, the predicted label was set to 1 if a sample from a uniform distribution between 0 and 1 was greater than β, and 0 otherwise. This process is described in pseudocode in Algorithm 5 in Appendix H. There is a temporal order to requests within a period, and the issuing policy was applied to one request at a time in this order.

5.4.1.5 Simulation experiments

I summarize the input settings for simulation experiments 1–7 in Table 5.1. Experiments 1 and 2 represent a setting with returns, using a standing order and $(\mathtt{s}, \mathtt{S})$ replenishment policy respectively. I used return rate $\rho = 8\%$ and slippage rate $\phi = 7\%$, estimated based on data from UCLH as described in Appendix H.3 and the parameters for the distribution of remaining useful life on arrival estimated from UCLH data set out in Table H.3 in Appendix H. The mean daily demand was estimated from UCLH data including returns (UCLH Tx+r), as set out in Table H.1 in Appendix H.

I investigated the potential benefit of my proposed ML-guided issuing policy, YUPR, to understand whether it could be beneficial and, if so, how good the model would need to be to improve KPIs. In each experiment, I considered all pairs of predictive model sensitivity and specificity between 0 and 1 in increments of 0.1.

Exp	Distribution of remaining useful life on arrival	ρ	ϕ	Demand distribution	Replenishment policy	Issuing policy
1	UCLH	8%	7%	UCLH Tx+r	Standing order	YUPR
2	UCLH	8%	7%	UCLH Tx+r	`(s,S)`	YUPR
3	R&R [54]	8%	7%	UCLH Tx+r	Standing order	YUPR
4	R&R [54]	8%	7%	UCLH Tx+r	`(s,S)`	YUPR
5	UCLH	⧫	7%	◊	`(s,S)`	OUFO, YUPR-PPM
6	UCLH	8%	⧫	UCLH Tx+r	`(s,S)`	OUFO, YUPR-PPM
7	⧫	8%	7%	UCLH Tx+r	`(s,S)`	OUFO, YUPR-PPM

The UCLH Tx+r demand distribution is based on units requested at at UCLH including those ultimately returned (see Table H.1 in Appendix H). The issuing policy YUPR denotes experiments that tested different combinations of predictive model sensitivity and specificity, including a setting equivalent to a standard OUFO issuing policy and a setting with a PPM. YUPR-PPM is a YUPR issuing policy with a PPM. A black lozenge ⧫ denotes an input that was varied during an experiment, and a white lozenge ◊ indicates an input that was varied during an experiment as a consequence of changing another input.

Table 5.1: Summary of the input settings for experiments 1–7 using the simulated platelet inventory management workflow with returns.

This range includes a policy that is equivalent to an OUFO policy (when sensitivity = 0.0 and specificity = 1.0) and a PPM (when sensitivity = 1.0 and specificity = 1.0). Comparing the results for a PPM to a baseline OUFO policy tells us whether the proposed policy would outperform OUFO and the scale of the benefits on offer, while comparing intermediate values to the OUFO policy identifies a region of combinations of predictive model sensitivity and specificity where the YUPR issuing policy could provide a benefit.

The benefit of the YUPR issuing policy may depend on inputs to the simulated workflow. I have not identified any previously published estimates of the return rate or slippage rate, but Rajendran and Ravindran [54] reported an alternative distribution of remaining useful life on arrival from a regional medical centre in Pennsylvania, USA (R&R). The maximum useful life is three days, 50% of the stock arrives fresh, 20% with two days of useful life, and 30% on the day that it will expire. The reason for this difference is that bacterial screening procedures adopted in the UK and elsewhere in Europe enable a shelf life of up to 7 days after donation, while until recently the US Food and Drug Administration limited storage to 5 days after donation [17, 18]. I repeated experiments 1 and 2 , using the same settings except for this alternative distribution of remaining useful life on arrival, and report

the results as experiments 3 and 4.

To evaluate the impact of the return rate and the slippage rate, and a wider range of distributions of remaining useful life on arrival, in each of experiments 5, 6 and 7 I varied one input parameter. The other settings remained the same as in experiment 2. I recorded the mean daily cost, wastage and service level when using an OUFO issuing policy and when using the YUPR issuing policy with a PPM.

In experiment 5, I changed the return rate ρ between 0.0 and 0.5 in increments of 0.05. I adjusted the mean daily demand as I adjusted the return rate so that the expected number of transfused units remained the same. The mean daily demand for each experiment is therefore equal to the mean daily demand for transfused units only reported in Table H.2 of Appendix H (UCLH Tx) divided by by $(1 - \rho)$.

In experiment 6, I changed the slippage rate ϕ between 0.0 and 1.0 in increments of 0.1.

In experiment 7, I changed the parameters of the multinomial distribution used to model the remaining useful life on arrival. I used the same distribution for each day of week, with the parameters set equal to the values of a binomial distribution with 4 trials and a probability of success between 0.0 and 1.0 in increments of 0.1. I added one to the value sampled from the binomial distribution, so that when the probability of success is 1.0, all units arrive with five days of remaining useful life, and when the probability of success is 0.0 all units arrive on the day that they expire. The distributions for each value of the probability of success are set out in Table H.5 in Appendix H.

In each of experiments 1-7, and for each issuing policy in experiments where the sensitivity and specificity of the predictive model were changed, I evaluated the combination of issuing policy and replenishment policy with the best parameters identified from simulation optimization on 10,000 evaluation episodes, each 365 days long with a warm-up period of 100 days. I report the mean daily cost, mean wastage and mean service level over these 10,000 episodes. For each episode, the wastage was calculated as proportion of units received (from both routine and emergency orders) that were wasted (due to either expiry or slippage) over the

365 days following the warm-up period. The service level was calculated as the proportion of the total demand filled by units in stock (rather than requiring an emergency order) over the 365 days following the warm-up period. I also calculated the standard deviation of these metrics over the 10,000 evaluation episodes which reflects the stochasticity in each scenario (due to random demand, remaining useful life, slippage, and whether a requested unit is transfused or returned).

When the only difference between two experiments is the policies, it is possible to directly compare their performance on each evaluation episode because I used random seeds to generate the random elements of the simulation in a reproducible manner. For pairs of replenishment and issuing policy combinations in experiments 1 and 2 (and, separately, experiments 3 and 4), I calculated the difference in daily cost, service level and wastage achieved by the policies on each evaluation episode, and then computed the mean of the differences and the standard error of the mean of the differences. These measures provide as estimate of the possible improvement in a metric by changing the replenishment policy alone, the issuing policy alone, or both policies.

5.4.1.6 Hardware

The simulation experiments were conducted on a desktop computer running Ubuntu 20.04 LTS via Windows Subsystem for Linux on Windows 11 with an AMD Ryzen 9 5900X processor, 64GB RAM, and an Nvidia GeForce RTX 3060 GPU.

5.4.1.7 Code availability

The code supporting the work in this chapter, for both the simulation experiments and model training described in Section 5.5, is available at: https://github.com/joefarrington/plt_returns

5.4.2 Results

5.4.2.1 Main results

Table 5.2 summarizes the results of experiments 1–4, in terms of the mean daily cost, service level and wastage over 10,000 simulated years: the evaluation episodes. The costs reflect the balancing of different priorities of the problem and

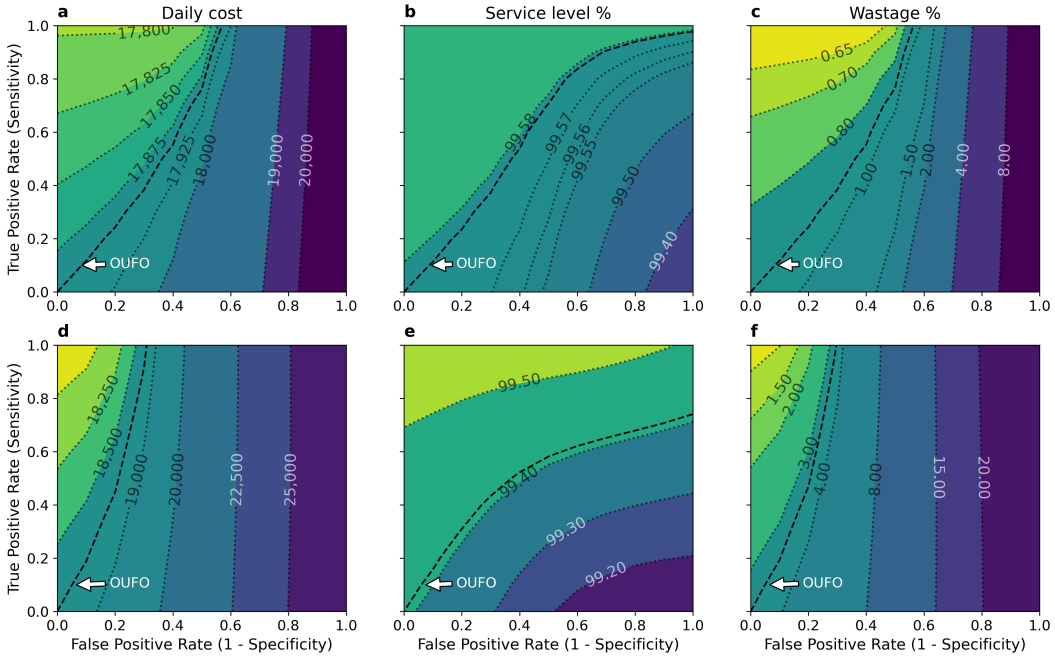
do not directly reflect monetary costs. I therefore focus the discussion of the results on the KPIs. A lower bound on the costs for experiments 1–4 is the expected daily cost for replenishment (16,066) based on the mean demand for units that will be transfused and assuming an order is placed every day.

I estimated the KPIs for different levels of predictive model quality defined by sensitivity and specificity. Tables 5.2 and 5.3 present the results at two key settings: a standard OUFO issuing policy (equivalent to a YUPR policy with sensitivity = 0.0 and specificity = 1.0) , and a YUPR policy with a perfect predictive model (YUPR-PPM; sensitivity = specificity = 1.0). Table 5.3 sets out differences in performance for combinations of replenishment and issuing policy over the 10,000 simulated years used for evaluation.

In the paired-sample comparisons between OUFO and YUPR-PPM issuing policies, the mean reduction in cost, increase in service level, and reduction in wastage were not less than three standard errors of the mean across both replenishment policies and both distributions of remaining useful life on arrival. In no fewer than 99.5% of episodes, the YUPR-PPM issuing policy resulted in less wastage than the OUFO issuing policy under the same replenishment policy and distribution of remaining useful life. There is therefore a clear, and significant, advantage to using YUPR with a perfect predictive model over an OUFO issuing policy when units may be returned.

There is also a clear benefit to improving jointly both the replenishment policy and the issuing policy: the best results are achieved when using an `(s,S)` replenishment policy and the YUPR issuing policy with a PPM. In experiments 2 and 4 wastage due to time expiry is significantly reduced, with the remaining wastage due almost entirely to slippage. The KPIs for this pair of experiments demonstrates the particular effectiveness of the proposed policy at mitigating the negative effects of receiving stock with a shorter average remaining useful life on arrival.

The full results from experiments 2 and 4, for different levels of predictive model performance, are presented above as contour plots in Figure 5.2. While



The daily cost (**a**), service level (**b**) and wastage (**c**) assuming the distribution of remaining useful life on arrival observed at UCLH, and the corresponding metrics (**d,e,f**) when assuming the distribution of remaining useful life on arrival reported by Rajendran and Ravindran [54]. Subplots **(a,b,c)** are based on results from experiment 2 and subplots **(d,e,f)** are based on results from experiment 4. These plots show that under both settings the proposed approach can reduce wastage and cost, with no reduction in service level, relative to an OUFO issuing policy. Lighter colours indicate better performance. A perfect predictive model, with sensitivity and specificity both equal to 1.0, would be in the top left corner of a subplot. The region above and to the left of the contour for an OUFO issuing policy comprises combinations of sensitivity and specificity required for the YUPR issuing policy to perform better than OUFO. Each plot contains a labelled contour showing the performance for a baseline OUFO issuing policy and the colour map for each plot is centred on that contour.

Figure 5.2: Contour plots illustrating the performance of the proposed YUPR issuing policy in terms of daily cost, service level and wastage assuming different levels of predictive model quality.

the YUPR issuing policy could cut wastage, with no reduction in service level, a predictive model with a high false positive rate would lead to higher wastage and costs than the baseline OUFO policy.

In Appendix H.4, I present the results of using the simulated workflow and scenario inputs under the assumption made in previous work: that all requested units are transfused. Wastage was low even with a simple standing order policy and was reduced to 0% for both distributions of remaining useful life on arrival under an `(s,S)` replenishment policy.

5.4.2.2 Results for additional simulation sensitivity analyses

In Figures 5.3 and 5.4 I present the results from the sensitivity analyses: experiments 5, 6 and 7. Figure 5.3 illustrates the mean values of the daily cost, service level

Distribution of remaining useful life on arrival	Exp	Replenishment policy	Issuing policy	Daily cost		Service level (%)		Wastage (%)	
				Mean	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*
UCLH	1	Standing order	OUFO	20,344	(500)	97.5	(0.8)	1.0	(0.2)
		Standing order	YUPR-PPM	20,337	(533)	97.8	(0.9)	0.7	(0.2)
	2	$(\mathtt{s}, \mathtt{S})$	OUFO	17,891	(199)	99.6	(0.1)	0.9	(0.1)
		$(\mathtt{s}, \mathtt{S})$	YUPR-PPM	17,797	(197)	99.6	(0.1)	0.6	(0.1)
R&R	3	Standing order	OUFO	20,992	(425)	97.5	(0.6)	4.8	(0.5)
		Standing order	YUPR-PPM	20,145	(301)	98.8	(0.4)	3.5	(0.7)
	4	$(\mathtt{s}, \mathtt{S})$	OUFO	18,719	(215)	99.4	(0.2)	3.4	(0.2)
		$(\mathtt{s}, \mathtt{S})$	YUPR-PPM	17,829	(200)	99.5	(0.1)	0.7	(0.1)

The daily cost and KPIs were calculated for each episode and the mean and standard deviation over the 10,000 evaluation episodes is reported.

Table 5.2: Simulation results for experiments 1–4 using the simulated platelet inventory management workflow with returns.

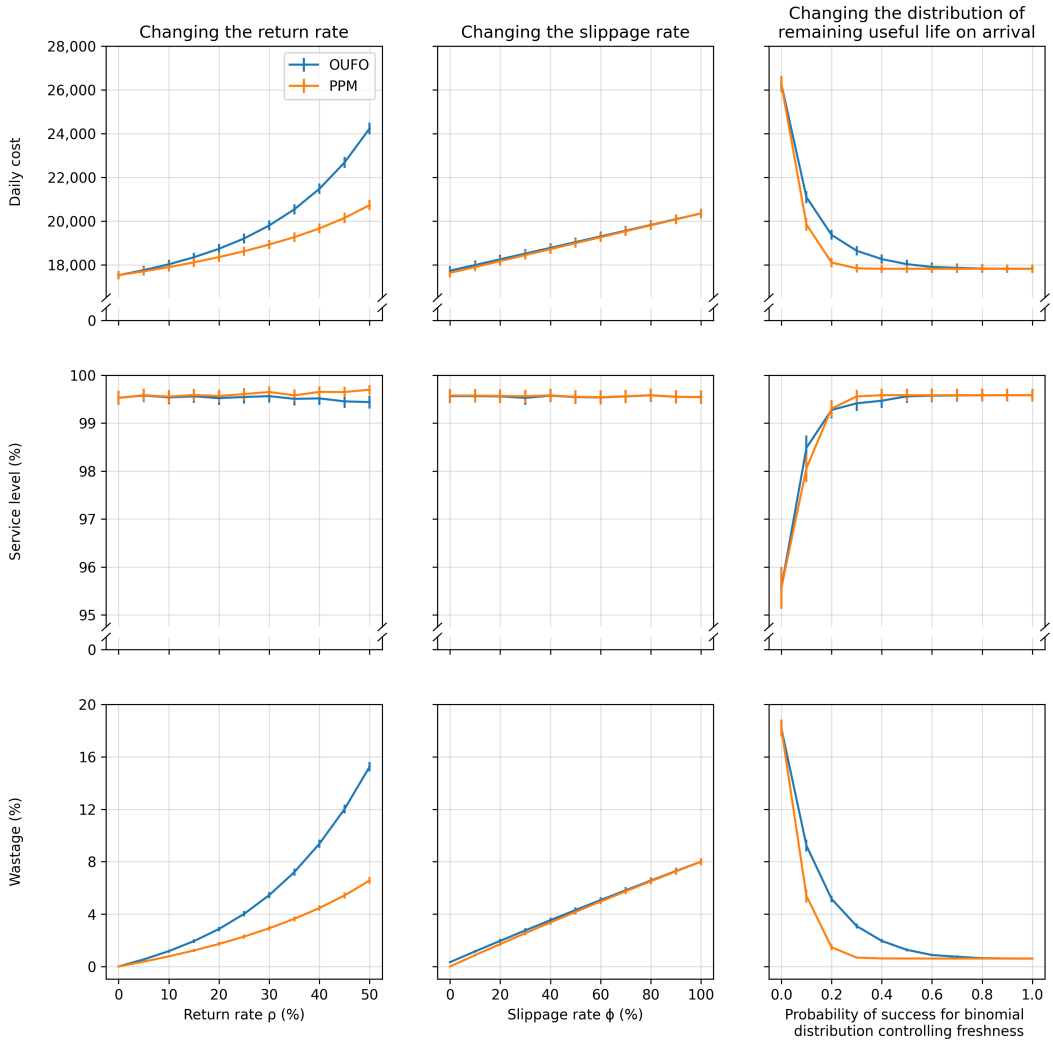
and wastage over the evaluation episodes for an OUFO issuing policy and the YUPR issuing policy with a PPM. Figure 5.4 illustrates the mean paired-sampled differences in these metrics between the two issuing policies over the evaluation episodes.

The YUPR issuing policy with a PPM had an increased beneficial impact as the return rate increased, as the slippage rate decreased, and as the average age of stock at arrival increased. As expected, the YUPR policy with PPM performed the same as OUFO in cases when all requests result in transfusion (return rate $\rho = 0\%$) or when units that are not transfused cannot be reissued (due to a high slippage rate, or because all of the stock expired on the day it arrived). The YUPR policy with a PPM also performed similarly to OUFO when most of the stock arrived fresh, because in this instance there is often sufficient time for a returned unit to be reissued even under an OUFO policy. The results show the importance of achieving a low slippage rate: at a slippage rate $\phi = 100\%$ the wastage under both issuing policies is 8% (equal to the return rate), but at a slippage rate of $\phi = 0\%$ this is reduced to 0.3% under an OUFO issuing policy and 0.0% under the YUPR issuing policy with a PPM.

Metric	Replenishment policy	Issuing policy	Standing order		(s,S)	
			OUFO	YUPR-PPM	OUFO	YUPR-PPM
Experiments 1 & 2: UCLH distribution of remaining useful life on arrival						
Daily cost	Standing order	OUFO	—	-7 ± 2	-2,452 ± 4	-2,547 ± 4
		YUPR-PPM	—	—	-2,446 ± 5	-2,540 ± 5
	(s,S)	OUFO	—	—	—	-94 ± 0
		YUPR-PPM	—	—	—	—
Service level (%)	Standing order	OUFO	—	0.3 ± 0.0	2.1 ± 0.0	2.1 ± 0.0
		YUPR-PPM	—	—	1.8 ± 0.0	1.8 ± 0.0
	(s,S)	OUFO	—	—	—	0.0 ± 0.0
		YUPR-PPM	—	—	—	—
Wastage (%)	Standing order	OUFO	—	-0.3 ± 0.0	-0.1 ± 0.0	-0.4 ± 0.0
		YUPR-PPM	—	—	0.2 ± 0.0	-0.1 ± 0.0
	(s,S)	OUFO	—	—	—	-0.3 ± 0.0
		YUPR-PPM	—	—	—	—
Experiments 3 & 4: R&R distribution of remaining useful life on arrival						
Daily cost	Standing order	OUFO	—	-847 ± 3	-2,274 ± 3	-3,164 ± 3
		YUPR-PPM	—	—	-1,426 ± 3	-2,316 ± 3
	(s,S)	OUFO	—	—	—	-890 ± 1
		YUPR-PPM	—	—	—	—
Service level (%)	Standing order	OUFO	—	1.3 ± 0.0	1.9 ± 0.0	2.0 ± 0.0
		YUPR-PPM	—	—	0.6 ± 0.0	0.7 ± 0.0
	(s,S)	OUFO	—	—	—	0.1 ± 0.0
		YUPR-PPM	—	—	—	—
Wastage (%)	Standing order	OUFO	—	-1.3 ± 0.0	-1.4 ± 0.0	-4.1 ± 0.0
		YUPR-PPM	—	—	-0.2 ± 0.0	-2.8 ± 0.0
	(s,S)	OUFO	—	—	—	-2.7 ± 0.0
		YUPR-PPM	—	—	—	—

The daily cost and KPIs were calculated for each episode. The mean and standard error of the mean paired-sample differences between the policy combination on the column and the policy combination on the row for these metrics over the 10,000 evaluation episodes is reported. Negative values for daily cost and wastage, and positive values for service level, indicate the the policy combination on the column performed better than the policy combination on the row.

Table 5.3: Paired-sample comparison of combinations of replenishment and issuing policies from experiments 1 & 2 and experiments 3 & 4 using the simulated platelet inventory management workflow with returns.



The daily cost and KPIs were calculated for each episode. The mean and standard deviation (as error bars) of each metric over the 10,000 evaluation episodes is plotted.

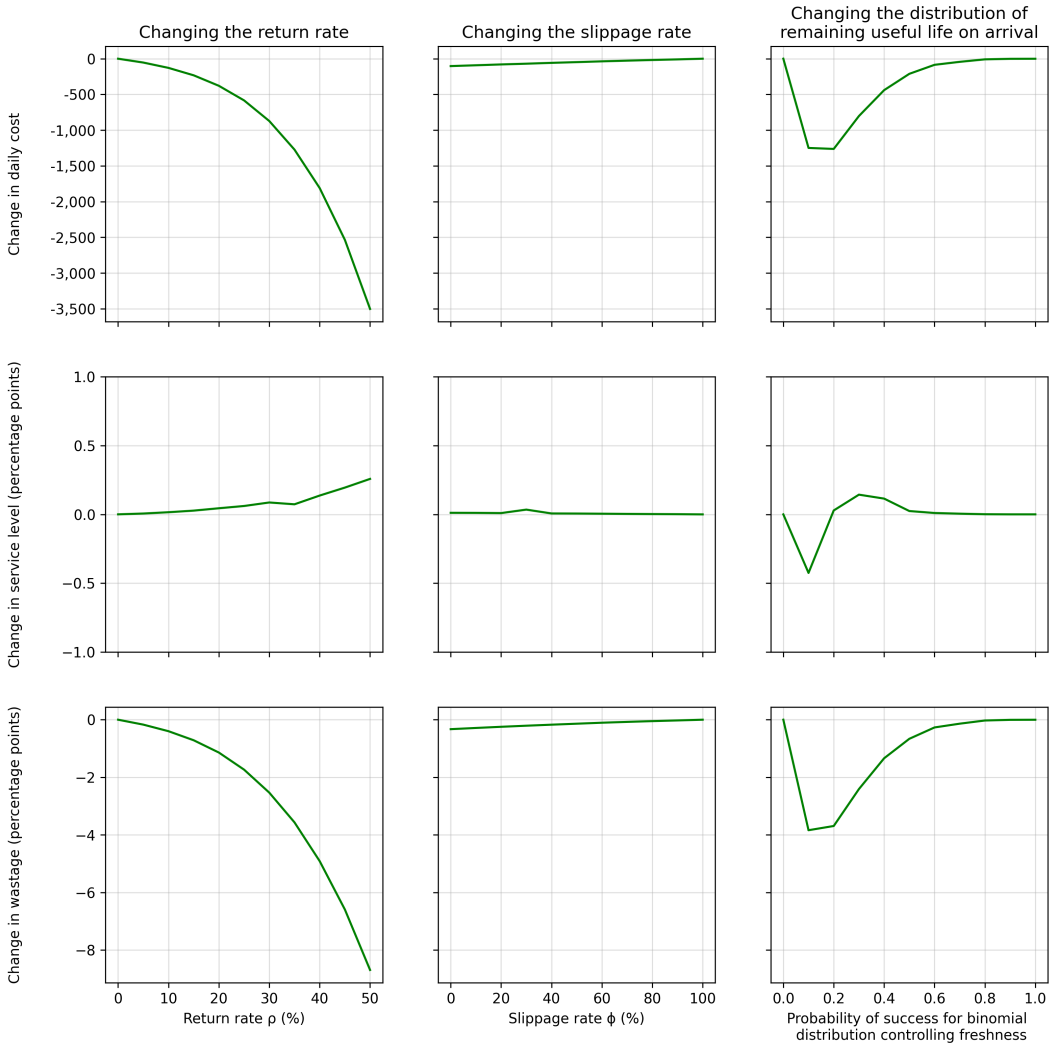
Figure 5.3: Impact of changing simulation input parameters on the the daily cost, service level and wastage when using the benchmark OUFO issuing policy and the proposed YUPR issuing policy with a PPM.

5.5 Building a predictive model

5.5.1 Methods

5.5.1.1 Data

I developed a binary classification model to predict, for each request, whether at least one requested unit would be returned. Demographic features (sex, year of birth) and platelet count test results were extracted from the UCLH Archive Data Store. Data related to requests for platelets, and individual platelet units,



The daily cost and KPIs were calculated for each episode. The mean and standard error of the mean (as the error bars) paired-sample differences between the the benchmark OUFO issuing policy and the proposed YUPR issuing policy with a PPM over the 10,000 evaluation episodes is plotted. Reductions in daily cost and wastage, and increases in service level, indicate cases where the YUPR issuing policy with a PPM performs better than an OUFO issuing policy. The error bars are not visible at the scale selected to show the change in cost, level and wastage respectively.

Figure 5.4: Paired-sample differences in daily cost, service level and wastage between the benchmark OUFO issuing policy and the proposed YUPR issuing policy with a PPM.

was extracted from the UCLH transfusion laboratory information system Bank Manager. Missing values for numeric values were imputed using the median and missing values for binary features were imputed using the mode. Binary missing indicator features were added corresponding to each numeric or binary feature. Categorical features were one-hot encoded, and the number of one-hot features for high-cardinality features (defined as 16 or more categories) was determined as part

of the hyperparameter tuning process.

The training set consisted of 17,297 requests with a required date between 1 February 2015 and 31 December 2016, while the test set consisted of 9,353 requests with a required date between 1 January 2017 and 31 December 2017. I excluded requests where no unit was assigned, where a neonatal unit was assigned, where the patient identifier was that of a test patient used for internal system checks and where the request was required more than 30 minutes before it was registered. Some calculated features are based on a look-back period of 30 days and therefore I also excluded the requests from January 2015 to ensure a full look-back period for all requests in the training and test sets. The time point of the prediction, used to determine the most recent platelet count and what other information was available when the prediction was made, was set as the later of one hour before the request was required, and the time when the request was registered in Bank Manager. See Table I.1 in Appendix I for a breakdown of exclusions, Table I.2 in Appendix I for a description of all of the input features and Tables I.3 and I.4 in Appendix I for details of the numeric and categorical features respectively including the percentage of missing entries, the mean and standard deviation of numeric features, and the most common categories for categorical features.

5.5.1.2 Training and evaluation

I used XGBoost [165], a popular and fast implementation of gradient boosting with decision trees which is known to perform well on a variety of prediction tasks based on tabular data [265]. The hyperparameters were tuned using 10-fold cross-validation over the training set using Optuna's [212] Tree-structured Parzen Estimator (TPE) sampler, with 200 trials. The folds were stratified so that each fold received approximately the same ratio of positive to negative examples, and each patient was assigned to only one fold. Based on the results of the simulation experiments, it was clear that the false positive rate of the model needed to be less than 0.6 for an issuing policy supported by the model to achieve lower daily cost and wastage than an OUFO issuing policy. I therefore used the partial-AUROC [266] to compare model performance during hyperparameter tuning, considering

only the area under the curve where the false positive rate was less than 0.6. An improvement in this metric should lead to an improvement in the KPIs whereas an improvement in standard AUROC could be due to better predictive performance in a region of the of the curve that the simulation results suggest will not lead to better inventory management performance. The hyperparameter search ranges, and final selected hyperparameters are set out in Table I.5 in Appendix I.

The final model was trained on the entire training set using the combination of hyperparameters that achieved the highest mean partial-AUROC over the 10 cross-validation folds, and applied to predict outcomes for the test set. Performance on the test set was reported in terms of AUROC because I consider this metric both more familiar and easier to interpret than partial-AUROC.

One approach to estimating the potential benefit of the trained model in terms of KPIs is to determine the sensitivity and specificity and look up the KPIs for simulated predictive models with those characteristics using the results of experiments 2 and 4. The sensitivity and specificity depend on the threshold used to determine a positive prediction, and the Receiver Operating Characteristic (ROC) curve represents the different possible combinations of sensitivity and specificity as the threshold is increased.

The KPIs observed at a certain level of sensitivity and specificity assumed in experiments 2 and 4 may not correspond to those that would be achieved at the same level of sensitivity and specificity for the trained model in reality because I made the simplifying assumption that the total demand was generated by requests for single units in the simulation experiments. In reality, during 2015 and 2016, 7% of requests were for more than one unit, the majority of these were for two units and the largest request was for four units. When I trained the ML model on real requests I assigned labels based on whether or not all of the requested units were transfused. As a result, there is a discrepancy between sensitivity and specificity as defined for the simulation and for the real predictive model: for example, a correct positive prediction on a real request for two units could mean that one is transfused and one is returned. I addressed this by performing an additional evaluation step:

applying the predictive model to real demand data from 2017 within the simulated workflow, making predictions at the request level (so, for example, a request for two units with a positive prediction would be met by issuing the two freshest units), and calculating the KPIs directly.

I created a subclass of the RL environment that used the real observed demand and returns from 2017 (instead of sampling them), and the predictions from the trained model to support the issuing decision. Morning demand consisted of requests that were required after 00:00 and before 12:00, and afternoon demand consisted of requests required after 12:00 and before 00:00. The slippage rate, ϕ, was set to 7%. The distribution of remaining useful life on arrival was calculated based on units received by UCLH in 2017, as set out in Table H.4 in Appendix H. The threshold for a positive prediction from the model used to implement the policy was set to minimize the estimated wastage, based on the results of experiment 2. For each potential model threshold in the training set, I estimated the expected wastage by linearly interpolating between the values of wastage for fixed combinations of sensitivity and specificity from experiment 2 and selected the threshold with the minimum estimated wastage. I selected a threshold to minimize wastage directly because the contour plots of service level produced for experiment 2 showed that the sensitivity and specificity of the predictive model had little to no impact on service level. Figure I.1 in Appendix I shows the ROC curve for the training set plotted over a contour plot of wastage from experiment 2, and estimated wastage plotted directly against the model threshold. I repeated the experiment with an OUFO issuing policy as a baseline and used `(s,S)` replenishment policies with parameters fit as part of experiment 2.

I repeated this analysis using the distribution of remaining useful life on arrival reported by Rajendran and Ravindran [54]. I used the same trained predictive model, but selected the threshold using the values of wastage from experiment 4. Figure I.2 in Appendix I shows the ROC curve for the training set plotted over a contour plot of wastage from experiment 4, and estimated wastage plotted directly against the model threshold. I used `(s,S)` replenishment policies with parameters

fit as part of experiment 4. All other settings remained the same.

5.5.1.3 Hardware

Training and evaluating the predictive model required patient level data and therefore these experiments were conducted on a virtual machine using the UCLH Data Science Desktop, a secure research environment.

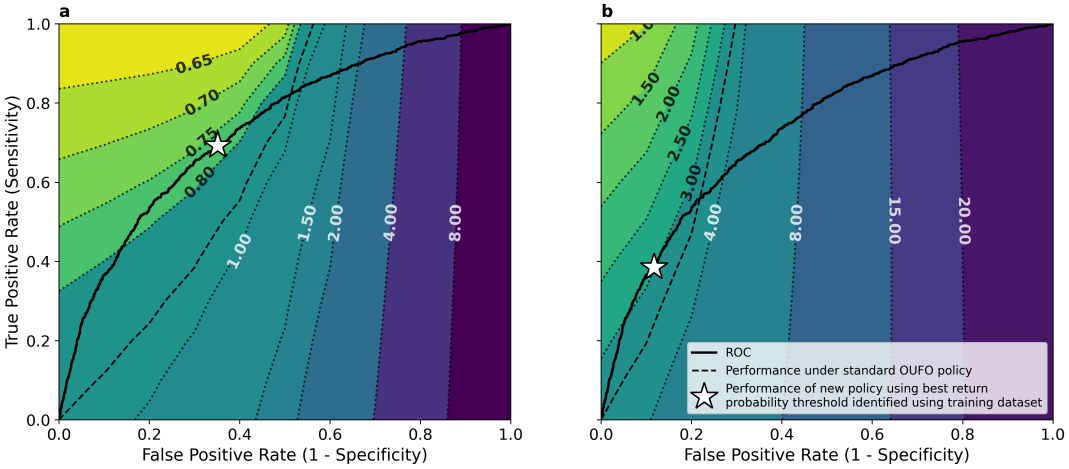
5.5.2 Results

Simulation-first experiments demonstrated that the YUPR issuing policy could cut wastage relative to an OUFO issuing policy with no detriment to service level, given a sufficiently good ML model. Figure 5.5 shows that YUPR with the trained predictive model is sufficiently good to reduce wastage relative to an OUFO policy in both contexts.

In Figure 5.5 I present a contour plot of simulated wastage as a function of model prediction performance (in terms of sensitivity and specificity) for two scenarios, overlaid with the ROC curve calculated on the test set for the predictive model I went on to develop. In Figure 5.5a, the remaining useful life of platelet units on arrival at the hospital is assumed to be distributed according to observed data from UCLH. In this context the YUPR policy with a PPM, with sensitivity and specificity both equal to 1.0, would cut wastage from 0.9% to 0.6% (a 33% reduction) compared to the standard OUFO policy. For the results shown in Figure 5.5b, the remaining useful life of platelets on arrival is based on figures reported by Rajendran and Ravindran [54] from a hospital in the USA where the stock has a shorter maximum useful life and arrives with fewer days of remaining useful life on average. Here the YUPR policy with a perfect predictive model would cut wastage from 3.4% to 0.7% (a reduction of 79%).

The ML model achieved an AUROC of 0.74 on the test set: platelet requests at UCLH from 2017. At a threshold identified from the training set, a YUPR issuing policy based on the model would reduce wastage by 14% at UCLH and 12% when using the distribution of remaining useful life reported by Rajendran and Ravindran [54]. Figure 5.2 illustrates that these reductions in wastage can be achieved while

maintaining the service level, defined as the proportion of requests that can be met using stock on hand.



The ROC curve on the test set plotted over the contour plot for wastage generated assuming **(a)** the distribution of remaining useful life on arrival observed at UCLH and **(b)** the distribution of remaining useful life on arrival reported by Rajendran and Ravindran [54]. Subplot **(a)** is based on results from experiment 2 and subplot **(b)** is based on results from experiment 4. Under both settings, the predictive model could achieve lower wastage than a baseline OUFO policy with no reduction in service level. A larger absolute reduction in wastage is possible in **(b)**, when platelets have a shorter average remaining useful life on arrival. Lighter colours indicate better performance. The ROC curve represents possible combinations of sensitivity and sensitivity that could be achieved by the selection of different thresholds for distinguishing positive and negative predictions, and the white star indicates the predictive performance that would be achieved by selecting a threshold to minimize wastage using the training set ROC curve.

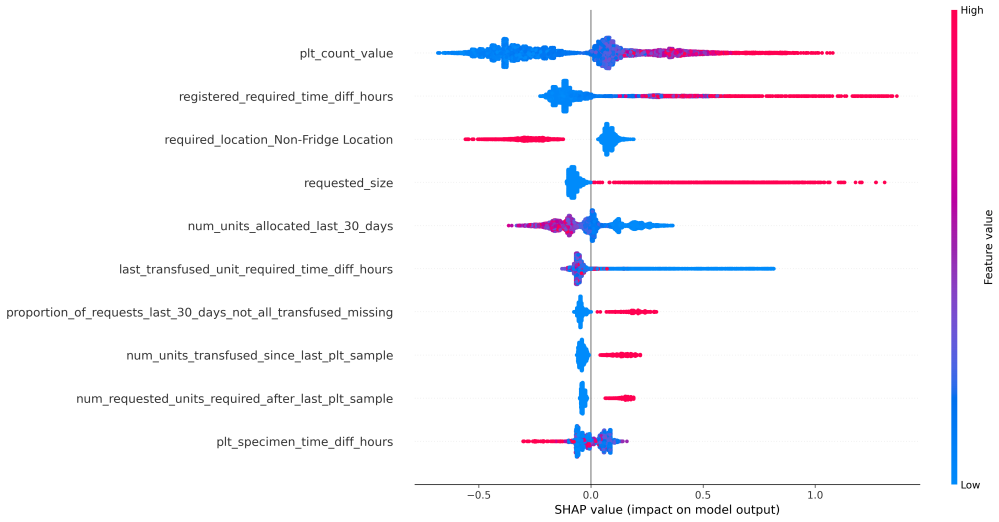
Figure 5.5: Contour plots illustrating estimated wastage when issuing platelets using the proposed YUPR issuing policy and the trained predictive model for platelet returns.

An alternative method for evaluating the performance of the predictive model is to incorporate the real demand and associated predictions into a simulated workflow. Using real requests for 2017 and the predictions of the trained models on these to inform YUPR issuing decisions, the simulated workflow gives estimated wastage of 1.1% and a service level of 99.4% (compared to 1.2% and 99.4% respectively using an OUFO issuing policy). Under the distribution of remaining useful life on arrival reported by Rajendran and Ravindran [54] our YUPR issuing policy with a trained predictive model gave wastage of 4.3% and a service level of 99.1% (compared to 5.0% and 99.1% respectively using an OUFO issuing policy). In both cases, wastage was reduced compared to the OUFO baseline with no reduction in service level. Although the wastage is higher than expected from the simulation results plotted in Figure 5.5, this is consistent with the higher proportion of requests that did not result in transfusion, and higher wastage, observed at UCLH in 2017 compared

to the preceding years.

The performance of the trained ML model on the test set is illustrated by the solid black line in Figure 5.5. Based on the results of the simulation experiments, further improvements in the quality of the predictive model would be expected to lead to further reductions in wastage. The results of the scenario analysis presented in Section 5.4.2.2 show that the potential benefits will depend on properties of the workflow in addition to model quality.

I computed feature importance values using Shapley Additive Explanations (SHAP) [267] and present a summary plot, illustrating how the features contribute to the predictions, in Figure 5.6. Unsurprisingly, the value of the most recent platelet count was the most important feature, and a lower platelet count made the model more likely to predict that requested units would be transfused. Seven of the top 10 features were hand-engineered features, including the second most important feature: how long in advance (in hours) a request was made.



The feature importance values were computed using the training set. Each point represents an example from the training set. A large positive SHAP value means that the feature pushed the model output towards a positive prediction. See Table I.2 in Appendix I for a description of each feature.

Figure 5.6: Summary plot of SHAP values for the 10 most important features in the trained predictive model for platelet returns.

5.6 Discussion

To the best of my knowledge this work is the first to consider the fact of post-issue returns when finding replenishment and issuing policies to manage platelet inventory, to develop an ML-based prediction of platelet usage and to develop an accompanying issuing policy for a perishable product. The results show that an ML-based issuing policy could reduce wastage when it is possible to reissue returned units. While the projected benefit of a 14% reduction in wastage for our partner hospital might be considered modest in the context of the low wastage they currently achieve, the sensitivity analyses highlight that when units have a shorter remaining useful life on arrival or when the return rate is higher, this approach can lead to large absolute reductions in wastage. The YUPR issuing policy with a PPM gave a reduction in wastage of 2.7 percentage points (79%) compared to the OUFO baseline (under an `(s,S)` replenishment policy) for the distribution of remaining useful life on arrival observed in a US hospital [54]. The performance with an `(s,S)` replenishment policy and YUPR issuing policy was similar under both distributions of remaining useful life, despite the much shorter useful life on arrival in the US case. The UCLH transfusion laboratory is considering moving to a YUPR policy, although they are exploring a rule based prediction model based on augmenting the insights from my ML model with expert knowledge rather than investing at this point in integrating different software systems to support real-time ML model predictions.

My results also demonstrate the importance of considering returns when evaluating policies for managing platelet inventory. Developing policies based on the number of transfused units could lead to shortages if not all requested units are transfused, because in this case the number of transfused units underestimates the demand that must be filled by the hospital blood bank. The order-up-to parameter `S` was higher for four of seven weekdays when introducing returns (comparing experiments 2 and 4). Consideration of returns, and my proposed issuing policy, may be less impactful at sites where issued units that are not transfused spend a shorter time away from the decision-making point. It may have a greater impact as

improvements in technology or procedures reduce the level of slippage.

This study also provides a compelling example of using a simulation-first approach to evaluate the extent to which an ML model can support a clinical or operational workflow. The simulated workflow was used in three ways: firstly to estimate the potential benefits of a proposed predictive model in the workflow, secondly to inform model selection during hyperparameter search and thirdly to estimate the performance of a trained model on a real time period using established KPIs. The results from the simulation informed the subsequent steps of my research by establishing both the potential gains on offer and the performance required for the proof-of-concept ML model to be beneficial. The results highlight the importance of investigating how a predictive model performs in a workflow in terms of KPIs as well as performing evaluation using ML metrics. The same predictive model, with the same AUROC, can have a bigger impact on the workflow under some circumstances, such as under the distribution of remaining useful life on arrival observed by Rajendran and Ravindran [54], than others. This chapter builds on recent work using simulation to evaluate trained ML models in terms of KPIs that have real-world impact rather than ML metrics [257, 258] but with the added potential to understand whether the predictions could be useful *before* building the predictive model and to estimate how well a predictive model needs to perform to achieve a specified improvement in one or more KPIs. A simulation-first approach therefore offers a route for early evaluation of predictive model utility which could be used to prioritize efforts on developing models that are more likely to lead to improvements in KPIs if integrated into a clinical or operational workflow. This is analogous to using estimates of the value of information to target clinical research in health economics [268], but additional work may be required to obtain empirical costs instead of the relative costs used for optimization in this study if the analysis forms part of a cost-benefit study. A further advantage of a simulation-first approach is that the estimated KPIs can help to focus the hyperparameter search on models that are likely to have a higher utility. While this workflow requires simulation, for other workflows, such as those that can be modelled as a simple queue, the impact

of a predictive model of a specified quality on KPIs can be investigated analytically [269]. A focus on evaluating potential utility may help to bridge the gap between the large number of ML models that are developed and the relatively small number that have been successfully deployed in healthcare settings [270].

This work has a number of limitations. Firstly, I only considered a single type of platelet and assumed that any unit could fill the demand from any patient. This is a common assumption in the literature [11, 54, 92], but neglects compatibility between the blood type of the donor and the patient and the fact that some patients have special transfusion requirements. The YUPR issuing policy could be adapted to incorporate blood types by, for example, issuing the youngest matching unit or youngest compatible unit for predicted returns. The proposed issuing policy may be less effective when including patients with special requirements because there is less (or no) choice about which unit to issue. I assumed there is no medical reason to allocate fresher platelets to specific patients. Previous studies have considered age-differentiated demand [48, 232] with fresher units preferred for patients with certain conditions, but a recent systematic review found no relationship between platelet storage time and clinical outcomes in critically ill or haematology patients [271] and this distinction is not made in UK guidance [272, 273]. The proposed issuing policy relies on the fact that issued units are kept in conditions that generally allow them to be reissued if they are not transfused and have not expired. This assumption is valid for our partner hospital UCLH (evidenced by the low estimated slippage rate), potentially due to the use of remote platelet agitators located around the hospital, but this may not be applicable at all sites.

Many features in the model are operational and site-specific, such as the ward name and the location where platelet units are to be delivered. As a result, the trained predictive model could not be directly transferred to another site without retraining or adaptation. Additionally, the model may be susceptible to concept drift caused by reorganizations (e.g., changes in how wards are used) or changes in clinical practices and guidelines (e.g., guidance on when a platelet transfusion is appropriate). If the predictive model were deployed, it would be crucial to

continuously monitor its performance and for the team maintaining the model to establish clear lines of communication with clinical and operational staff. This collaboration would help ensure awareness of significant changes that could affect the learned relationships the relied on by the model, and model could be updated to maintain accuracy and reliability.

I have assumed that the hospital blood bank staff would always follow the replenishment and issuing policies, but Wornow *et al.* [258] noted that predictions may not be acted upon in practice. Presenting the possible benefits in terms of meaningful and familiar KPIs instead of ML performance metrics could help to build trust in a policy based on predictive models and increase adherence to the policy.

In evaluating different issuing policies, I used simple heuristic replenishment policies that do not account for the age profile of the stock or any knowledge about the number and age profile of units issued during the day that may be returned. I note that it may be possible to achieve better performance with an OUFO issuing policy using a more sophisticated replenishment policy that takes these factors into account, or by using the return predictions to support both issuing and replenishment decisions.

I focus the discussion of performance on KPIs of wastage and service level because they are easily interpretable in a way that the notional costs assigned to wastage, shortages and holding are not. As with other literature in this area, the relative costs assigned to these different inventory events do not have an empirical underpinning. The relative costs partly reflect the decision makers' concern about consequences with intangible costs that are difficult to determine, such as the impact on a patient's health caused by a delay in treatment due to a shortage or loss in donor confidence due to high wastage [37].

While the YUPR policy gave lower wastage than the OUFO issuing policy under both distributions of remaining useful life on arrival, as assessed using both the estimated KPIs from the simulation experiments and the real demand data from 2017, it is not possible to make a conclusion about the statistical significance of

this reduction. I trained a single model and evaluated it on a single year of data, I therefore could not generate the performance metric distributions required to test the statistical significance. In future work, I would consider training multiple predictive models using bootstrapped samples from the training data and estimating the KPIs for each. This would allow statistical testing to determine whether the reduction in wastage with YUPR compared to OUFO is statistically significant.

The absolute levels of wastage in the simulated system are low relative to previously reported figures but, when using the OUFO issuing policy, are similar to those observed at UCLH during the time period under consideration. Additionally, I optimized the replenishment policies and over-ordering is a commonly cited reason for wastage [274, 275]. Given the potential for the proposed YUPR policy to mitigate the wastage associated with receiving older stock, future work could investigate whether it is possible to achieve system-wide improvements by directing older units to sites able to implement such a policy.

5.7 Conclusion

In this chapter, I proposed a way for hospital blood banks to alter their issuing policies, from a standard OUFO policy, to account for the fact that not all issued platelet units are transfused. I developed a model of the platelet inventory management workflow in a hospital incorporating the observed flow of some issued units being returned unused, and used this simulation to explore the potential utility of the proposed, ML-guided, issuing policy for different levels of predictive model quality and different workflow properties. This analysis showed that, with a sufficiently good predictive model, this new policy can reduce wastage while maintaining service level and I went on to train a sufficiently good supervised learning model.

A policy of issuing the youngest units for predicted returns could support efforts to reduce platelet wastage, and the transfusion laboratory at UCLH is considering how best to implement these ideas. The broader concept of a simulation-first approach may help to target future development of predictive

models to applications most likely to be beneficial in practice.

Chapter 6

Joint optimization of replenishment and issuing policies for multiple products with substitution

6.1 Motivation

A key gap between the scenarios considered in previous chapters and the replenishment decisions made in a hospital blood bank is the importance of ABO/RhD blood types. To provide actionable recommendations for real life management, a replenishment policy should output an order quantity for platelet units of each of these eight blood types. Managing multiple products with complex substitution relationships also requires an issuing policy that recommends which unit to issue in response to demand. Previous work that has included ABO/RhD compatibility in the management of blood product inventory has utilized heuristic issuing policies, optimized an issuing policy with a fixed replenishment policy, or let issuing decisions be made as part of a mathematical optimization process from which there is no way to extract a policy that can be applied in real time. There may be a benefit to jointly optimizing the replenishment and issuing policies, and there is also an advantage to being able to evaluate the performance of different combinations of replenishment and issuing policies in a modular fashion. In this chapter, I therefore modelled perishable inventory management using a multi-agent

approach and implemented three scenarios, with different numbers of perishable products, as multi-agent RL environments in which independent replenishment and issuing policies co-operate to meet demand.

In this problem, the agents act at different frequencies with multiple issuing actions (depending on the demand) between each replenishment action. I used neuroevolutionary methods to fit the policies in this chapter because they learn from performance at the level of an episode (e.g. a simulated year), instead of a step, and so are well suited to the long time horizons and complex, irregular pattern of rewards that will result. Neuroevolutionary methods have several other advantages. Firstly, they can be readily parallelized and combine well with GPU-accelerated RL environments. Secondly, it is conceptually straightforward to incorporate KPIs calculated over a whole episode into the optimization process and to explore alternative methods for balancing different objectives instead of relying on notional costs. Thirdly, because exploration is performed in policy parameter space, it is also straightforward to 'warm-start' the optimization process using a policy network pre-trained to clone the behaviour of a heuristic policy.

6.2 Contribution statement

The main contributions of this chapter are:

- developing GPU-accelerated multi-agent RL environments suitable for learning and evaluating policies when the agents do not act in parallel or in turn;
- exploring whether there is a benefit to jointly fitting replenishment and issuing policies for perishable products when substitution is possible;
- investigating the suitability of neuroevolutionary methods to fit replenishment and issuing policies for multiple perishable products, up to and including the eight different ABO/RhD blood types; and
- fitting replenishment and issuing policies with neuroevolution using KPIs to avoid the need for notional costs, and visualizing the range of feasible

trade-offs between service level and wastage.

6.3 Background and related work

6.3.1 Blood product inventory management considering multiple blood types

In the UK, hospital blood banks specify a number of units of each type (including ABO, RhD, and potentially other features) of platelets when placing a replenishment order with the central provider, NHS Blood and Transplant (NHSBT). However, much of the work on platelet inventory management, including my work in the preceding chapters, has adopted the simplifying assumption that there is only a single type of platelet unit that can meet demand for any patient. While an exact ABO match (and only issuing units from RhD- donors to RhD- patients) is desirable, the short useful life of platelets and the potential mismatch between the distribution of donor and recipient blood types means that, in practice, substitution is used to avoid high wastage levels [20]. As described in Section 2.2, the two main considerations are the potential for haemolysis of the recipient's PRBCs if there is a minor ABO incompatibility between the donor and the recipient and alloimmunization if units from an RhD+ donor are transfused to an RhD- patient [20]. UK guidance from the Blood Stocks Management Scheme states that hospitals should issue an exact ABO match when possible, but that it is acceptable to issue a platelet unit that is not ABO-identical to avoid wastage or placing a replenishment order [226]. A recent US study covering the period 2013-2016 found that only 39% of platelet transfusions to RhD- patients were from RhD- donors, and that 19% of platelet transfusions were minor ABO incompatible [4]. The same study reported that 31% of platelet transfusions were major ABO incompatible, which has been associated with a smaller increase in platelet count after transfusion and a correspondingly shorter time between transfusions [276]. A national survey conducted in Greece in 2015 reported that 88-94% of platelets transfused to RhD-negative patients were from an RhD- donor, 20-23% of platelet transfusions were ABO minor incompatible and 16-24% were ABO major incompatible. These

ranges reflect the differences between platelet units derived from whole blood and those collected via apheresis [5]. Replenishment and issuing policies that incorporate blood types, and the compatibility relationships between them, better reflect the actual decisions made in the hospital blood bank. It may be possible to reduce the number of less preferred matches and, thereby reduce the risk to patients and improve the efficacy of the transfusions, by finding better replenishment and issuing policies. Additionally, it may be possible to reduce the number of platelet transfusions required by minimizing the number of patients given ABO major incompatible transfusions.

In a 2023 review of the application of optimization methods in the blood supply chain, Meneses *et al.* [56] observed that it is not common to consider multiple blood types and the compatibility relationships between them when modelling a hospital blood bank. As I note in Section 2.4.2 researchers using simulation to investigate the impact of changes suggested by human experts have generally incorporated blood type substitution into their work. This aspect of reality has been considered more in the context of managing PRBCs [36, 47, 55, 63, 66, 67, 68, 69] in hospital blood banks using optimization methods, than for platelets [64, 65, 70].

In these studies, when replenishment decisions have been optimized the authors have sought independent heuristic replenishment policy parameters (e.g. order-up-to levels S) for each product [36, 47, 55, 67], or a sequence of replenishment orders for each product over a planning horizon [64, 65, 66, 68, 69].

A straightforward approach to selecting the blood type of the unit that should be issued to a patient is to follow a heuristic policy that selects an unit based on a preference order, with an exact match to the patient's blood type given if possible. This method has been used in research considering the inventory management of platelets [64], and PRBCs [47, 66, 67, 68] in hospital blood banks. Heuristic issuing policies may also include additional considerations such as freshness requirements based on the reason for transfusion [64] and whether the transfusion is required for an emergency case [67], or sample the blood type of the unit to use as a substitute

from a probability distribution [47]. These studies assumed that all demand for a period arises simultaneously, and therefore issuing decisions are made with full knowledge about the demand for a given day. In practice a hospital blood bank must issue units one (or several, as part of one request) at a time without knowing exactly what demand (both in total, and the mix of recipient blood types) will occur over the rest of the day. Heuristic policies that follow the same logic (e.g. issuing based on a substitution preference order) can be developed that serve one unit at a time but the best heuristic rule may be different.

An alternative approach is to make issuing decisions as part of a stochastic mixed integer linear programming process based on permitted substitutions, with or without penalty costs for substitution. This method enables issuing and replenishment decisions to be jointly optimized, and has been used for issuing both platelets [65] and PRBCs [36, 55, 65, 69] in studies set in hospital blood banks. However, the assumption is again made that all demand for a period is known before any issuing decisions are taken and the output of the optimization process is a sequence of decisions taken, so there is no policy that can be extracted and applied in reality. Studies concerned with the decisions of regional blood centres managing multiple blood types have also used either heuristic rules or linear programming decision variables for issuing units to hospitals [10, 80, 277, 278, 279, 280].

Abdulwahab and Wahab [70] and Dumkreiger [63] both used approximate dynamic programming to find issuing policies for platelets and PRBCs, respectively, in hospital blood banks, assuming fixed replenishment policies. Dumkreiger [63] assumed that meeting demand with any compatible unit was the same, while Abdulwahab and Wahab [70] imposed a penalty for substitution with a compatible unit, and a larger penalty for the use of universal donor units unless they were an exact match. The issuing policy fit by Dumkreiger [63] achieved better performance, under a fixed replenishment policy, than the benchmark issuing policy of selecting units in order of priority and she specifically noted that the analysis would be strengthened by jointly optimizing the replenishment policy.

6.3.2 Modelling the problem with multiple agents

Modelling the hospital blood bank as a multi-agent problem would support the evaluation of different combinations of replenishment and issuing policies and enable these two policies to be optimized individually or jointly. These policies could both be represented using neural networks. A neural network issuing policy could be more flexible than heuristic issuing policies for multiple products and take into account additional features including the age profile of each type of product in stock. Unlike the recent work using stochastic mixed integer linear programming to allocate units in a more flexible manner to minimize the objective function [36, 55, 65, 69], it would be possible to use such an issuing policy to support issuing decisions in real time as demand arises. Due to the co-operative nature of the problem, jointly optimizing both policies may help to find better combinations of policies than fixing one and optimizing the other alone: in the context of perishable goods with age-differentiated demand and possible substitution Deniz *et al.* [281] found that it was important to "coordinate issuing decisions with replenishment".

Recent studies have investigated the application of multi-agent RL to supply chain problems including the use of separate agents to set the price of perishable products [282], separate agents to set the replenishment quantity for each of multiple products [155, 283, 284], and agents determining replenishment quantities at different echelons of the supply chain [285, 286]. The most similar example to separate replenishment and issuing agents for managing platelet inventory is Khirwar *et al.* [287], which includes multiple store agents making replenishment orders for multiple products, and a warehouse agent with two policies: placing replenishment orders from a supplier and an allocation policy for filling the orders placed by the stores. However, the warehouse allocation policy acts at the same frequency as the store agents and makes one allocation action once all the store agents have placed their orders for the day. In the scenarios in this chapter, the issuing agent acts between replenishment steps and selects a unit to meet demand as it arises.

The software libraries for multi-agent RL are (as at the time of writing in June

2024) much less well established, and more fragmented, than those for single-agent RL. There is no standard environment API that has reached the same level of acceptance that OpenAI Gym [139] attained for single-agent environments and, as a result, there is a lack of easy-to-use implementations of multi-agent RL algorithms. This is particularly the case when a problem is non-standard - such as the different frequencies of the replenishment and issuing agents in the scenarios considered in this chapter. At the time this work was completed, PettingZoo [130] provided a Python class for multi-agent RL environments for CPU. The Python library PGX [288] offered a class for GPU-accelerated multi-agent environments, but with the requirement that the same agent must act across all parallel environments at each step. This is not suitable for the scenarios considered here, and therefore I developed custom GPU-accelerated environments based on gymnax [289] and PettingZoo.

6.3.3 Main challenges

The size of the action space is a challenge when managing multiple perishable products. In Chapter 3, when placing replenishment orders for a single perishable product, the PPO policy networks output a categorical distribution from which the order quantity was sampled during training. This approach does not scale well to multiple products: the categorical joint-action space for ordering multiple products grows exponentially in the number of products. Additionally, using a categorical distribution does not make use of the fact that replenishment orders for each product are on an interval scale. This prevents generalization of experience to similar order quantities when using DRL methods and therefore more training data is required to explore the action space. One approach to managing multiple products is to have one agent per product, each with its own replenishment policy [155, 283, 284]. An alternative approach, which I adopted in this chapter, is to use a policy neural network with a single continuous output for each product that is mapped to an integer order quantity. This mapping can be achieved by rounding the continuous output [290], by using a similarity measure between the continuous output and the discrete actions [291, 292] or by using a mapping function to convert the continuous output to a feasible discrete action [293]. I adopted the latter approach, which

Vanvuchelen and Boute [293] recently used to obtain replenishment order quantities for up to 40 products using PPO.

A second challenge, specifically related to jointly optimizing the replenishment and issuing policies, is the different frequencies at which the agents act. A common assumption in multi-agent RL problems, including the recent work in the context of supply chains, is that the agents act simultaneously or in turn. However, in the scenarios considered in this chapter there will be multiple issuing steps, with the number varying per day depending on the random demand, between each replenishment action. This leads to relatively long time horizons for the issuing agent, and also to delayed, irregular rewards because certain components of the reward (for example relating to holding costs and wastage) will be received only on issuing steps immediately following a replenishment step. Salimans *et al.* [117] demonstrated the effectiveness of evolutionary strategies for fitting neural network policies on a variety of RL control problems, and suggested that because performance is determined at the episode level (instead of rewards being attributed to individual steps), this and other black-box optimization methods may be particularly appropriate when the reward structure is complicated or sparse, and when time horizons are long. Alongside PPO, I also consider OpenAI ES [117] as an evolutionary strategy and SimpleGA [118] as a genetic algorithm, representing the two main types of evolutionary algorithms used for fitting neural network policy parameters (neuroevolution). These methods are well suited to parallelization, and implementations have been developed in the Python library evosax [121] that are compatible with the GPU-accelerated RL environments I adopted in previous chapters.

6.3.4 Additional benefits of neuroevolution

Two further advantages of neuroevolution for the problem under consideration are the potential to incorporate existing knowledge into the policy parameter search process and the potential to incorporate KPIs calculated over each episode into the optimization process.

For both replenishment and issuing decisions, it is straightforward to obtain

heuristic policies that perform reasonably well: for example, a replenishment policy with parameters fit using simulation optimization and a heuristic issuing policy that selects a unit based on an order of preferences. Using supervised learning, a neural network can be trained to follow these heuristic policies, subject to some error: this process is behavioural cloning. Silver *et al.* [100] used behavioural cloning on expert moves from the board game Go as part of the initial stage of AlphaGo, which beat a leading human player in 2016. The policy neural network for AlphaGo was subsequently trained using policy gradient RL using self-play against a previous version of the policy network. It is not generally straightforward to use a pretrained neural network as the starting point for training a policy with DRL methods. When using PPO, for example, both an effective value network and an approach for exploring when starting with a network representing a deterministic policy would be required because, unlike the human Go experts, the same observed state will always lead to the same output from a heuristic inventory replenishment policy. However, neuroevolutionary methods explore in policy parameter space instead of the state-action space and start with an initial population of possible neural network parameterizations (or a distribution over the neural network parameters). The parameters of the neural network trained using behavioural cloning can therefore be used as a starting point for further optimization using neuroevolution. To the best of my knowledge, this approach has not been used with heuristic inventory policies, but De Moor *et al.* [40] used an alternative approach to incorporate knowledge from heuristic policies into the training process for replenishment policies for perishable inventory using DQN with reward shaping: adding a penalty to the reward that was proportional to the absolute difference between the action taken and the action the heuristic policy would have taken.

A second advantage is that it is conceptually straightforward to optimize policies using KPIs (for example, the service level and the wastage) directly. The fitness score used to update the parameters of the neural network is calculated at the level of an episode (for example, one simulated year). In common with much of the blood product inventory management literature, I have used notional costs

to balance competing priorities in this and preceding chapters. This is particularly helpful for RL methods like PPO where a reward is received at each step, but it can be difficult to determine the appropriate notional costs. Blake *et al.* [37] argued that KPIs are easier for decision makers to reason about, and therefore described an analytical method for identifying policies that comply with restrictions on KPIs (for example, a minimum service level or maximum wastage) under certain simplifying assumptions (e.g. all stock arriving on the same weekday is of the same age, and ABO/RhD blood types can be ignored) and produced exchange curves showing the best performance available on one objective for a fixed target value of the other. Other studies have investigated alternative methods such as goal programming for optimizing multiple objectives in the platelet supply chain [60, 294], optimizing a single objective while applying constraints on others [47, 69] or restricted feasible policies to those meeting maximum or minimum levels of performance in terms of KPIs when managing PRBCs [36, 55]. When comparing potential solutions using multiple objectives an important concept is Pareto efficiency. A solution is Pareto efficient when it is not possible to achieve a better score on one of the objectives without achieving a worse result for at least one other objective [295]. A solution Pareto-dominates another solution if it gives a better result for one objective and the results on the other objectives are no worse. The Pareto frontier is the set of all Pareto efficient solutions. Approximating, and plotting, the Pareto frontier enables a decision maker to understand the possible trade-offs that are available. A single-objective evolutionary algorithm could be used to estimate the Pareto frontier by optimizing one objective at a time subject to constraints placed on the others. Alternatively, some evolutionary algorithms, such as NSGA-II [213], have been specifically designed for multi-objective optimization and directly output a set of solutions approximating the Pareto frontier. Chołodowicz and Orłowski [296] recently used NSGA-II to fit small neural networks representing replenishment policies for a perishable inventory problem with a single product and uncertain deterioration and plotted the Pareto frontier between lost sales and surplus stock.

6.3.5 Relevant work from the wider inventory management literature

In the context of the broader operational research literature, platelet inventory management considering blood types is an example of a joint replenishment problem, and includes substitution between products. A joint replenishment problem is concerned with determining the optimal order quantities for multiple products ordered from the same supplier, subject to a "major" ordering cost that is independent of the number of different products ordered and a "minor" ordering cost that depends on the number of different types of product in the order [297]. In joint replenishment problems the inventory levels of every product may be relevant when determining the optimal order quantity for one product, to help minimize the ordering costs (as in a joint replenishment problem without substitution) and because it may be necessary to consider the total number of units in stock able to serve demand (when substitution is possible). Some of the scenarios considered in this chapter include a "major" (fixed) ordering cost but I implicitly set the "minor" ordering cost to zero. Peng *et al.* [298] gave an overview of research on the joint replenishment problem from 2006 to 2022, following a previous review of work up to 2005 by Khouja and Goyal [297], while Nagpal *et al.* [299] reviewed studies that consider substitution between products. These reviews focused on the joint replenishment problem and substitution in general, and not on perishable inventory in particular. Using the framework described by Shin *et al.* [300] to classify work on substitutable products, the platelet replenishment problem with multiple blood types would be classified as having supplier-driven substitution (because issuing decisions are made by the hospital blood bank instead of being chosen by patients), and a mixture of directions of substitution (because some pairs of patient blood type and donor blood type have bidirectional substitution and others have unidirectional substitution). None of these reviews identified RL or neural network policies as major categories of solution approaches, or examples of jointly optimizing neural network policies for replenishment and issuing policies, but Peng *et al.* [298] identified work by Vanvuchelen *et al.* [301] using PPO for the joint replenishment

problem for products which were not perishable and where substitution was not possible.

Aside from the recent work by Chołodowicz and Orłowski [296] noted above, the use of neuroevolution appears to be limited in the inventory management literature: Prestwich *et al.* [302] used neuroevolution to fit the parameters of a small neural network to represent an inventory control policy in a multi-echelon system, while Jackson [303] used neuroevolutionary methods to develop a meta-model to use in place of an inventory simulation.

6.4 Methods

6.4.1 Overview

In this chapter I consider three scenarios of increasing complexity: managing one, two and eight perishable products. In Table 6.1 I provide a summary of the methods and approaches employed in this chapter.

This section (6.4) sets out the core methods that were applied to more than of these scenarios. I introduce the scenarios in Section 6.4.2 and then describe the GPU-accelerated reinforcement learning environments used to model them, including the multi-agent and adapted single-agent environments which support jointly optimizing replenishment and issuing policies (Section 6.4.3.)

I subsequently describe the replenishment (Section 6.4.4) and issuing policies (Section 6.4.5), followed by the approaches to optimize their performance on a single objective (6.4.6). These approaches include both algorithms used to fit policies in previous chapters (simulation optimization and PPO), and neuroevolutionary methods for fitting neural network policies.

Finally, I explain how I used two advantages of neurevolutionary methods: "warm-starting" the policy search process using supervised pretraining on the actions of a heuristic policy (Section 6.4.7), and directly optimizing multiple KPIs to visualize and understand the trade-offs between them (Section 6.4.9). The relevant KPIs for the scenarios are set out in Section 6.4.8.

Sections 6.5.1, 6.6.1 and 6.7.1 provide additional scenario-specific details of

the methods for the single, two and eight product scenarios respectively.

6.4.2 Scenarios

6.4.2.1 Single product scenario

As an initial proof-of-concept, I selected the single product, single-echelon, periodic review perishable inventory problem described by De Moor *et al.* [40], which was adopted as Scenario A in Chapter 4. I selected this scenario because I had previously computed the optimal replenishment policies as part of the work described in Chapter 4 and the optimal issuing policy is known. Therefore, it was straightforward to determine the relative performance of the replenishment and issuing policies fit using alternative methods. If the alternative methods do not perform well on this simple scenario, they are unlikely to achieve good results on more complex, multi-product scenarios. Re-implementing a scenario from a previous chapter also provides a useful test case for the general multi-agent RL environment Python class I developed as part of the work for this chapter: the return and KPIs for the optimal and heuristic replenishment policies fit in Chapter 4 should be the same in the multi-agent environment as they were in the single-agent environment, subject to random variation due to sampling the demand

My work on the single product scenario is set out in Section 6.5, with a description of the scenario in Section 6.5.1.1.

6.4.2.2 Two product scenario

The two product scenario is an extension of the single product scenario: it is also a single-echelon, periodic review perishable inventory problem but with two products, product A and product B, and unidirectional substitution between them.

My work on the two product scenario is set out in Section 6.6, with a description of the scenario in Section 6.6.1.1.

6.4.2.3 Eight product scenario

The eight product scenario extends the two product scenario to the management of platelets of all eight ABO/RhD blood types and uses previously reported data from the literature for the mean daily platelet demand, demand per blood type, remaining

useful life on arrival, and priority order for substitutions.

My work on the eight product scenario is set out in Section 6.7, with a description of the scenario in Section 6.7.1.1.

			Policy type	Single product §6.5	Two product §6.6	Eight product §6.7
Fitting policies to optimize a single objective	§6.4.6					
		Replenishment policy only				
		Value iteration	Exact	§6.5.1.2		
		Simulation optimization	Heuristic	§6.5.1.2	§6.6.1.4	§6.7.1.4
		Replenishment and/or issuing policies				
		SimpleGA	Neural network	§6.5.1.4	§6.6.1.4	§6.7.1.4
		OpenAI ES	Neural network	§6.5.1.4		
		PPO	Neural network	§6.5.1.4		
Supervised pretraining of policies	§6.4.7					
		Pretrain replenishment policy	Neural network	§6.5.1.5		§6.7.1.4
		Pretrain issuing policy	Neural network			§6.7.1.4
Multi-objective optimization	§6.4.9					
		Replenishment policy only				
		Value iteration	Exact	§6.5.1.6		
		Simulation optimization	Heuristic	§6.5.1.6	§6.6.1.6	
		Replenishment and issuing policies				
		Outer-loop method	Neural network	§6.5.1.6	§6.6.1.6	
		NSGA-II	Neural network	§6.5.1.6	§6.6.1.6	

Value iteration was only used for the single product scenario for computational reasons (see Section 6.6.1.1). Based on the results of experiments for the single product scenario, SimpleGA was selected as the preferred method for fitting neural network policies on the subsequent scenarios (see Section 6.5.2.1). Supervised pretraining was not considered necessary for the two product scenario (see Section 6.6.2.1). Extending the multi-objective optimization to the eight product scenario is considered beyond the scope of this chapter as explained in Section 6.8.

Table 6.1: Overview of how the methods used in in this chapter were applied to each of the three scenarios.

6.4.3 GPU-accelerated reinforcement learning environments

In Chapters 4 and 5, the simulations of the platelet inventory management workflows were implemented as GPU-accelerated RL environments using the Python library gymnax [119]. These environments were single-agent environments in which the agent made a replenishment decision. In this chapter, building on the foundation provided by gymnax, I developed a Python class to represent a generic multi-agent RL environment that could be run in parallel on GPU. For computational reasons, I also a developed an adapted version of the single-agent environment that could be used to jointly fit policies for both replenishment and issuing agents using methods that only require information about policy performance at the level of an entire episode. These environments each provide

a straightforward, modular way to evaluate different combinations of replenishment and issuing policies because, unlike in the single-agent environment where the issuing logic was specified inside the environment definition, both policies can be provided as functions. The key differences between a single-agent gymnax environment, my multi-agent environment, and my adapted single-agent environment are illustrated in Figure 6.1.

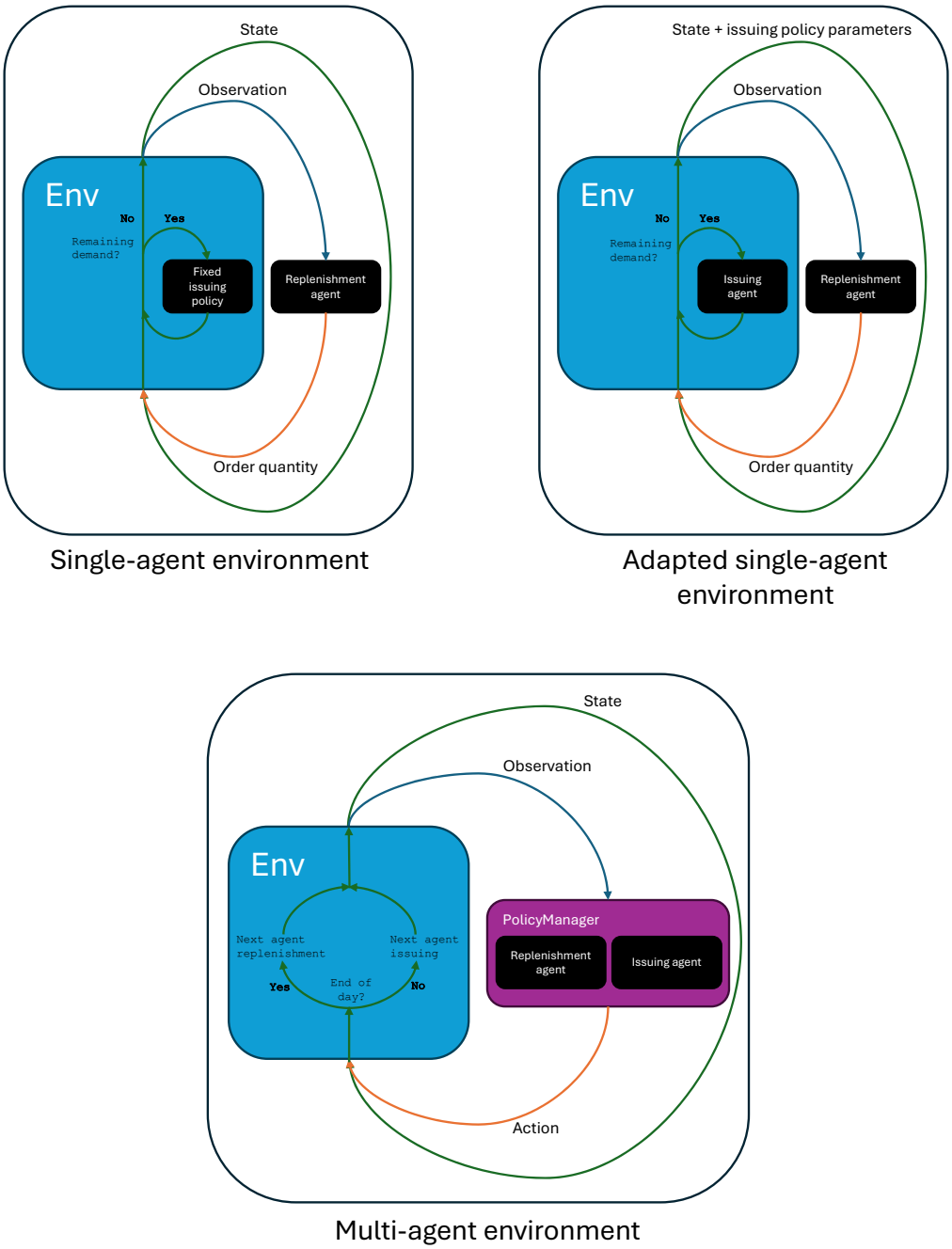


Figure 6.1: Comparison between a single-agent RL environment, adapted single-agent RL environment and multi-agent RL environment.

6.4.3.1 Multi-agent environment

I developed a custom Python class to represent a generic multi-agent environment in which the agents do not have to act simultaneously or in turn. It was based on the GPU-accelerated environment class from gymnax, and the multi-agent RL environment class from the Python library PettingZoo [130].

Similar to to a single-agent gymnax environment, a step in the environment (corresponding to a call to the `step` method of the environment) returns the state, an observation, a reward, Boolean values representing whether the episode is complete, and a data structure containing additional information (`info`). The observation includes an `agent_id` field, an integer which specifies which agent is the next to act. The reward and the `info` are the reward and additional information accumulated for that agent since it last acted.

In addition to the `agent_id` and any environment-specific observation elements for the environment, the data structure for the observation includes an action mask. An action mask is a binary vector where each entry is 1 if the corresponding action is valid in a given state and 0 if it is not permitted. Action masks restrict the action space, ensuring the agent cannot select invalid actions. In the policy-based methods used in this chapter, this is done by setting the probability of selecting any forbidden action to zero before sampling an action.

Following the approach of PettingZoo [130], the Boolean values for episode completion are split into termination and truncation, depending on whether the episode ends because of conditions within the environment or because of an artificial time limit. An episode is considered to be complete when all of the agents have terminated or the episode has been truncated. The environment is automatically reset when this occurs, following the logic for a single-agent gymnax environment.

In order to take advantage of the benefits of JAX by vectorizing and JIT compiling rollouts in the environment, the observation and action spaces must be the same for all agents. This was achieved using padding. I developed a `PolicyManager` Python class to apply the policies and obtain the next action based

on an observation and any policy parameters. It applies the function corresponding to the acting agent's policy based on the `agent_id` in the observation.

This environment supports the training of both replenishment and issuing policies using methods that learn from performance at both an episode (e.g. simulation optimization and neuroevolution) and an individual step (e.g. PPO) level. A subclass of this environment was used for my experiments on the single product scenario, described in Section 6.5.

6.4.3.2 Adapted single-agent environment

Tracking the accumulated reward and additional information for multiple agents requires increasing amounts of memory as the number of products increases, and there is computational overhead associated with each step in the multi-agent environment due to, for example, the need to automatically reset an environment when the episode is complete. In preliminary experiments, I found that this led to much slower episodes when using the multi-agent environment compared to a single-agent environment as the number of products increased. When using a method that learns from performance at the level of an episode (e.g. simulation optimization and neuroevolution), it is possible to avoid these computational penalties by adapting a single-agent environment while still representing the same scenario and evaluating the performance of policies in the same way. I developed the Python class for the adapted single-agent environment to efficiently investigate the problems of interest. However, it is much less general: for example, the current implementation only supports two agents instead of any number of agents.

In the adapted single-agent environment, each step in the environment represents a replenishment step as in the standard single-agent environment: the issuing policy is applied to one unit of demand at a time until the end of the time period represented by the single step. The distinction is that, to support optimizing the issuing policy, the adapted single-agent environment requires a function to be provided for the issuing policy, and issuing policy parameters (for example, a preference order for substitution or the parameters of a neural network) are included in the data structure representing the state. This data structure maintains a separate

state for each parallel rollout and therefore this inclusion supports evaluating multiple different issuing policies in parallel.

The key difference between the adapted single-agent environment and the multi-agent environment is that intermediate observations for issuing are handled inside the environment class and separate information is not captured about the performance of the issuing agent. The performance of the environment is determined using the reward for the replenishment agent, but in all of the scenarios considered in this chapter (whichever environment class is used to implement them) the task is co-operative and the same reward is provided to each agent so this does not impose any additional limitations. As for the multi-agent environment, the observations include an action mask that can be used to restrict available actions.

The adapted single-agent environment is more computationally efficient than the multi-agent environment because there are fewer calls to the environment's step function than in the multi-agent environment (one per day instead of one per day and one per unit of demand), and less memory is required (and fewer associated write operations) because the information about the performance of the issuing agent is not stored when rolling out the policy.

A subclass of the adapted single-agent environment was used for my experiments using both the two and eight product scenarios, described in Sections 6.6 and 6.7 respectively.

6.4.4 Replenishment policies

6.4.4.1 Heuristic replenishment policies

I adopted a base stock policy, with a separate order-up-to level $\mathtt{S}$ for each product, as the baseline replenishment policy. The replenishment order quantities for each product $p \in \mathbb{P}$, $A^{r,p}$ are determined independently:

$$A_t^r = [A_t^{r,p} \quad \forall p \in \mathbb{P}] = \left[\left[\mathtt{S}^p - I_t^p \right]^+ \forall p \in \mathbb{P} \right] \tag{6.1}$$

where $\mathtt{S}^p$ is the order-up-to-level parameter for product p, and I_t^p is the total stock on hand and in-transit for product p at the start of day t.

While more complex heuristic policies exist for ordering multiple products, such as a can-order policy [297, 304], this heuristic policy is simple and widely used, including as a benchmark in a recent study using the DRL algorithm Soft Actor-Critic for managing two perishable products with consumer-driven substitution [161].

6.4.4.2 Neural network replenishment policies

All neural network replenishment policies had a single hidden layer with 64 neurons, and a rectified linear unit (ReLU) activation function between the hidden and output layers.

For the single product scenario, the output layer has $A^r_{\max} + 1$ units: one for each possible order quantity, including ordering no units.

For the two and eight product scenarios, I adopted the approach described by Vanvuchelen and Boute [293] for multi-product replenishment policies: the neural network produces a real number output $\hat{a}^p_t$ for each product p which is then mapped to an integer $\hat{A}^{r,p}_t$. The mapping process is set out in Equation J.1 in Appendix J.

The integer output from the neural network (obtained directly in the case of the single product scenario, and mapped from a continuous output using Equation J.1 for the scenarios with multiple products) was used in one of two ways. Vanvuchelen and Boute [293] used the integer output as an order-up-to level `S` for the product so that, analogously to Equation 6.1, $A^{r,p}_t = \left[\hat{A}^{r,p}_t - I^p_t\right]^+$. I refer to this approach as an "order-up-to" neural network replenishment policy. Alternatively, it can be taken directly as the replenishment order quantity, so $A^{r,p}_t = \hat{A}^{r,p}_t$. I refer to this approach as a "direct action" neural network replenishment policy.

Vanvuchelen and Boute [293] reported an order-up-to neural network replenishment policy improved both performance and the stability of training compared to a direct action neural network replenishment policy. An order-up-to neural network replenishment policy allows for more complex replenishment policies than the heuristic base stock policy, because each state may have a different value of `S`, but it constrains the range of actions available from a given state compared to a direct action neural network. When the continuous neural network

output is mapped to an integer, the the highest possible value of S is capped by a user-defined value $\hat{A}^r_{\max}$. Therefore, for example, in a state with $I_t = 5$, if $\hat{A}^r_{\max} = 10$ the maximum order with an order-up-to neural network would be five units (corresponding to a neural network output of $\hat{A}^r_t = 10$), compared to a maximum possible order of 10 in any state when using a direct action neural network replenishment policy.

6.4.5 Issuing policies

6.4.5.1 Heuristic issuing policies

A FIFO issuing policy was used as the heuristic benchmark policy for the single product scenario. It is equivalent to an OUFO issuing policy for this scenario because all the stock has the same remaining useful life on arrival.

For the two and eight product scenarios, I considered three different benchmark heuristic issuing policies: an exact match policy, a priority match policy, and a policy that issues the oldest compatible unit. The priority match policy and the oldest compatible unit policy require a matrix of preferences which sets out, for each requested type of product, the order of preference for any possible substitutions.

Under the exact match policy, the oldest unit of the requested type is issued if any units of that type are in stock. If not, no units are issued and a shortage is recorded.

Under the priority match policy, the highest preference match for which there are units in stock is selected, and the oldest unit of that type is issued. If there are no compatible units is stock, no units are issued and a shortage is recorded.

Under the oldest compatible match policy, the oldest compatible unit is issued. If there are multiple compatible product types with units of the same age, the product type with the highest priority is issued. If there are no compatible units in stock, no units are issued and a shortage is recorded.

The priority match policy prioritizes compatibility while the oldest compatible match policy prioritizes the age of the stock and may be suitable in cases where the decision maker is less concerned about the consequences of substitution than with wastage.

6.4.5.2 Neural network issuing policies

All neural network issuing policies had a single hidden layer with 64 neurons, and a ReLU activation function between the hidden and output layers. Action masking was used to prevent the policy selecting a type of unit that was incompatible with the request or out of stock, or an age of unit that was out of stock. In these cases the policy would default to not issuing a unit.

For the single product scenario, a unit was selected to be issued based on its age. The output layer of the policy neural network was therefore equal to $m + 1$: one action for products of each possible age, plus an action for not issuing a unit.

For the two and eight product scenarios, a unit was selected based on its type. The output later of the policy neural network was therefore equal to $|\mathbb{P}| + 1$: one action for each type of product plus an action for not issuing a unit. The oldest unit of the selected type was issued. For implementation purposes, so that the same policy class could be used with both an adapted single-agent environment and a multi-agent environment (for which the actions for both agents must be represented by arrays of the same shape to support JIT compilation), the output was converted into a one-hot vector with $|\mathbb{P}|$ elements and the action of not issuing a unit was represented by setting all values in the one-hot vector to zero.

6.4.6 Fitting policies to optimize a single objective

As in previous chapters, in the main experiments for each scenario I optimized a single objective, using notional costs to balance different priorities. The reward functions for each scenario are set out in the description of each scenario below.

6.4.6.1 Simulation optimization

Simulation optimization was used to fit the parameters of heuristic replenishment policies. When practical, as for the single and two product scenarios, all possible combinations of policy parameters were applied to a fixed set of training episodes and the policy parameters giving the highest mean return were selected for subsequent evaluation following the approach used for Scenarios A and B in Sections 4.5 and 4.6. When this was not practical, for the eight product scenario,

the NSGA-II algorithm was used to search the policy parameter space, following the process used for Scenario C in Section 4.7, and the policy parameters giving the highest mean return were selected for subsequent evaluation

6.4.6.2 Neuroevolution

Two evolutionary algorithms, OpenAI ES [117] and SimpleGA [118], were used to fit the parameters of neural networks representing both replenishment and issuing policies. The neural networks were implemented in Flax [305]. I used the implementations of the algorithms from the Python library evosax [121]. Pseudocode representing these algorithms is set out in Algorithm 6 and Algorithm 7 in Appendix J

The main difference between the two approaches is that OpenAI ES maintains the parameters of a normal distribution over the policy parameters and samples each generation of candidates from that distribution. SimpleGA maintains a population of elite solutions, those that have achieved the best fitness so far, and each generation of candidate policy parameters is created by applying a mutation operation to randomly selected candidates from the current set of elite solutions.

For both methods, I took the candidate neural network parameters that achieved the lowest fitness (equivalent to the highest mean return, because the software implementation expects minimization problems by default) on the training episodes for subsequent evaluation.

The original implementation of SimpleGA [118] includes the best parameters identified to date as a candidate solution in each generation and evaluates the 10 best individuals per generation on additional episodes. This was deemed unnecessary here due to the comparatively large number of training episodes used to evaluate each candidate solution.

When jointly optimizing both issuing and replenishment policies, the parameters from both policies were expressed as a single vector for the purposes of running OpenAI ES or SimpleGA.

For both replenishment and issuing policies for the single product scenario, and the issuing policies for the two and eight product scenarios, the action was

determined by taking the argmax over the output layer. For the replenishment policies in the two and eight product scenarios, the continuous output for each product was converted to an integer action following the approach described above in Section 6.4.4.2.

6.4.6.3 PPO

The DRL algorithm PPO [114] was used to fit both replenishment and issuing policies for the single product scenario. I adapted a JAX-based implementation of single-agent PPO, PureJAXRL [306], to support training one or jointly training both of these policies. The neural networks were implemented in Flax [305]. When jointly training the two policies I adopted the Independent PPO (IPPO) [135] approach: each policy was trained individually based on its own observations, actions and rewards without sharing a value-function or any other information. For each agent, the algorithm was the same as previous presented in Algorithm 2 in Chapter 3. The policy and value neural networks for each agent shared the same network structure but did not share parameters. There was no parameter sharing between different agents.

The main challenge with implementing IPPO for the scenarios in this chapter was collecting experience tuples from vectorized, GPU-accelerated environments when the agents acted at different frequencies. This meant that episodes lasted for a different number of steps, and that a fixed number of steps could include a different proportion of replenishment and issuing steps. When using JIT compilation, arrays must be a consistent size. Therefore for each training iteration, fixed-sized arrays were instantiated to hold a specified number of steps from each agent. The policies were rolled-out using a while loop, and terminated when sufficient steps had been collected for each agent in each parallel environment. In some parallel rollouts excess steps were taken and the experience discarded if the fixed-size array for that environment and that agent was already full, but this allowed experience to be collected quickly from a large number of environments in parallel.

PPO was only used for the single product scenario, for which the output of the PPO actor networks was a categorical distribution. During training the

replenishment and issuing actions were sampled from the categorical distribution, while the mode of the output distribution was used to select the action during evaluation.

6.4.6.4 Hyperparameter tuning

The hyperparameters for OpenAI ES, SimpleGA and PPO were tuned using the Bayesian optimization method in "sweeps" functionality provided by the Python library wandb [307]. I refer to a hyperparameter search conducted with this functionality as a "sweep".

Sweeps were conducted on the UCL high performance computing cluster Myriad. For each run in a sweep, the final policy parameters were applied to 1,000 validation episodes generated using a consistent random seed across runs and the sweeps for a given experiment. The policy with the highest mean return for each experiment was selected for subsequent evaluation.

The search ranges for the hyperparameters were based on the default values for OpenAI ES and SimpleGA in the Python library evosax v0.1.5 [121], and for PPO in the GitHub repository PureJAXRL [306], and are set out in Appendix K.

6.4.6.5 Performance evaluation

The best policy parameters identified for each experiment were evaluated on 10,000 evaluation episodes generated using a consistent random seed. KPIs were calculated over these episodes as described in Section 6.4.8 below.

For the single product scenario I report the mean and standard deviation of the return over 10,000 evaluation episodes, as in Section 4.5. Additionally, I report the mean and standard error of the mean of the paired-sample percentage difference between the return of the policy given by each approach and the replenishment policy computed using value iteration.

For the two and eight product scenarios, I report the mean daily cost over 10,000 evaluation episodes, and the mean and standard error of the mean of the paired-sampled percentage difference in daily cost between alternative approaches.

6.4.7 Supervised pretraining of policies

When fitting policies using neuroevolution, the initial neural network parameters were randomly initialized. I have not quantified the impact of this random initialization, or of other stochastic elements, on the quality of the policies produced by the optimization methods for computational reasons. Instead, I prioritized investigating the effects of simulation input parameters that represent features of the inventory management workflow. However, there may be a benefit to initializing the search process at a point in the policy parameter space that performs well based on existing knowledge, so that the subsequent policy search process is focused on an area of policy parameter space known to include reasonable solutions, both in terms of the computational time required and/or the final performance of the identified neural network policies.

The heuristic policies adopted as benchmarks, including the parameters for the base stock replenishment policy fit using simulation optimization, provide a policy that performs well for each problem. I used supervised learning to fit a neural network to match the decisions of a heuristic replenishment or issuing policy. I modified versions of the evosax implementations of OpenAI ES and SimpleGA so that the parameters of the resulting neural network(s) could be used as a starting point for the policy search process.

See Appendix J.3 for additional detail on the supervised pretraining process, including how the observations were collected, and the loss functions and hyperparameters used for training.

6.4.8 Key performance indicators

As in previous chapters, policies were evaluated using KPIs in addition to the return or the mean daily cost. For the single product scenario, the KPIs were calculated in line with the previous work on the same scenario in Chapter 4 (see Section 4.4.4).

For the scenarios with two and eight products an additional KPI, the exact match percentage, was evaluated and the KPIs were calculated in line with the multi-product work on PRBC inventory management by Meneses *et al.* [55]. The overall service level, $\overline{\Lambda}_{\%}$, is the percentage of demand filled with a compatible unit

(Equation 6.2). Overall wastage, $\overline{W}_{\%}$, is the percentage of units received through routine orders that expired (Equation 6.3). Mean holding, $\overline{B}$, is the mean number of units in stock at the end of each day (Equation 6.4). The exact match percentage, $\overline{M}_{\%}$, is the percentage of demand filled from stock that was met with a product that was an exact match (i.e. first preference) (Equation 6.5). These equations show how the metrics were calculated over K episodes each T days long.

For the eight product scenario, modelling platelet inventory management, demand that could not be met from stock would have been met by placing an emergency order. These orders were excluded from both the numerator and the denominator when calculating the exact match percentage. The calculation of percentage wastage using on the number of units received from routine orders as the denominator is consistent with prior work [55], and previous chapters of this thesis, but strictly overestimates wastage compared to the standard "wastage as percentage issued" metric reported by NHSBT, which includes all units received by a hospital blood bank in the denominator.

$$\overline{\Lambda}_{\%} = \frac{1}{K} \sum_{k} \frac{\sum_t D_{t,k} - E_{t,k}}{\sum_t D_{t,k}} \times 100 \tag{6.2}$$

$$\overline{W}_{\%} = \frac{1}{K} \sum_{k} \frac{\sum_t W_{t,k}}{\sum_{i,t} A^{r,i}_{t,k}} \times 100 \tag{6.3}$$

$$\overline{B} = \frac{1}{K} \sum_{k} \frac{\sum_t B_{t,k}}{T} \tag{6.4}$$

$$\overline{M}_{\%} = \frac{1}{K} \sum_{k} \frac{\sum_{i=j,t} M^{i,j}_{t,k}}{\sum_{i,j,t} M^{i,j}_{t,k}} \times 100 \tag{6.5}$$

In Equations 6.2 to 6.5, $D_{t,k}$ is the total demand on day t in episode k, $E_{t,k}$ is the number of units of demand that could not be met from stock due to a shortage on day t in episode k, $W_{t,k}$ is number of units that expire at end of day t in episode k, $A^{r,p}_{t,k}$ is the replenishment action taken for product p on day t in episode k, B_t is the number of units in stock at the end of day t in episode k after disposing of units

that have expired and $M_{t,k}^{i,j}$ is the number of units of product j issued to fill demand for product i during day t in episode k.

6.4.9 Multi-objective optimization

In addition to using notional costs to balance different objectives in a single cost or reward function, I also considered multi-objective optimization methods in order to visualize the range of possible trade-offs between service level, wastage and, in the case of multiple products, the exact match percentage.

I compared different approaches by plotting the estimate of the Pareto frontier generated by each method, and by calculating the hypervolume indicator. The hypervolume indicator [308] quantifies the quality of an approximation of the Pareto frontier. For a given set of points (in this case, the performance of policies in terms of service level and wastage), the hypervolume represents the size of the space dominated by these points, bounded by a reference point that is dominated by all points in the set.

I jointly optimized neural network replenishment and issuing policies using two approaches: an outer-loop method with ϵ-constraints using SimpleGA, and NSGA-II. In both cases, the replenishment policies were direct action neural networks to avoid restrictions on the policy space.

6.4.9.1 Outer-loop method with ϵ-constraints

A multi-objective optimization problem can be converted into a single-objective problem using the ϵ-constraint method: selecting one primary objective to optimize directly, setting upper or lower bounds on the other objectives, and then optimizing the primary objective subject to the constraints. Different trade-offs between the objectives can be explored by varying the levels of the constraints.

To achieve good coverage over both objectives, I adopted a simple outer-loop approach [309] selecting service level and wastage as the primary objective in turn (the outer loop), and jointly fitting replenishment and issuing policies using SimpleGA for different levels of constraint on the other objective (the inner loop). I used SimpleGA in the outer-loop approach because I found it to be effective, and to

be relatively robust to the choice of hyperparameters, in the main experiments for the single product scenario.

The sets of solutions obtained when optimizing each objective were concatenated and filtered to exclude any Pareto-dominated solutions. I refer to this approach as the "outer-loop method".

Due to the large number of optimization runs required by the outer-loop method, I used a single set of hyperparameters for each experiment, set out in Table K.5 of Appendix K.

6.4.9.2 NSGA-II

NSGA-II [213] is a population-based genetic algorithm for multi-objective optimization, designed to maintain a diverse population that approximates the Pareto frontier directly. I set out pseudocode for NSGA-II in Algorithm 8 in Appendix J.

Similar to SimpleGA, NSGA-II generates each new generation through a crossover step (exchanging elements between two parent candidate solutions) and a mutation step. After each generation, candidate solutions to be carried forward are selected based on Pareto fronts. Non-dominated sorting identifies multiple fronts: for instance, $\mathbb{F}_1$ is the Pareto front of the entire set, and $\mathbb{F}_2$ is the Pareto front of the remaining candidates after $\mathbb{F}_1$ is removed. These fronts are sequentially added to the population from which the next generation will be created. When there is insufficient space to accommodate the entire next front, solutions in less densely populated areas of the objective space are prioritized for retention. Parent solutions are selected for reproduction based on the rank of the front to which they have been assigned and the crowding distance around their objective values to maintain population diversity.

Unlike the outer-loop method, NSGA-II can be used to optimize multiple objectives at the same time. For the single product solution, the objectives were service level and wastage while for the two product scenario the objectives were service level, wastage, and exact match percentage.

I used the implementation of NSGA-II from the Python library Optuna [212]

for simulation optimization of heuristic replenishment policies using a single objective. I used the implementation in the Python library pymoo [310] for the multi-objective optimization of neural network policies because I found it more straightforward when optimizing the large number of parameters required for the neural network policies.

Due to the shorter time required to run NSGA-II compared to the outer loop method, it was practical to tune the hyperparameters. For each experiment, I ran a sweep for eight hours on the UCL high performance computing cluster Myriad. For each run in a sweep, the hypervolume indicator was calculated based on the mean performance over 1,000 validation episodes and the set of policies from the run that achieved the highest hypervolume indicator on each experiment was selected for subsequent evaluation. The hyperparameter search ranges are set out in Tables K.6 and K.10 in Appendix K. Unless otherwise stated, I used the default hyperparameters from pymoo v0.6.1.1.

6.4.9.3 Performance evaluation

I compared the different approaches for fitting policies over various combinations of service level and wastage by plotting the non-dominated points to approximate the Pareto frontier and calculating the hypervolume indicator. For simplicity, I did not consider stock holding levels when conducting multi-objective optimization. To normalize the hypervolume indicator to a range between 0 and 1, I defined the ideal point (0% wastage and 100% service level). The normalized hypervolume measures the proportion of the volume between the ideal point and the reference point that is dominated by the points in the set. I set a reference point for each problem, rather than a common worst-case scenario (100% wastage and 0% service level), to better distinguish between different solutions during hyperparameter tuning and evaluation. The hypervolume indicator values should therefore only be used to compare different methods on the experimental settings, and not between experiments.

For each approach to each problem setting, I ran the associated policies on 10,000 evaluation rollouts, calculated the mean service level and wastage over these

episodes as described in Equations 6.2 and 6.3, and used these KPIs to plot the approximate Pareto frontier and compute the normalized hypervolume using the Python library pymoo [310].

6.4.10 Hardware

Hyperparameter tuning was conducted on nodes of the UCL high-performance computing cluster Myriad with 8 CPU cores, 64GB RAM and an Nvidia V100 GPU. Evaluation experiments for multi-objective optimization were conducted using the same nodes.

All other evaluation experiments for results reported in the tables and figures below were conducted on a desktop computer running Ubuntu 20.04 LTS via Windows Subsystem for Linux on Windows 11 with an AMD Ryzen 9 5900X processor, 64GB RAM, and an Nvidia GeForce RTX 3060 GPU.

6.4.11 Code availability

The code supporting the work in this chapter is available at: https://github.com/joefarrington/bloodbank_marl

6.5 Single product scenario

6.5.1 Scenario-specific methods

6.5.1.1 Scenario description

I describe this scenario, introduced by De Moor *et al.* [40] and adopted as Scenario A in Chapter 4, in detail in Section 4.5 and Appendix F.1. In this section I therefore focus on how the scenario can be modelled as an AEC game [130], with separate agents making replenishment and issuing decisions. The AEC game formalism for a multi-agent problem is set out in Section 2.5.3.

I selected a subset of experiments instead of repeating the whole suite of 30 different combinations of maximum useful life, lead time and wastage costs from the original paper and Chapter 4. Specifically, I selected four experiments to consider examples where the maximum useful life m was 2 days or 5 days, and the lead time L was 1 day or 2 days. I selected the experiments with the FIFO

issuing policy and where the wastage cost per unit, $C_w = 7$. This range of problems includes one where, due to the two-dimensional state space, it is possible to plot the policies on a grid ($m = 2, L = 1$), an example with one of the highest optimality gaps between the heuristic and optimal replenishment policies ($m = 2$, $L = 2$), and an example with the largest state space ($m = 5, L = 2$ with 1.8M states).

The state comprises three components: the orders in-transit, the units in stock, and the remaining demand for the current day (which is never observed by an agent). The orders in-transit and units in stock are the same as for the single-agent case in Appendix F.1, but the vector for the orders in-transit also includes the order placed on the current day.

In the initial state, s_0, there are no units in stock or in-transit, and no remaining demand but, as in Chapter 4, I used a warm-up period of 100 days to minimize the influence of the initial conditions because I am interested in the performance of the policies on an ongoing basis rather than from a particular starting point.

There are two agents (excluding the environment agent): one for replenishment and one for issuing. The action space of the replenishment agent is $A^r \in \{0, 1, ..., A^r_{\max} = 10\}$. The action space of the issuing agent is $A^u \in \{0, 1, ..., m\}$, where 0 corresponds to not issuing a unit, m corresponds to issuing a unit on its last day of remaining useful life, and 1 corresponds to issuing units that were received at the start of the current day.

The transition functions for the replenishment, issuing and environment agents are defined by the simulation. At the start of each day the replenishment agent observes the current inventory in stock (split by remaining useful life) and in-transit (split by period ordered), and places a replenishment order. The ordered quantity is added to the to the left-most position of the vector for stock in-transit. The environment agent acts immediately after the replenishment agent, and demand for the new day is sampled from a truncated gamma distribution and rounded to the nearest integer. While there is remaining demand, the issuing agent observes the current inventory in stock (split by remaining useful life) and in-transit (split by period ordered, but excluding the order placed on the current day) and selects a unit

to issue to meet the demand. As a result of this action, remaining demand is reduced by one and the stock of units corresponding to the issuing action is reduced by one, down to a minimum of zero. If the remaining demand is zero after an issuing step then the end of the current day has been reached. The stock is aged one day, expired stock is disposed of, the order placed $L - 1$ days ago is received, and the next day will begin will another replenishment order.

The observation space for the two agents is the same and, for the sake of simplicity, is the same as described in Appendix F.1 for the single-agent environment: stock in-transit and on hand ordered by ascending age. The observation function for each agent is exact and deterministic. By using the same observation as for the single-agent environment, in which only replenishment orders were considered, the observation of the issuing agent is missing one piece of information: the element of the stock in-transit vector corresponding to the order placed at the start of the current day. However, this does not pose a problem because the optimal issuing policy for this scenario is FIFO which does not depend on orders in-transit.

For the multi-agent environment, rewards are assigned to each agent. The reward function includes four components: a holding cost per unit in stock at the end of each day ($C_h = 1.0$), a variable ordering cost per unit ($C_v = 3.0$), a shortage cost per unit of unmet demand ($C_s = 5.0$) and a wastage cost per unit that perishes at the end of each day ($C_w = 7.0$). When an agent takes a step, it receives the total reward accumulated since it last acted.

The single-step reward for the replenishment agent after taking replenishment action A_t^r based on observation O_t^r is the same as in the single-agent version:

$$R_{t+1}^r = -C_v A_t^r - C_h B_t - C_s E_t - C_w W_t \tag{6.6}$$

where A_t^r is the replenishment order placed at the start of day t, B_t is the number of units in stock at the end of day t after disposing of expired units, E_t is the unmet demand due to a shortage of stock during day t, and W_t is the number of units that expired at the end of day t.

The single-step reward for the issuing agent after taking action A_i^u based on observation O_i^u is:

$$R_{i+1}^u = -C_s E_i - \mathbb{1}_{\text{day}(i)\neq\text{day}(i+1)} \left[C_v A_{\text{day}(i+1)}^r + C_h B_{\text{day}(i)} + C_w W_{\text{day}(i)} \right] \quad (6.7)$$

Unless successive steps i and $i+1$ are on different days, only the cost component corresponding to a shortage may be non-zero. E_i is the number of units of unmet demand due to a shortage of stock between steps i and $i+1$. The function $\text{day}(i)$ gives the day t on which step i occurred. At the point where reward R_{i+1}^u is received, $A_{\text{day}(i+1)}^r$ is a replenishment order placed at the start of the current day, $W_{\text{day}(i)}$ is the number of units that expired at the end of the previous day and $B_{\text{day}(i)}$ is the number of units in stock at the end of the previous day after disposing of expired units.

The total reward received by each agent over a whole episode will be the same:

$$\sum_t R_t^r = \sum_i R_i^u \quad (6.8)$$

The next agent function depends on remaining demand. While there is remaining demand at the end of a step, the next agent is the issuing agent. Once remaining demand is equal to zero, the day is assumed to have ended and the next agent is the replenishment agent.

As in Chapter 4, each episode is 365 days long, and preceded by a 100 day warm-up period. All other settings including the demand distribution parameters, cost component values and discount factor are consistent with those used in Chapter 4 and set out in Section 4.5 and Appendix F.1.

6.5.1.2 Replenishment policies

The optimal policies computed using value iteration and the base stock policies fit using simulation optimization in Section 4.5 were used as benchmark policies. These benchmark policies were evaluated on the same set of evaluation episodes as those fit in this chapter. This acted as a test of the multi-agent environment I

developed and to ensure that comparisons made in the results table for this section are all based on the same set of evaluation episodes.

The neural network replenishment policies were order-up-to neural networks as described above in Section 6.4.4.2.

6.5.1.3 Issuing policies

The benchmark issuing policy is a FIFO issuing policy, which is optimal for this scenario.

Neural network issuing policies output the unit to be issued based on age, or opt not to issue a unit, as described above in Section 6.4.5.2

6.5.1.4 Fitting policies to optimize a single objective

I evaluated three different options:

- fitting a neural network replenishment policy, with a FIFO issuing policy;
- fitting a neural network issuing policy, using the optimal replenishment policies established for this scenario using value iteration in Section 4.5; and
- jointly fitting neural networks policies for replenishment and issuing.

For each of these three options, I fit the policy or policies using OpenAI ES, SimpleGA and PPO. To support comparison with the results in Chapter 4, the rewards for the replenishment agent were used to calculate the return when evaluating policies. When fitting policies using the two evolutionary methods the fitness score for each episode was the negative return, the discounted sum of rewards received by the replenishment agent. The per-step rewards for each agent were used when training the policies using PPO.

For PPO, a total of 1M replenishment steps and 4M issuing steps were used for training. This ratio was set to minimize the excess number of steps required when the policies were being jointly trained because the mean daily demand, $\mu = 4$.

For each of the 36 experiments (four scenario settings, three optimization methods, and three options for which policies were fit), I ran a hyperparameter

tuning sweep for 5 hours. The hyperparameter search ranges are set out in Tables K.1, K.2 and K.3 in Appendix K.

6.5.1.5 Supervised pretraining of policies

The main results (see Table 6.2) suggest that the settings where the lead time $L = 2$ were more challenging. It may be possible to achieve better results by initializing the policy search in an area of policy parameter space known to give reasonable performance on the task. I therefore conducted an supplementary set of experiments using supervised pretraining, concentrating on the settings where $L = 2$.

Neural networks were trained using supervised learning to match the outputs of a heuristic base stock policy, with target stock level `S` set based on the simulation optimization experiments from Section 4.5, as listed in Table G.1 in Appendix G. See Appendix J.3 for a detailed description of the supervised pretraining procedure.

For these experiments I considered both order-up-to and direct action policy neural networks. The latter approach was considered because when pretraining the order-up-to policy network, the network would only have to learn to output a single number (the value of `S` for the heuristic policy), and this may be too simple a task to provide good initial parameters for further optimization. In contrast, the direct-action neural network must learn to represent the complete base stock policy to output the correct order quantity for each state.

After the supervised pretraining was complete, the hyperparameter tuning and evaluation process used for the main experiment was repeated for both OpenAI ES and SimpleGA, using the parameters learned using supervised pretraining to initialize these methods for each run in each sweep as described in Section 6.4.7. Additionally, to confirm whether differences resulted from pretraining or the different neural network output, I performed the same steps for direct-action policy networks without initializing the policy search process using pretrained parameters.

6.5.1.6 Multi-objective optimization

I used four approaches to estimate the Pareto frontier of possible trade-offs between service level and wastage for the single product scenario: the outer-loop method (Section 6.4.9.1), NSGA-II (Section 6.4.9.2), and two benchmarks: a base stock

replenishment policy and replenishment policies computed using value iteration with a range of different values for the cost components.

When using the outer-loop method, I fit neural network policies for replenishment and issuing using SimpleGA and a single set of hyperparameters as described Section 6.4.9.1. The hyperparameters are set out in Table K.5 in Appendix K.

When using the outer loop method I minimized the wastage subject to 25 linearly spaced target minimum service levels, using the fitness function:

$$\min_{\theta^r,\theta^u} F(\theta^r,\theta^u) = \overline{W}_{\%} + \lambda \mathbb{1}_{\overline{\Lambda}_{\%}<\Lambda_{\text{min}}} \tag{6.9}$$

and maximized the service level (expressed here in line with convention as minimizing the negative service level) for 25 linearly spaced target maximum wastage levels using the fitness function:

$$\min_{\theta^r,\theta^u} F(\theta^r,\theta^u) = -\overline{\Lambda}_{\%} + \lambda \mathbb{1}_{\overline{W}_{\%}>W_{\text{max}}} \tag{6.10}$$

where θ^r and θ^u are the parameters of the replenishment and issuing policies respectively, Λ_{min} is the target minimum service level, W_{max} is the target maximum wastage and λ is a Lagrange multiplier.

For the two scenario settings with $m = 2$, the target minimum service levels were between 50% and 99% inclusive, and the target maximum wastage percentages were between 0.1% and 20% inclusive. For the two scenario settings with $m = 5$, the target minimum service levels were between 75% and 99.5% inclusive, and the target maximum wastage percentages were between 0.0001% and 2% respectively.

NSGA-II was used to fit neural networks for both the replenishment and issuing policies by minimizing wastage and maximizing service level as described in Section 6.4.9.2.

For the base stock policy, each value of $\mathtt{S} \in \{0, 1, ..., A^r_{\text{max}} = 10\}$ was

considered. Value iteration was used to compute the optimal policy for a range of combinations of wastage cost C_w and shortage cost C_s using the approach described in Section 4.5. The wastage cost C_w was fixed at 40 and 100 logarithmically spaced points between 3.1 (slightly higher than the variable order cost, $C_v = 3.0$) and 400 inclusive were used for the shortage cost, C_s. A FIFO issuing policy was used for both of these benchmarks.

When calculating the hypervolume indicator for both tuning the hyperparameters for NSGA-II and final evaluation of all approaches I used a reference value of service level = 50% and wastage = 30% for the two scenario settings with $m = 2$, and service level = 75% and wastage = 2% for the two scenario settings with $m = 5$ based on preliminary experiments.

6.5.2 Results

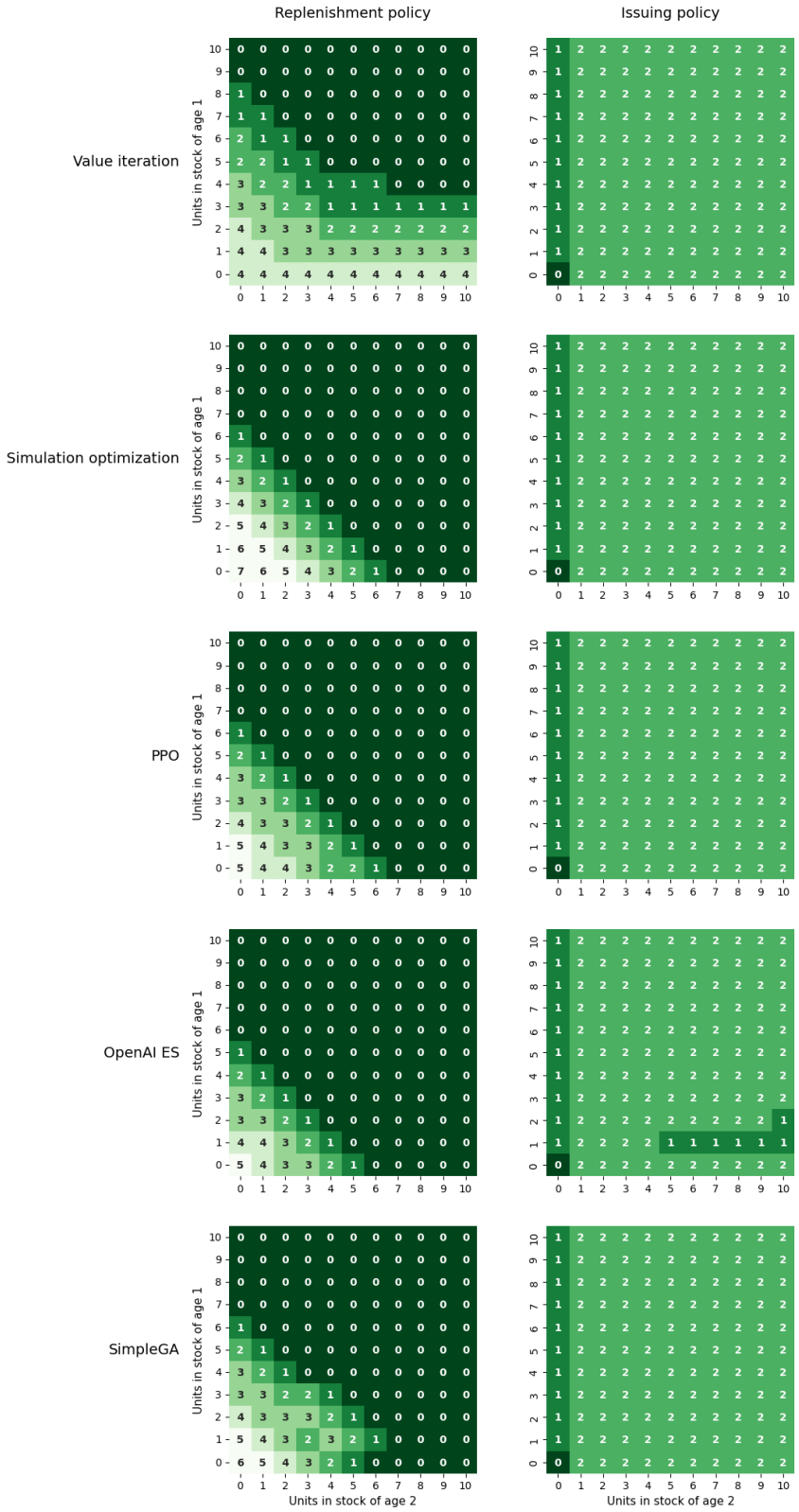
6.5.2.1 Main results

The results for the single product scenario are presented in Table 6.2. The results for replenishment policies using value iteration and simulation optimization concur with those in Table 4.3 in Chapter 4, providing good evidence that the multi-agent RL environment operated as intended. All three methods for fitting neural network issuing policies were able to learn a FIFO issuing policy in each experimental setting. SimpleGA consistently gave the lowest optimality gaps (with a maximum of 0.79%), and the performance was similar when fitting a replenishment policy alone and jointly fitting both the replenishment and issuing policies. These results show that effective policies for replenishment and issuing can be learned when jointly optimized using neuroevolutionary methods. The optimality gaps for PPO and OpenAI ES are higher when jointly fitting the replenishment and issuing policies than when fitting the replenishment policy alone. Jointly fitting the policies with SimpleGA and OpenAI ES acheives better returns than simulation optimization, even though simulation optimization has been given the optimal issuing policy.

In Figure 6.2, I present the optimal policy computed using value iteration, the heuristic replenishment policy fit using simulation optimization (both alongside a

FIFO issuing policy), and the replenishment and issuing policies fit jointly using OpenAI ES, SimpleGA and PPO for the experiment with $m = 2$ and $L = 1$.

I observed from the hyperparameter tuning process that SimpleGA gave more consistent results over a range of hyperparameters than the other approaches. I therefore decided to use SimpleGA for the two and eight product scenarios because it gave performance closest to the optimal policies and appears to be less sensitive to the choice of hyperparameters.



The value iteration and simulation optimization policies were fit as part of the work for Section 4.5 with FIFO issuing policies. The PPO, OpenAI ES and SimpleGA rows correspond to the experiments where the replenishment and issuing policies were jointly optimized. The annotations in each grid are the actions taken in each state – order quantities for the replenishment policies and the age of the stock for the issuing policies (with 0 representing the decision to not issue a unit).

Figure 6.2: Replenishment and issuing policies for the single product scenario with $m = 2$ and $L = 2$.

		Value iteration		Simulation optimization			PPO			OpenAI ES			SimpleGA		
m	L	Return		Return		Opt	Return		Opt	Return		Opt	Return		Opt
		Mean	*(s.d.)*	*Mean*	*(s.d.)*	gap (%)	*Mean*	*(s.d.)*	gap (%)	*Mean*	*(s.d.)*	gap (%)	*Mean*	*(s.d.)*	gap (%)
Fit replenishment policy, issuing policy FIFO															
2	1	-1,457	(61)	-1,474	(57)	1.19 ± 0.01	-1,458	(60)	0.03 ± 0.00	-1,458	(61)	0.05 ± 0.00	**-1,457**	(61)	0.00 ± 0.00
	2	-1,461	(60)	-1,495	(62)	2.32 ± 0.01	-1,507	(65)	3.12 ± 0.01	-1,468	(60)	0.42 ± 0.01	**-1,466**	(60)	0.35 ± 0.01
5	1	-1,422	(57)	-1,430	(54)	0.54 ± 0.01	-1,426	(54)	0.28 ± 0.01	-1,422	(54)	0.01 ± 0.00	**-1,422**	(54)	0.01 ± 0.01
	2	-1,432	(57)	-1,453	(60)	1.45 ± 0.01	-1,495	(66)	4.38 ± 0.01	-1,445	(60)	0.88 ± 0.01	**-1,443**	(60)	0.76 ± 0.01
Fit issuing policy, replenishment policy as per value iteration															
2	1						**-1,457**	(61)	0.00 ± 0.00	**-1,457**	(61)	0.00 ± 0.00	**-1,457**	(61)	0.00 ± 0.00
	2						**-1,461**	(60)	0.00 ± 0.00	**-1,461**	(60)	0.00 ± 0.00	**-1,461**	(60)	0.00 ± 0.00
5	1						**-1,422**	(57)	0.00 ± 0.00	**-1,422**	(57)	0.00 ± 0.00	**-1,422**	(57)	0.00 ± 0.00
	2						**-1,432**	(57)	0.00 ± 0.00	**-1,432**	(57)	0.00 ± 0.00	**-1,432**	(57)	0.00 ± 0.00
Jointly fit replenishment and issuing policies															
2	1						-1,458	(61)	0.05 ± 0.00	-1,471	(63)	0.92 ± 0.01	**-1,457**	(61)	0.00 ± 0.00
	2						-1,518	(66)	3.90 ± 0.01	-1,478	(61)	1.14 ± 0.01	**-1,466**	(60)	0.35 ± 0.01
5	1						-1,430	(54)	0.55 ± 0.01	-1,424	(56)	0.12 ± 0.00	**-1,422**	(54)	0.02 ± 0.00
	2						-1,497	(66)	4.48 ± 0.01	-1,449	(60)	1.18 ± 0.01	**-1,444**	(60)	0.79 ± 0.01

The return was calculated for each episode and the mean and standard deviation of the return over the 10,000 evaluation episodes is reported. The percentage difference in return was calculated between the value iteration policy and each other approach per episode, and the reported optimality gap is the mean ($\pm$ the standard error of the mean) percentage difference over the 10,000 evaluation episodes. Value iteration and simulation optimization replenishment policies were fit in Section 4.5 The best mean return achieved by a non-exact method is indicated in **bold** (multiple entries may be bold in the case of a tie).

Table 6.2: Results for fitting the replenishment policy alone, the issuing policy alone, and jointly fitting both the replenishment and issuing policies for the single product scenario

6.5.2.2 Supervised pretraining of policies

The results for the supervised pretraining experiments are set out in Table 6.3. The benefits of supervised pretraining are limited to direct action replenishment neural network policies. For two of the four cases with an order-up-to neural network policy for replenishment the return was lower when using supervised pretraining than when starting from a random initialization. The direct action neural network achieves higher return in all cases, both with and without pretrainining, and gives an optimality gap of 0.00% for the two cases where $m = 2$. This suggests that pretraining can be beneficial when the neural network learns a complex function but not, as was the case for pretraining the order-up-to network, when the network is pretrained just to output a single number.

		Value iteration	Order-up-to neural network		Direct action neural network	
			No pretraining	Pretrained	No pretraining	Pretrained
m	L	*Mean return*	*Opt gap (%)*	*Opt gap (%)*	*Opt gap (%)*	*Opt gap (%)*
Jointly fit replenishment and issuing policies with SimpleGA						
2	2	-1,461	0.35 ± 0.01	0.31 ± 0.01	0.09 ± 0.00	$\mathbf{0.00 \pm 0.01}$
5	2	-1,432	0.79 ± 0.01	0.81 ± 0.01	0.20 ± 0.00	$\mathbf{0.09 \pm 0.00}$
Jointly fit replenishment and issuing policies with OpenAI ES						
2	2	-1,461	1.14 ± 0.01	0.98 ± 0.01	0.04 ± 0.00	$\mathbf{0.00 \pm 0.00}$
5	2	-1,432	1.18 ± 0.01	1.45 ± 0.01	0.57 ± 0.01	$\mathbf{0.20 \pm 0.00}$

The percentage difference in return was calculated between the value iteration policy and each other approach per episode, and the reported optimality gap is the mean ($\pm$ the standard error of the mean) percentage difference over the 10,000 evaluation episodes. The value iteration replenishment policies were computed in Section 4.5. Results for order-up-to neural network replenishment policies with no pretraining are from Table 6.2. The lowest optimality gap for each row is indicated in **bold**.

Table 6.3: Optimality gaps when jointly fitting replenishment and issuing policies using SimpleGA and OpenAI ES for the single product scenario, with and without supervised pretraining based on a heuristic replenishment policy.

6.5.2.3 Estimating the Pareto frontier for service level and wastage

The estimated Pareto frontiers are plotted in Figures 6.3 to 6.6 for each of the four experimental settings respectively. The points representing solutions from a given approach are connected by a stepped line. Any points located below this line are Pareto-dominated by a solution from that approach. The hypervolume indicators for the Pareto frontiers estimated using value iteration, the outer-loop approach, and NSGA-II are presented in Table 6.4. Note that the Pareto frontiers estimated using the heuristic base stock replenishment policy do not span the whole range of the KPIs due to the structure of the policy. I therefore did not calculate the hypervolume for the base stock policy for this scenario.

When $m = 2$, the Pareto frontiers estimated using NSGA-II and the outer loop method closely match those estimated by value iteration, and generally offer improved performance compared to the base stock policy over the range for which there are comparable solutions. For the problems with a larger state space, when $m = 5$, the performance of both approaches is worse compared to value iteration, but the outer-loop approach gives the better performance of the two over the range of KPIs and could still provide a useful basis for discussions with decision makers about possible trade-offs.

These results demonstrate that jointly fitting replenishment policies using

SimpleGA is effective across a range of possible trade-offs between KPIs and that the outer-loop method can be used to obtain a good estimate of the Pareto frontier under different scenario settings. For the simpler problems, NSGA-II is a compelling alternative because it requires only a single optimization run and took approximately five minutes to run, compared to two hours for the outer-loop approach,

Due to the computational time required for multiple runs of SimpleGA with different constraints in the outer-loop method, I only used a single set of hyperparameters across all four experiments. It may therefore be possible to bring the Pareto frontier estimated using the outer-loop method closer to that produced using value iteration by tuning the hyperparameters.

The estimated Pareto frontiers could be used as the basis for discussions with decision makers, to help explain feasible trade-offs and select a combination of policies that matches their preferences. For example, for the scenario where $m = 2$ and $L = 1$ in Figure 6.3, it is possible to achieve a service level of 95% if the decision maker is willing to accept wastage of 10%, but if wastage cannot exceed 5% then the best possible service level is less than 90%.

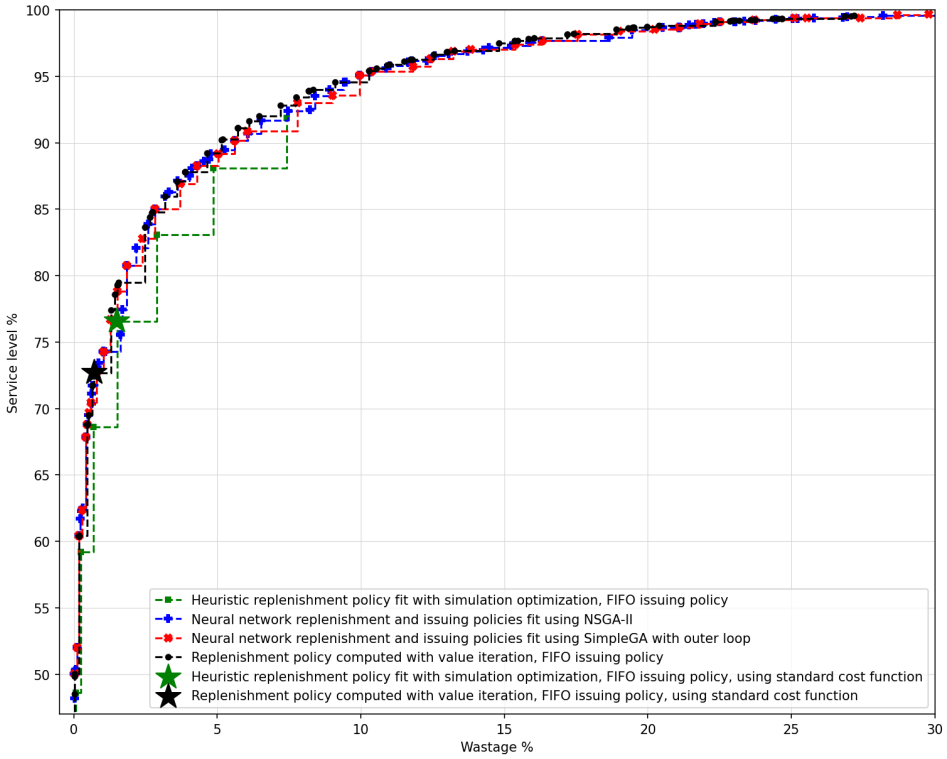


Figure 6.3: Estimated Pareto frontiers for the single product scenario with $m = 2$, $L = 1$

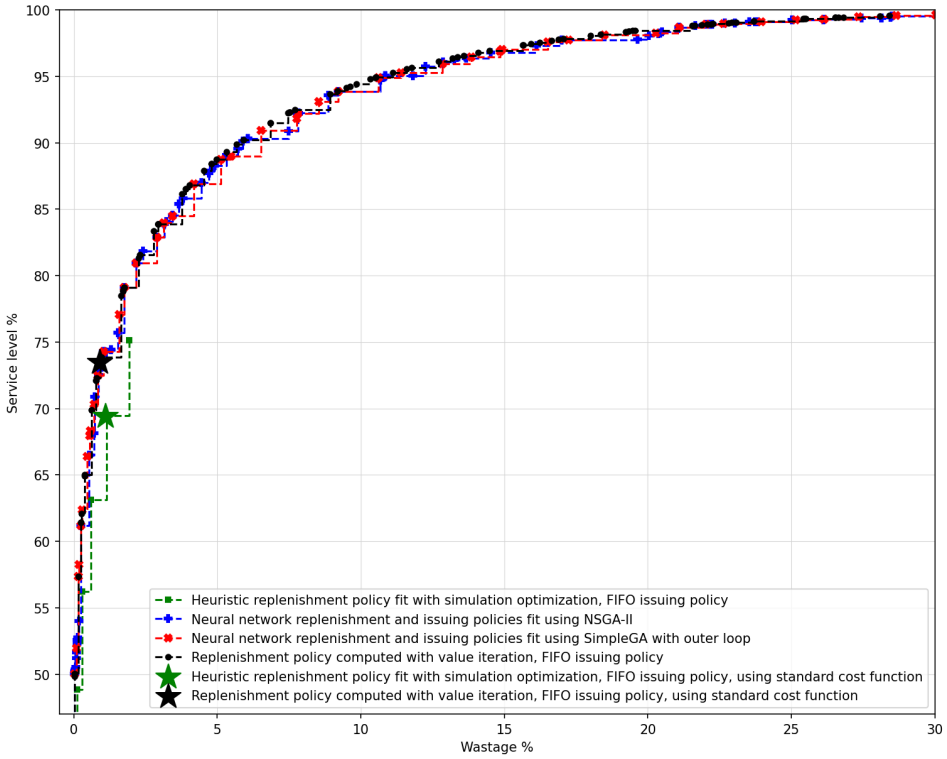


Figure 6.4: Estimated Pareto frontiers for the single product scenario with $m = 2$, $L = 2$

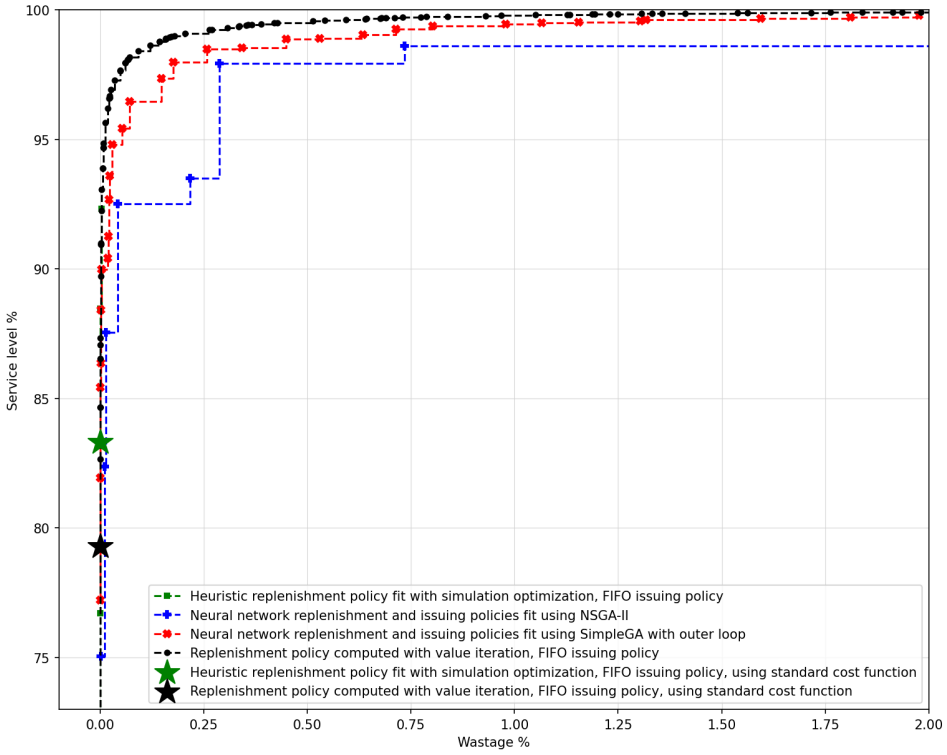


Figure 6.5: Estimated Pareto frontiers for the single product scenario with $m = 5$, $L = 1$

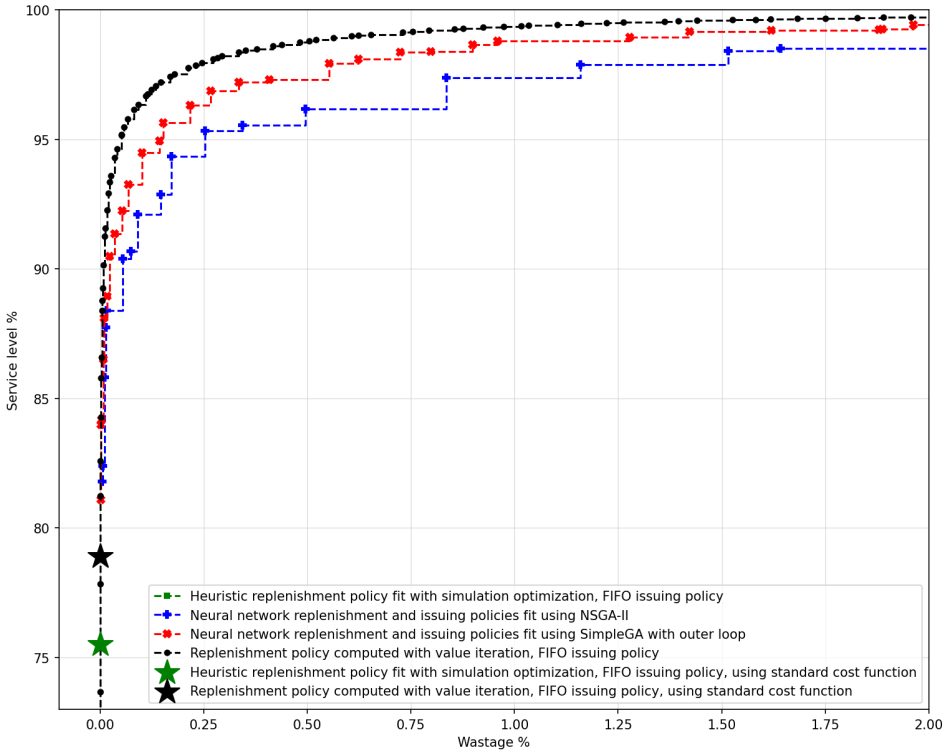


Figure 6.6: Estimated Pareto frontiers for the single product scenario with $m = 5$, $L = 2$

m	L	Value iteration	Outer-loop	NSGA-II
2	1	0.884	0.879	0.881
2	2	0.874	0.869	0.869
5	1	0.981	0.957	0.898
5	2	0.955	0.917	0.862

The service level and wastage were calculated per episode. The hypervolume indicator for each method was calculated using the mean service level and wastage over 10,000 evaluation episodes for each solution.

Table 6.4: Hypervolume indicator for the estimated Pareto frontiers for the single product scenario.

6.6 Two product scenario

6.6.1 Scenario-specific methods

6.6.1.1 Scenario description

I next consider a scenario with two products, product A and product B, with unidirectional substitution: product A can be used to fill demand for product B, with some substitution penalty, but the reverse is not true.

The key differences between the single product scenario and the two product scenario are:

- for each unit of demand the product type requested, product A or product B, is sampled from a categorical distribution;
- daily demand is assumed to be Poisson distributed instead of being sampled from a gamma distribution and rounded to an integer;
- the state and observation include vectors for stock on hand and in-transit for each product type;
- the state includes an integer representing the type of the next required product, product A or product B, and the time of day (as a decimal);
- the observation for the issuing agent is different to that for the replenishment agent and includes the type of the next requested product and the time of day;
- the order placed at the start of the current day is included in the vectors of stock in-transit;

- the reward function includes a term for a substitution penalty when demand for product B is met with a unit of product A;
- the discount factor γ is set to unity and the mean daily cost is used to report performance instead of the return.

This scenario was implemented using an adapted single-agent environment, described above in Section 6.4.3.2, instead of the full multi-agent environment used for the single product scenario. In preliminary experiments I found that running episodes for this scenario in the full multi-agent environment took between four and five times longer than in a single-agent, or adapted single-agent, environment. The multi-agent environment is essential for using PPO and other DRL methods than learn from the reward received at individual steps, but SimpleGA was the best performing method on the single product scenario and does not require the step-level information. Reducing the time required for each experiment enabled me to consider a wider range of input settings.

The daily demand is modelled as a Poisson arrival process with $\mu = 4$ up to the maximum demand $D_{\max} = 100$. An issuing step is made for each interval while the cumulative sum of the intervals was less than one day. At the end of the day units are aged one day, expired stock is disposed of, the holding position is calculated, and the order placed $L - 1$ periods ago is received. This approach means that a representation of the time of day (the cumulative sum of intervals) can be included in the observation for the issuing agent which may be useful when deciding which of multiple products to issue.

In the base setting it is equally likely that a given unit of demand will be for product A or product B: the parameters of the categorical distribution are $[0.5, 0.5]$.

The reward function is the same as for the replenishment agent in the single product case, but with an additional term for the substitution cost. $C_m^{i,j}$ is the cost for issuing a unit of type j to meet demand for product type i. Impermissible matches are given an arbitrarily large value set in practice to $1e10$. $M_t^{i,j}$ is the number of units of product j issued to fill demand for product type i during day t. No substitution cost is incurred in the case of a shortage. For the base setting of this

scenario $C_m^{b,a} = 1$, $C_m^{a,b} = 1e10$ and $C_m^{a,a} = C_m^{b,b} = 0$.

$$R_{t+1} = -C_v A_t^r - C_h B_t - C_s E_t - C_w W_t - \sum_{i,j} C_m^{i,j} M_t^{i,j} \tag{6.11}$$

For this scenario, there is no separate reward function for the issuing policy because the issuing steps are handled within the `step` method of the adapted single-agent environment. It is not necessary because, for this co-operative task, the total reward for both policies will be the same and the methods used to fit policies for this scenario learn from performance at the level of whole episodes rather than individual steps.

The replenishment action consists of two elements, one order for each product: $A_t^r = [A_t^{r,a}, A_t^{r,b}]$ where $A^{r,a} \in \{0, 1, ..., A_{\max}^{r,a} = 10\}$ and $A^{r,b} \in \{0, 1, ..., A_{\max}^{r,b} = 10\}$. The issuing action space is $A^u \in \{0, 1, 2\}$: 0 corresponds to not issuing a unit, 1 corresponds to issuing a unit of product A and 2 corresponds to issuing a unit of product B. The oldest available unit of the selected type is issued.

As for the single product case, each episode is 365 days long and preceded by a 100 day warm-up period. All other settings including the mean daily demand and cost component values are the same as in the single product case and for the base setting the lead time $L = 1$ and maximum useful life $m = 2$.

In Chapter 4 I considered a different scenario with partial, unidirectional substitution between two perishable products. A major difference here, which makes the scenario both more realistic and more challenging, is that in this case I did not assume that demand for both products is known before any issuing decisions are made. The issuing decisions must be made for one unit of demand at a time. This better reflects reality and provides a policy that could be applied in practice in cases where demand arises throughout the day (as in a hospital blood bank). However, using value iteration to compute the optimal policy (which I was able to do for the two product scenario in Chapter 4) would require considering all of the different possible temporal orderings of demand (e.g. sequences of demand for product A and product B) in addition to demand quantity and therefore, for computational reasons, I did not attempt to compute the optimal policy for this scenario.

6.6.1.2 Replenishment policies

The heuristic benchmark replenishment policy was a multi-product base stock policy, which has two parameters: an order-up-to level $\mathtt{S}$ for each of the two products as described above in 6.4.4.1.

The neural network replenishment policies were order-up-to neural networks as described above in Section 6.4.4.2.

6.6.1.3 Issuing policies

The benchmark issuing policies were the three multi-product heuristic issuing policies set out in Section 6.4.5.1. In this case, demand for product A can only be filled with units of product A, while product A is the second preference (after product B itself) for filling demand for product B.

Neural network issuing policies output the type of unit to be issued (product A or B), or opt not to issue a unit, as described above in Section 6.4.5.2

6.6.1.4 Fitting policies to optimize a single objective

The parameters of the multi-product base stock policy were fit using simulation optimization for each of the three heuristic issuing policies in turn, following the approach adopted in Section 4.5. The parameter space is small, so all combinations of $\mathtt{S}^a \in \{0, 1, ..., A^{r,a}_{\max} = 10\}$ and $\mathtt{S}^b \in \{0, 1, ..., A^{r,b}_{\max} = 10\}$ were run using the Optuna grid sampler. Each combination was evaluated on 4,000 training episodes and the combination that achieved the highest mean return was selected for subsequent evaluation.

Neural network replenishment policies were fit using SimpleGA with each of the three heuristic issuing policies in turn. Additionally, the neural network replenishment and issuing policies were jointly fit using SimpleGA. I ran a hyperparameter tuning sweep for 5 hours for each of the four experiments with SimpleGA. The hyperparameter search ranges are set out in Table K.7 in Appendix K.

6.6.1.5 Sensitivity analyses

I conducted sensitivity analyses to investigate the effectiveness of different replenishment and issuing policies on cases with different input assumptions, comparing a range of values for the lead time L, the maximum useful life m, the parameters of the categorical distribution over product types, the ratio of shortage and wastage costs, the mean daily demand and the substitution cost for filling demand for product B with product A, $C_m^{b,a}$.

When the mean daily demand was increased, the maximum order quantity $A^r_{\max}$ was also increased: $A^r_{\max} = 17$ when $\mu = 8$, $A^r_{\max} = 28$ when $\mu = 16$ and $A^r_{\max} = 49$ when $\mu = 32$. These were set so that the cumulative distribution of the Poisson distribution with mean μ evaluated at $A^r_{\max}$ was the same in each case.

For each level of the sensitivity analysis I repeated the main analysis, but using a fixed set of hyperparameters for SimpleGA across all the experiments instead of performing hyperparameter tuning. The 10,000 evaluation episodes were generated using a consistent random seed that was different from those used for the base case and estimating the Pareto frontier.

6.6.1.6 Multi-objective optimization

When estimating the Pareto frontiers for the two product scenario I considered the exact match percentage as an objective in addition to service level and wastage. To use the outer-loop approach, which only supports the optimization of one objective at a time, I considered three target levels of exact match percentage: 50%, 75% and 95%.

Replenishment and issuing policies were jointly fit using the outer loop approach for each of these three levels of exact match percentage. I minimized the wastage subject to 25 linearly spaced minimum service level targets between 10% and 99.5%, using the fitness function:

$$\min_{\theta^r,\theta^u} F(\theta^r,\theta^u) = \overline{W}_{\%} + \lambda \mathbb{1}_{(\overline{\Lambda}_{\%} < \Lambda_{\min}) \vee (\overline{M}_{\%} < M_{\min})} \tag{6.12}$$

and maximized the service level (expressed here in line with convention as

minimizing the negative service level) for 25 linearly spaced target maximum wastage levels using the fitness function:

$$\min_{\theta^r,\theta^u} F(\theta^r,\theta^u) = -\overline{\Lambda}_{\%} + \lambda \mathbb{1}_{(\overline{W}_{\%} > W_{\max}) \vee (\overline{M}_{\%} < M_{\min})} \tag{6.13}$$

where θ^r and θ^u are the parameters of the replenishment and issuing policies respectively, $\Lambda_{\min}$ is the target minimum service level, $W_{\max}$ is the target maximum wastage, $M_{\min}$ is the target minimum exact match percentage and λ is a Lagrange multiplier to enforce the constraints.

The target minimum service levels were between 10% and 99.5% inclusive, the target maximum wastage percentages were 1% and 40% inclusive. In an effort to avoid cases where the constraints were met on the training episodes but not on the evaluation episodes, the target exact match percentages were increased by 0.5 percentage points when implementing the constraints.

In addition to the outer-loop method, NSGA-II was run to simultaneously optimize all three objectives: minimize the wastage and maximize both the service level and exact match percentage as described in Section 6.4.9.2.

The performance of each policy was compared to three benchmarks: the base stock replenishment policy with each of the three heuristic ordering policies. Each combination of $\mathtt{S}^a \in \{0, 1, ..., A^{r,a}_{\max} = 10\}$ and $\mathtt{S}^b \in \{0, 1, ..., A^{r,b}_{\max} = 10\}$ was evaluated using each of the three heuristic issuing policies.

When tuning the hyperparameters for NSGA-II, I calculated the hypervolume indicator using a reference point of service level = 0%, wastage = 100%, exact match = 0%. This reference point was chosen as it represents the worst-case scenario, covering the entire objective space, ensuring that the hypervolume calculation could reflect the performance of any possible set of policies. For the results table I used a reference point of service level = 0% and wastage = 45%, to make the numeric metric consistent with the area of interest depicted in the plots. Although the reference point affects the absolute value of the hypervolume, a better solution (which dominates more of the objective space) will have a higher hypervolume under both sets of reference points.

For each plot, the set of policies produced using each method was filtered to remove solutions with an exact match percentage on the evaluation episodes less than or equal to the target, $M_{\min}$, and any solutions that were Pareto dominated by another solution produced using the same method.

6.6.2 Results

6.6.2.1 Base case

The results for the two product scenario are presented in Table 6.5. The expected daily cost for a policy with perfect foresight, which ordered quantities of products A and B each morning exactly equal to the demand for the following day and therefore incurred no holding, shortage or wastage costs, is 12.00.

Fitting the replenishment policy alone using SimpleGA resulted in a lower mean daily cost than using a base stock policy with parameters fit using simulation optimization for each of the three heuristic issuing policies. The possible combinations of order-up-to parameters for the base stock policy were evaluated exhaustively, and therefore the performance improvement given by the neural network replenishment policies must be due to an alternative functional form for the learned policy.

Jointly fitting replenishment and issuing policies with SimpleGA gave the lowest mean daily cost. The mean paired-sample percentage difference in daily cost between jointly fitting both policies using SimpleGA and the next best alternative (fitting the replenishment policy using SimpleGA and using the oldest compatible match heuristic issuing policy) was 0.58 $\pm$ 0.00 %. The mean paired-sample percentage difference between jointly fitting both policies using SimpleGA and the best approach with heuristic replenishment and issuing policies (using the priority match heuristic issuing policy) was 3.28 $\pm$ 0.01 %. The results from this scenario therefore show that, at least in some cases, jointly optimizing the replenishment and issuing policies can give better performance than optimizing the replenishment policy alone.

I performed supplementary analysis to investigate the reasons for the observed improvement. First I evaluated the performance of the replenishment policy given

by joint optimization when paired with each of the three heuristic issuing policies. These results were compared to the performance given by replenishment policies fit with the corresponding issuing policy using SimpleGA, as reported in Table 6.5.

Second, I evaluated the performance of the issuing policy given by joint optimization when paired each with the three replenishment policies obtained when fitting the replenishment policy using SimpleGA and a heuristic issuing policy. These results were compared to the performance given by the issuing policies with which each replenishment policy was fit, as reported in Table 6.5.

The performance of each combination of policies was evaluated on the 10,000 evaluation episodes and the results of these comparisons are set out in Table 6.6.

There was no benefit to substituting the jointly fit replenishment policy for any of the replenishment policies fit with a heuristic issuing policy. This is consistent with my expectations and suggests that SimpleGA is fitting good replenishment policies for each heuristic issuing policy. However, the issuing policy obtained when jointly fitting both policies gives a better performance than the priority or oldest compatible match policies when used with a replenishment policy fit under those issuing policies. The improvement from jointly fitting policies can therefore be attributed to a better issuing policy. For this problem a neural network issuing policy could achieve lower costs than a heuristic issuing policy, and it was identified by jointly fitting the replenishment and issuing policies using SimpleGA.

Supervised pretraining was not conducted for the two product scenario because policies fit using SimpleGA performed well when trained from a random initialization and, unlike for the single product scenario, the optimal policy was not available to quantity the remaining level of improvement available. It may be possible to achieve further reductions in daily cost using supervised pretraining.

6.6.2.2 Sensitivity analyses

The results for the sensitivity analyses are set out in Table 6.7. There is a clear benefit to using SimpleGA to fit the replenishment policy over a wide range of settings: in every case the lowest mean daily cost given by a replenishment policy fit using SimpleGA is less than the lowest mean daily cost given by a replenishment

	Replenishment policy		
Issuing policy	Simulation optimization	SimpleGA	Paired-sample difference (%)
Exact	17.41	16.97	-2.51 $\pm$ 0.01
Priority	17.07	16.63	-2.58 $\pm$ 0.01
Oldest compatible	17.09	16.61	-2.81 $\pm$ 0.01
SimpleGA	—	**16.51**	—

The daily cost was calculated for each episode and the mean over the 10,000 evaluation episodes is reported. The percentage difference in daily cost when using a specific issuing policy was calculated between the heuristic replenishment policy and replenishment policy fit using SimpleGA for each episode, and the reported paired-sample difference is the mean ($\pm$ the standard error of the mean) percentage difference over the 10,000 evaluation episodes. The lowest mean daily cost is indicated in **bold**.

Table 6.5: Mean daily cost and paired-sample percentage difference in mean daily cost when fitting replenishment policies with three heuristic issuing policies, and jointly fitting the replenishment and issuing policies using SimpleGA, for the two product scenario.

Performance baseline		Substitute in jointly fit replenishment policy		Susbtitute in jointly fit issuing policy	
	Mean daily cost	Mean daily cost	Paired-sample difference (%)	Mean daily cost	Paired-sample difference (%)
SimpleGA replenishment policy fit with exact match	**16.97**	17.72	4.47 $\pm$ 0.02	17.06	0.53 $\pm$ 0.01
SimpleGA replenishment policy fit with priority match	16.63	16.64	0.05 $\pm$ 0.00	**16.51**	-0.74 $\pm$ 0.00
SimpleGA replenishment policy fit with oldest compatible match	16.61	16.61	0.02 $\pm$ 0.00	**16.51**	-0.60 $\pm$ 0.00

The daily cost was calculated for each episode and the mean over the 10,000 evaluation episodes is reported. The percentage difference in daily cost was calculated between the combination of policies indicated by the row label and that indicated by the column label, and the reported paired sample difference is the mean ($\pm$ the standard error of the mean) percentage difference over the 10,000 evaluation episodes. The lowest mean daily cost per row is indicated in **bold**.

Table 6.6: Mean daily cost and paired-sample percentage difference in mean daily cost when substituting the jointly fit replenishment or issuing policy into base case experiments from Table 6.5

policy fit using simulation optimization. The size of the benefit can depend on the value of input parameters, for example the cost savings from the neural network increase with higher lead time L, but are reduce as mean daily demand increases.

In seven of the thirty experiments jointly optimizing the replenishment and issuing policies using SimpleGA gave the lowest (or joint-lowest) mean daily cost. In the case where 75% of the demand is for product A and 25% of the demand is for product B, jointly fitting the issuing and replenishment policies using SimpleGA gave a mean paired-sample cost reduction of 1.64 $\pm$ 0.00 % compared to the best result fitting the replenishment policy alone using SimpleGA. However, in three of the experiments, jointly optimizing the replenishment and issuing policies using SimpleGA gave a higher mean daily cost than the best result achieved using heuristic replenishment and issuing policies. Additional results for the sensitivity

analysis, including the mean daily cost for each heuristic issuing policy when fitting the replenishment policy using simulation optimization and SimpleGA, are set out in Table L.1 in Appendix L.

This analysis shows that SimpleGA, with a small neural network and single set of hyperparameters, can be used to jointly fit effective replenishment and issuing policies over a wide range of different inputs and, in some cases, give better results than fitting the replenishment policy alone. These results represent a floor on the performance of the policies fit using SimpleGA and it may be possible to achieve better results for a given setting by tuning the hyperparameters.

6.6.2.3 Estimating the Pareto frontier for service level and wastage

The estimated Pareto frontiers are plotted in Figures 6.7, 6.8 and 6.9 for the three different levels of minimum exact match percentage respectively. As in the single product case, the points below the dotted line connecting the solutions for a given approach are Pareto-dominated by a solution from that approach. The hypervolume indicators for the Pareto frontiers estimated using each approach are presented in Table 6.8.

As in single product scenario, jointly fitting policies with the outer-loop approach was an effective way to estimate Pareto frontier, and gave the largest hypervolume indicator over all three settings of minimum exact match percentage. In line with my expectations, the hypervolume indicator for each approach (except for a heuristic replenishment policy with an exact match issuing policy) was lower as the minimum exact match percentage increased, because satisfying the constraint required sacrificing some performance in terms of service level and/or wastage.

NSGA-II performed well when the exact match percentage was 50%, giving a similar hypervolume to the outer-loop approach and better than any of the three approaches with heuristic policies. However, it was outperformed by at least one heuristic approach for both of the other settings. As the minimum exact match percentage increases, it appears to be more difficult for NSGA-II to find solutions with low wastage. An advantage of NSGA-II was that all of the solutions were generated in a single optimization run, which took approximately 5 minutes,

	Simulation optimization and heuristic issuing	**SimpleGA and heuristic issuing**	**SimpleGA jointly fit policies**	**Paired-sample difference** (%)	
	Mean daily cost	Mean daily cost	Mean daily cost	Best SimpleGA - SimOpt and heuristic issuing	SimpleGA jointly fit - SimpleGA and heuristic issuing
Base setting					
	17.07	**16.72**	16.72	-2.05 ± 0.01	0.02 ± 0.00
Lead time L					
0	16.44	16.39	**16.35**	-0.54 ± 0.01	-0.29 ± 0.00
1	17.07	**16.72**	16.72	-2.05 ± 0.01	0.02 ± 0.00
2	17.34	**16.69**	16.72	-3.71 ± 0.01	0.17 ± 0.00
Maximum useful life m					
2	17.07	**16.72**	16.72	-2.05 ± 0.01	0.02 ± 0.00
3	16.17	**16.12**	16.16	-0.27 ± 0.01	0.22 ± 0.01
4	15.88	**15.82**	15.98	-0.37 ± 0.01	0.97 ± 0.01
5	15.79	**15.76**	15.83	-0.21 ± 0.00	0.45 ± 0.01
Product probabilities [A:B]					
[0.10, 0.90]	16.08	**15.84**	**15.84**	-1.50 ± 0.01	0.00 ± 0.00
[0.25, 0.75]	16.80	**16.45**	16.47	-2.06 ± 0.01	0.13 ± 0.01
[0.50, 0.50]	17.07	**16.72**	16.72	-2.05 ± 0.01	0.02 ± 0.00
[0.75, 0.25]	16.81	16.60	**16.33**	-2.87 ± 0.01	-1.64 ± 0.00
[0.90, 0.10]	16.00	**15.75**	15.76	-1.57 ± 0.01	0.08 ± 0.00
Shortage cost C_s : wastage cost C_w					
5:7	17.07	**16.72**	16.72	-2.05 ± 0.01	0.02 ± 0.00
7:7	19.24	**18.73**	18.88	-2.65 ± 0.01	0.83 ± 0.01
14:7	23.16	**22.37**	22.81	-3.40 ± 0.01	2.00 ± 0.01
21:7	25.30	**24.50**	24.51	-3.16 ± 0.02	0.04 ± 0.01
28:7	26.79	**25.81**	26.08	-3.62 ± 0.02	1.07 ± 0.01
35:7	27.91	**26.89**	27.23	-3.64 ± 0.02	1.28 ± 0.02
5:5	16.92	**16.51**	16.58	-2.38 ± 0.01	0.44 ± 0.00
5:10	17.28	**16.84**	17.00	-2.57 ± 0.01	0.98 ± 0.01
5:15	17.54	17.15	**17.12**	-2.42 ± 0.01	-0.21 ± 0.01
5:20	17.77	**17.27**	17.39	-2.77 ± 0.01	0.66 ± 0.01
5:25	17.95	17.62	**17.61**	-1.92 ± 0.01	-0.06 ± 0.01
Substitution cost $C_m^{b,a}$					
0.50	16.08	**15.89**	**15.89**	-1.15 ± 0.01	0.00 ± 0.00
1.0	16.51	**16.16**	16.17	-2.13 ± 0.01	0.11 ± 0.01
2.0	17.07	**16.72**	16.72	-2.05 ± 0.01	0.02 ± 0.00
4.0	17.41	**16.97**	17.04	-2.51 ± 0.01	0.44 ± 0.00
Mean daily demand μ					
4	17.07	**16.72**	16.72	-2.05 ± 0.01	0.02 ± 0.00
8	30.60	30.28	**30.18**	-1.39 ± 0.00	-0.36 ± 0.00
16	56.41	**56.14**	56.31	-0.47 ± 0.00	0.30 ± 0.00
32	107.10	**107.08**	107.90	-0.01 ± 0.00	0.77 ± 0.00

The daily cost was calculated for each episode and the mean over the 10,000 evaluation episodes is reported. When using a heuristic issuing policy, only the best (with the lowest mean daily cost over the evaluation episodes) is reported - See Table K.8 for results foe each heuristic issuing policy. "Best SimpleGA" for each row is the combination of policies that gave the lowest mean daily cost given when fitting the replenishment policy using SimpleGA (with either a heuristic issuing policy or jointly fitting a neural network issuing policy), and is indicated in **bold**. Paired-sample differences were calculated between two pairs of policies. In each case, the percentage difference in daily cost was calculated between the combinations of policies and the reported paired-sample difference is the mean (± the standard error of the mean) percentage difference over the 10,000 evaluation episodes. In the base setting $L = 1, m = 2$, product probabilities=[0.50, 0.50], $C_s = 7$, $C_w = 5$, $C_m^{b,a} = 2$, $\mu = 4$. The base setting row is repeated for each input so that it can be considered in the context of the other levels of the input. The mean daily cost for the base setting is higher here than reported in Table 6.5 because a single set of hyperparameters was used for all SimpleGA experiments in this table, instead of being tuned for each experiment. In cases where the mean daily cost given by jointly optimizing both policies with SimpleGA is greater than the lowest cost given by fitting a replenishment policy with simulation optimization, the result is written in red.

Table 6.7: Mean daily cost and paired-sample percentage difference in mean daily costs for the sensitivity analyses on the two product scenario.

compared to the outer-loop approach which required separate runs for each level of the exact match percentage, each of which took approximately 2.5 hours. It may be possible to achieve better results from NSGA-II for a given minimum exact match percentage by imposing that target as a constraint and optimizing only for service level and wastage, instead of optimizing for all three objectives at the same time.

A single set of hyperparameters was used to run the outer-loop approach for each of the three settings and it may be possible to further improve the performance by tuning the hyperparameters for specific settings.

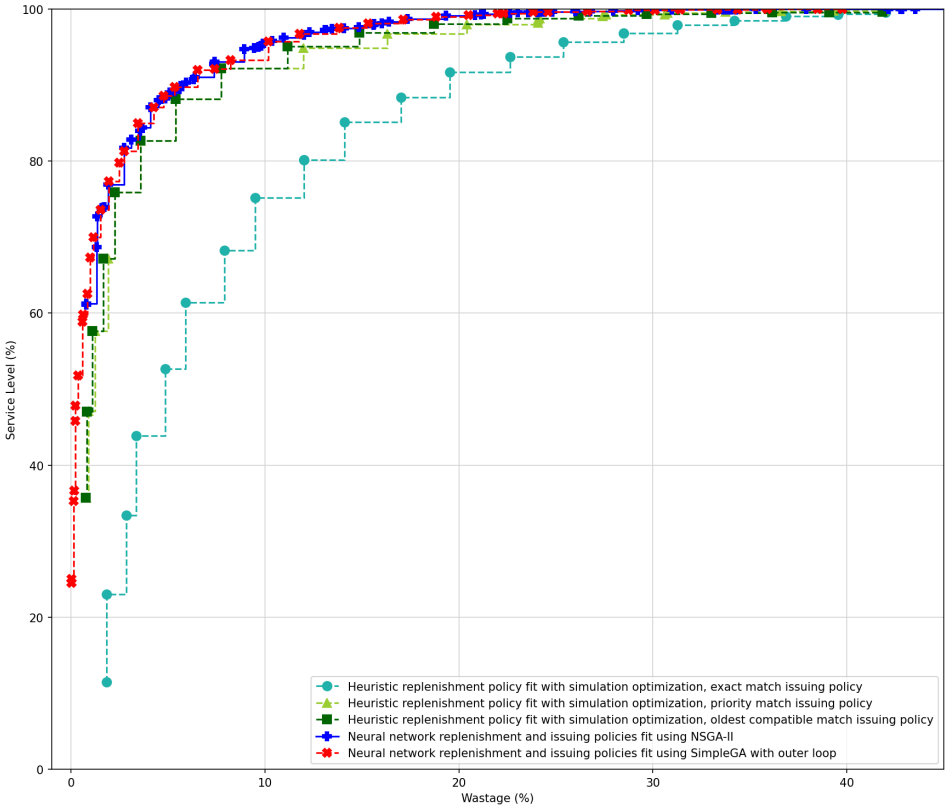


Figure 6.7: Estimated Pareto frontiers for the two product scenario with a minimum exact match percentage of 50%

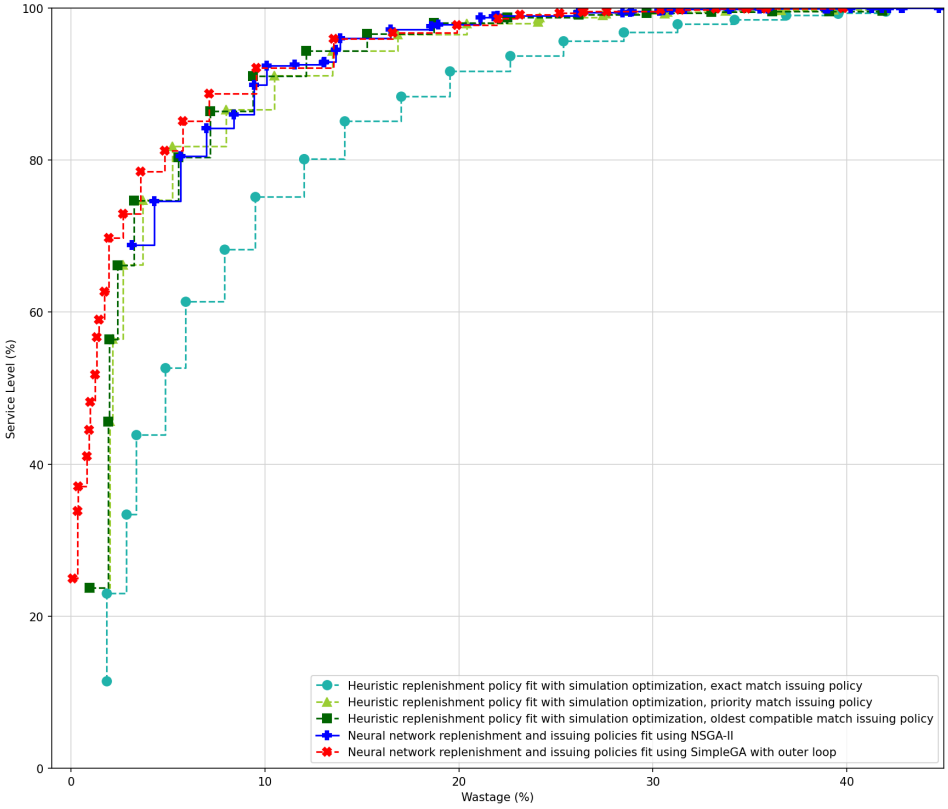


Figure 6.8: Estimated Pareto frontiers for the two product scenario with a minimum exact match percentage of 75%

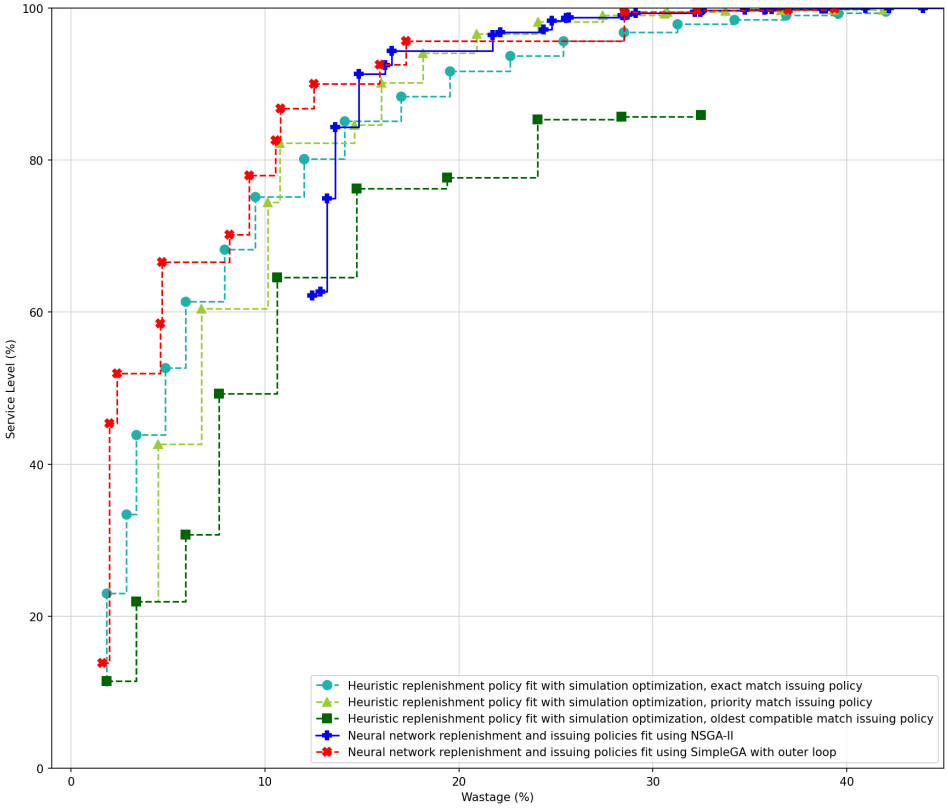


Figure 6.9: Estimated Pareto frontiers for the two product scenario with a minimum exact match percentage of 95%

	Simulation optimization			Outer-loop	NSGA-II
Minimum exact match (%)	Exact	Priority	Oldest compatible		
50	0.819	0.929	0.931	0.955	0.947
75	0.819	0.902	0.908	0.930	0.886
95	0.819	0.814	0.679	0.861	0.697

The service level, wastage and exact match percentage were calculated per episode. The hypervolume indicator for each method was calculated using the mean service level and wastage over 10,000 evaluation episodes for each solution, after filtering the solutions to include only those where the mean exact match percentage exceeded the target minimum exact match percentage.

Table 6.8: Hypervolume indicator for the estimated Pareto frontiers for the two product scenario.

6.7 Eight product scenario

6.7.1 Scenario-specific methods

6.7.1.1 Scenario description

I consider now an eight product, single-echelon, periodic review perishable inventory problem. It is a direct extension of the two product case, specifically

	Day of the week						
	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Low demand	4.7	4.7	4.9	4.7	5.1	3.4	3.6
High demand	37.5	37.3	39.2	37.8	40.5	27.2	28.4

The high demand case was reported by Rajendran and Srinivas [45] and was used in Chapter 3.

Table 6.9: Mean of the day-of-the week specific Poisson distributions for daily demand in the eight product scenario.

representing platelet units with the eight different ABO/RhD blood types. Like the two product scenario, the eight product scenario was also implemented using an adapted single-agent RL environment. Aside from the increased number of products, the key differences compared to the two product scenario are to bring it into line with the platelet replenishment scenario from Rajendran and Srinivas [45] adopted in Chapter 3. The maximum useful life $m = 3$, the lead time $L = 0$, the state and observation include the day of the week, the demand is generated by day-of-the-week specific Poisson arrival processes and the parameters for the demand distributions and the cost components (except for the substitution costs) are taken from Rajendran and Srinivas [45]. Unlike the single and two product scenarios, the stock does not all arrive fresh. This scenario extends the problem described by Rajendran and Srinivas [45] by accounting for recipient and donor ABO/RhD blood types. As in the other scenarios in this chapter, but unlike the scenario from Rajendran and Srinivas [45], the replenishment order is assumed to be placed at the start, instead of the end, of the day.

The cost components, except for the substitution costs, are set out in Table 3.1. I considered two cases for daily demand: the values from Rajendran and Srinivas [45], which I used in Chapter 3 and set out in Table 3.1, are the "high demand" case, and those values divided by eight (to maintain the weekly pattern, while giving demand similar to the single and two product scenarios in which the mean daily demand was four units) is the "low demand case". The parameters of the day-of-the-week specific demand distributions for the two cases are set out in Table 6.9.

The maximum order quantity for each product was set to 60 in the high demand

Recipient ABO/RhD blood type							
O-	O+	A-	A+	B-	B+	AB-	AB+
0.07	0.38	0.06	0.34	0.02	0.09	0.01	0.03

Table 6.10: Parameters of the categorical distribution for the blood type of the recipient reported by Ensafian *et al.* [64] and used in the eight product scenario.

case, as in Chapter 3. The maximum order quantity for the low demand case was set to 15, which has the same cumulative probability under a Poisson distribution with the mean daily demand for the low demand case as a quantity of 60 has for the high demand case.

The parameters of the categorical distribution from which the recipient's blood type is sampled for each unit of demand are set out in Table 6.10 [64]. These figures were collected in Iran, but are similar to those observed at UCLH (see Table B.7 in Appendix B).

The substitution preferences for each recipient blood type are set out in Table 6.11. These preferences were recently used by Yu Suen *et al.* [294] and are based on the guidance of the Irish Health Service Executive [311]. An impermissible choice is represented in the table using the symbol '$\times$' (set to 1e10 for implementation purposes). The substitution cost $C_m^{i,j}$ for issuing a unit of type j to meet demand for type i is obtained by multiplying the values of Table 6.11 by $\frac{C_m^{\max}}{8}$, where $C_m^{\max}$ is the maximum substitution cost. Following the approach of Meneses *et al.* [55] and Dillon *et al.* [36] $C_m^{\max}$ was set equal to the shortage cost, $C_m^{\max} = C_s = 3,250$. Under this structure, there is no penalty for an exact match, the first preference substitution incurs a cost of $\frac{C_m^{\max}}{8}$, the second preference a substitution cost of $\frac{2C_m^{\max}}{8}$ and so on, such that the immediate cost of a substitution is always lower than a shortage. I also considered an alternative, where the maximum substitution cost was set equal to the wastage cost, $C_m^{\max} = C_w = 650$.

The remaining useful life of platelet units on arrival was sampled from a multinomial distribution, as in Chapter 5: 50% of the stock arrives fresh, 20% with two days of useful life, and 30% on the day that it will expire. These parameters, along with a maximum useful life $m = 3$ days, were observed by Rajendran and

Recipient ABO/RhD blood type	Donor ABO/RhD blood type							
	O-	O+	A-	A+	B-	B+	AB-	AB+
O-	0	×	2	×	1	×	×	×
O+	1	0	5	4	3	2	×	×
A-	3	×	0	4	2	×	1	5
A+	×	×	1	0	5	4	3	2
B-	3	×	2	×	0	4	1	5
B+	×	×	5	4	1	0	3	2
AB-	3	×	1	5	2	×	0	4
AB+	×	×	3	2	5	4	1	0

Priority order reported by Yu Suen *et al.* [294]. Smaller numbers represent a higher priority, with 0 for an exact match and × indicating a match that is not permitted.

Table 6.11: Order of substitution preferences for platelets in the eight product scenario.

Ravindran [54] at a regional medical centre in Pennsylvania, USA and considered as an alternative scenario by Rajendran and Srinivas [45]. In Chapter 5, when I used this distribution of remaining useful life on arrival in the context of returns, the setting led to high wastage. I adopted it here because the comparatively short useful life and high proportion of older stock make it a challenging scenario where taking account of the age profile of the stock and the stock-levels of compatible units may be important, giving scope for better performance by neural network policies.

The reward function for the replenishment agent is similar to the two product case (Equation 6.11), but includes a term for a fixed ordering cost:

$$R_{t+1} = -C_f \mathbb{1}_{\sum_p A_t^{r,p}>0} - C_v A_t^r - C_h B_t - C_s E_t - C_w W_t - \sum_{i,j} C_m^{i,j} M_t^{i,j} \quad (6.14)$$

The initial day of the week at the start of the warm-up period is sampled from a uniform distribution. In the observation input to the neural network policies the day of the week (for both policies) and the type of the requested product (for the issuing policy only) were represented as one-hot vectors.

6.7.1.2 Replenishment policies

The heuristic replenishment policy is a multi-product base stock policy, with an order-up-to level `S` for each of the eight products as described above in Section 6.4.4.1 The same parameter is used for each product on each weekday, unlike in

Chapter 3 in which there were parameters for each weekday, because it would require 56 parameters to have a separate parameter for each weekday and each product and optimizing such policies was challenging in preliminary experiments.

The mapped output of the neural network policy was used directly as the order quantity, as described in Section 6.4.4.2. For this scenario I only considered direct action replenishment policy neural networks. During preliminary experiments, it was challenging to fit neural network replenishment policies from scratch that matched the performance of the heuristic policies and therefore I focused on investigating the potential benefits of supervised pretraining. During work on the single product scenario I found that supervised pretraining only had a clear benefit when using direct action replenishment policy neural networks (see Section 6.5.2.2).

6.7.1.3 Issuing policies

The issuing policies are the same as those used for the two product case, described above in Section 6.4.5.1. The substitution preferences required for the priority match and oldest compatible match policies are set out in Table 6.11.

Neural network issuing policies output the blood type of the unit to be issued, or opt not to issue a unit, as described above in Section 6.4.5.2.

6.7.1.4 Fitting policies to optimize a single objective

The parameters of the multi-product base stock policies were fit using simulation optimization for each of the three heuristic policies in turn, following the approach adopted in Chapter 4. The parameter space is large because there are eight different products and therefore, as in Chapters 4 and 5, the order-up-to parameter space was searched using Optuna's NSGA-II sampler. Each proposed parameter combination was evaluated on 400 episodes and the combination that gave the highest return was selected for subsequent evaluation. For the high demand case, the parameter search space was $\mathtt{S}^p \in \{0, 1, ..., A^{r,p}_{\max} = 60\}$ and for the low demand case the parameter space was $\mathtt{S}^p \in \{0, 1, ..., A^{r,p}_{\max} = 15\}$, in both cases for all $p \in \{$O-, O+, A-, A+, B-, B+, AB-, AB+$\}$.

For each of the four scenario settings (high or low demand and maximum

substitution cost equal to the wastage or shortage cost), I ran the three combinations of base stock policy and heuristic issuing policy, with order-up-to parameters identified using the simulation optimization procedure, on 1,000 validation episodes and selected the combination that gave the highest mean return. This pair of heuristic policies was used for supervised pretraining. See Appendix J.3.2 for a full description of the supervised pretraining procedure for both replenishment and issuing policies.

For each of the four scenario settings, I conducted eight experiments fitting neural network policies using SimpleGA: fitting only replenishment (with or without supervised pretraining), fitting only issuing (with or without pretraining), and jointly fitting both policies under four conditions (no pretraining, pretraining only replenishment, pretraining only issuing, and pretraining both policies). When fitting only one policy, the other policy was set as the best-performing heuristic policy selected as the target for supervised pretraining.

I ran a hyperparameter tuning sweep for 8 hours for each of the experiments in which policies were fit with SimpleGA. The hyperparameter search ranges are set out in Table K.13 in Appendix K.

For computational reasons, training episodes for the experiments with full demand were reduced to 90 days long, following a shorter warm-up period of 25 days. The validation and evaluation episodes remained 365 days long following a 100 day warm-up period.

6.7.2 Results

The results for the eight product scenario are set out in Tables 6.12 to 6.15: one for each scenario setting with different levels of demand and a different maximum substitution costs. The best results for each scenario are summarized in Table 6.16. The expected daily cost for a policy with perfect foresight, which always ordered exactly enough stock to meet the demand each day with an exact match, is 23,252 for the high demand case and 3,113 for the low demand case. Both would have a service level of 100%, 0% wastage, 100% exact match and no holding.

This scenario appears to be more challenging than the single and two product

scenarios: there are no cases for the eight product scenario where fitting neural network policies without supervised pretraining gave a lower mean daily cost than the best combination of heuristic policies. However, the results clearly demonstrate a benefit to supervised pretraining: for each scenario setting, the lowest mean daily cost was given by the experiment fitting a neural network replenishment policy using SimpleGA, initialized using supervised pretraining, and a heuristic issuing policy. Additionally, in all four scenario settings the second lowest mean daily cost was given by the experiment jointly fitting neural network replenishment and issuing policies, with both initialized using supervised pretraining. There are only two cases where supervised pretraining leads to a higher mean daily cost than in a comparable experiment without supervised pretraining: in both the low and high demand case when the maximum substitution cost is equal to the wastage cost, jointly fitting both policies with no supervised pretraining gives a lower mean daily cost than jointly fitting both policies with supervised pretraining for the issuing policy alone.

Unlike in the two product scenario, there does not appear to be a benefit to jointly optimizing the replenishment and issuing policies. However, the computational resources used for hyperparameter tuning were determined by number of hours (instead of number of runs). Experiments where the issuing policy is a neural network took longer to run and therefore fewer runs were completed in the sweeps for experiments where the issuing policy was found using a neural network (approximately 60 runs compared to 150 runs when only the replenishment policy was a neural network). If sweep duration had been set based on the same number of runs, instead of the same number of hours, it is possible that better hyperparameters may have been identified for the experiments where the issuing policy is a neural network.

In the one and two product scenarios, I was able to exhaustively evaluate each possible parameterization for the base stock heuristic replenishment policy and therefore it was clear that, when a neural network policy gave better performance, it was due to learning some more complicated function rather than representing

the same policy with better parameters. In this scenario, as in Chapters 4 and 5 when the heuristic policy parameter space was large, the parameters were fit using NSGA-II. In Figures L.1 to L.4 in Appendix L, I plot the order quantity for each blood type against the total stock holding for that blood type over 5,000 steps in the adapted single-agent environment for the best-performing replenishment policy fit using SimpleGA. It is clear from these figures that the best-performing neural network policies are not following the same base stock policy structure with a different set of parameters because different order quantities are observed for the same total stock-holding. The neural network policies therefore appear to be taking account of additional information: one or more of the weekday, the age profile of the stock, and the age profile and stock levels of other products.

The benefits on offer from neural network replenishment policies, summarized in Table 6.16, appear to be relatively limited for the eight product scenario, particularly compared to the two product scenario where the mean cost saving was over 3% for multiple experimental settings. Therefore, for a platelet replenishment problem with realistic inputs, including all eight ABO/RhD blood types and their compatibility relationships, fitting the parameters of a simple, interpretable, heuristic replenishment policy using simulation optimization gives competitive performance with neural network policies fit with SimpleGA.

The KPIs for the best policies in each setting do not reach the ideal values of 0% wastage, 100% service level, or 100% exact match percentage. Without knowing the optimal policy for the eight-product scenario, the room for improvement is unclear. It may therefore be possible to improve the performance of neural network policies through more extensive hyperparameter tuning, additional feature engineering, by using alternative algorithms to fit the parameters. It may also be possible to improve the results for the heuristic replenishment policy by including additional parameters (e.g. day of the week specific values of `S` for each blood type), or adopting more complicated heuristics.

Replenishment policy		Issuing policy		Daily cost	Service level (%)	Wastage (%)	Exact match (%)	Holding
Type	Pretrained	Type	Pretrained	*Mean*	*Mean*	*Mean*	*Mean*	*Mean*
Simulation optimization		Exact		7,314	85.4	29.2	100.0	3.6
Simulation optimization		Priority		5,116	95.6	14.3	49.0	2.0
Simulation optimization*		Oldest compatible*		**4,903**	95.5	6.6	29.8	2.5
SimpleGA		Oldest compatible		4,993	96.1	7.4	23.3	2.6
SimpleGA	✓	Oldest compatible		**4,827**	96.8	7.0	29.9	2.7
Base stock		SimpleGA		4,949	95.9	8.7	33.2	2.3
Base stock		SimpleGA	✓	4,882	95.8	6.5	31.0	2.5
SimpleGA		SimpleGA		5,034	95.9	8.9	31.7	2.4
SimpleGA	✓	SimpleGA		4,955	96.5	7.3	11.3	2.5
SimpleGA		SimpleGA	✓	5,271	95.4	6.3	6.1	2.4
SimpleGA	✓	SimpleGA	✓	4,859	96.8	7.2	29.8	2.8

The daily cost and KPIs were calculated for each episode and the mean over the 10,000 evaluation episodes is reported. The rows with the lowest mean daily cost when the replenishment policy was fit using simulation optimization, and when at least one policy was fit using SimpleGA, are each indicated in **bold**. The lowest overall mean daily cost is **underlined**. The pair of heuristic policies giving the lowest mean daily cost on the validation episodes and therefore used for pretraining SimpleGA policies, and as the other policy when only one policy was fit, are indicated by *. A ✓indicates that the neural network parameters were initialized based on supervised pretraining using the corresponding policy marked with a *.

Table 6.12: Mean daily cost and KPIs for the eight product scenario with low demand and the maximum substitution cost equal to the wastage cost.

Replenishment policy		Issuing policy		Daily cost	Service level (%)	Wastage (%)	Exact match (%)	Holding
Type	Pretrained	Type	Pretrained	*Mean*	*Mean*	*Mean*	*Mean*	*Mean*
Simulation optimization		Exact		7,314	85.4	29.2	100.0	3.6
Simulation optimization*		Priority*		**5,974**	97.5	19.3	74.2	2.6
Simulation optimization		Oldest compatible		6,768	96.9	10.9	50.8	3.2
SimpleGA		Priority		6,113	96.2	17.2	69.8	2.8
SimpleGA	✓	Priority		**5,925**	97.0	18.2	73.4	2.4
Base stock		SimpleGA		6,028	97.4	18.1	72.5	2.7
Base stock		SimpleGA	✓	5,974	97.5	19.3	74.2	2.6
SimpleGA		SimpleGA		6,565	95.1	14.6	64.8	2.8
SimpleGA	✓	SimpleGA		6,229	94.5	14.5	68.9	2.2
SimpleGA		SimpleGA	✓	6,207	95.9	16.6	68.2	2.7
SimpleGA	✓	SimpleGA	✓	5,925	96.7	17.4	72.7	2.4

The daily cost and KPIs were calculated for each episode and the mean over the 10,000 evaluation episodes is reported. The rows with the lowest mean daily cost when the replenishment policy was fit using simulation optimization, and when at least one policy was fit using SimpleGA, are each indicated in **bold**. The lowest overall mean daily cost is **underlined**. The pair of heuristic policies giving the lowest mean daily cost on the validation episodes and therefore used for pretraining SimpleGA policies, and as the other policy when only one policy was fit, are indicated by *. A ✓indicates that the neural network parameters were initialized based on supervised pretraining using the corresponding policy marked with a *.

Table 6.13: Mean daily cost and KPIs for the eight product scenario with low demand and the maximum substitution cost equal to the shortage cost.

Replenishment policy		Issuing policy		Daily cost	Service level (%)	Wastage (%)	Exact match (%)	Holding
Type	Pretrained	Type	Pretrained	*Mean*	*Mean*	*Mean*	*Mean*	*Mean*
Simulation optimization		Exact		31,025	96.9	5.2	100.0	18.8
Simulation optimization*		Priority*		**26,966**	99.3	1.9	85.8	12.1
Simulation optimization		Oldest compatible		28,606	99.1	1.3	64.2	12.4
SimpleGA		Priority		28,387	99.7	2.9	72.3	16.7
SimpleGA	✓	Priority		**26,839**	99.3	1.4	82.6	11.3
Base stock		SimpleGA		27,485	99.3	0.8	73.5	12.5
Base stock		SimpleGA	✓	26,947	99.4	1.9	85.6	12.1
SimpleGA		SimpleGA		29,327	99.1	1.4	60.7	16.1
SimpleGA	✓	SimpleGA		28,544	99.4	0.6	67.6	13.2
SimpleGA		SimpleGA	✓	31,090	99.5	0.2	29.0	13.2
SimpleGA	✓	SimpleGA	✓	26,848	99.4	1.5	82.0	11.6

The daily cost and KPIs were calculated for each episode and the mean over the 10,000 evaluation episodes is reported. The rows with the lowest mean daily cost when the replenishment policy was fit using simulation optimization, and when at least one policy was fit using SimpleGA, are each indicated in **bold**. The lowest overall mean daily cost is **underlined**. The pair of heuristic policies giving the lowest mean daily cost on the validation episodes and therefore used for pretraining SimpleGA policies, and as the other policy when only one policy was fit, are indicated by *. A ✓indicates that the neural network parameters were initialized based on supervised pretraining using the corresponding policy marked with a *.

Table 6.14: Mean daily cost and KPIs for the eight product scenario with high demand and the maximum substitution cost equal to the wastage cost.

Replenishment policy		Issuing policy		Daily cost	Service level (%)	Wastage (%)	Exact match (%)	Holding
Type	Pretrained	Type	Pretrained	*Mean*	*Mean*	*Mean*	*Mean*	*Mean*
Simulation optimization		Exact		31,025	96.9	5.2	100.0	18.8
Simulation optimization*		Priority*		**28,555**	99.6	3.2	93.1	14.6
Simulation optimization		Oldest compatible		36,637	98.6	1.6	71.7	14.5
SimpleGA		Priority		30,970	99.7	5.0	91.5	19.7
SimpleGA	✓	Priority		**28,524**	99.6	3.1	93.0	14.3
Base stock		SimpleGA		32,307	99.5	3.1	83.9	14.6
Base stock		SimpleGA	✓	28,560	99.6	3.2	93.1	14.6
SimpleGA		SimpleGA		40,145	99.2	5.0	76.8	24.6
SimpleGA	✓	SimpleGA		36,704	99.0	0.8	78.2	15.9
SimpleGA		SimpleGA	✓	35,412	99.1	5.1	75.9	23.7
SimpleGA	✓	SimpleGA	✓	28,544	99.6	3.0	93.0	14.3

The daily cost and KPIs were calculated for each episode and the mean over the 10,000 evaluation episodes is reported. The rows with the lowest mean daily cost when the replenishment policy was fit using simulation optimization, and when at least one policy was fit using SimpleGA, are each indicated in **bold**. The lowest overall mean daily cost is **underlined**. The pair of heuristic policies giving the lowest mean daily cost on the validation episodes and therefore used for pretraining SimpleGA policies, and as the other policy when only one policy was fit, are indicated by *. A ✓indicates that the neural network parameters were initialized based on supervised pretraining using the corresponding policy marked with a *.

Table 6.15: Mean daily cost and KPIs for the eight product scenario with high demand and the maximum substitution cost equal to the shortage cost.

Demand	$C_m^{\max}$	Best SimOpt	Best SimpleGA	Jointly fit from pretrained SimpleGA	Paired-sample differences (%)	
					Best SimpleGA - best SimOpt	Jointly fit from pretrained SimpleGA - best SimpleGA
Low	650	4,903	4,827	4,859	-1.53 $\pm$ 0.01	0.67 $\pm$ 0.01
Low	3,250	5,974	5,925	5,925	-0.81 $\pm$ 0.00	0.00 $\pm$ 0.01
High	650	26,966	26,839	26,848	-0.47 $\pm$ 0.00	0.04 $\pm$ 0.00
High	3,250	28,555	28,524	28,544	-0.11 $\pm$ 0.00	0.07 $\pm$ 0.00

Mean daily cost values from Tables 6.12 to 6.15. Paired-sample differences were calculated between two pairs of policies. In each case, the percentage difference in daily cost was calculated between the combination of policies and the reported paired-sample difference is the mean ($\pm$ the standard error of the mean) percentage difference over the 10,000 evaluation episodes. For each experiment, "Best SimOpt" was the solution with the lowest mean daily cost when fitting the replenishment policy using simulation optimization, "Best SimpleGA" was the solution with the lowest mean daily cost when fitting at least one policy with SimpleGA, and "Jointly fit from pretrained SimpleGA" was the method where both policies were fit using SimpleGA and were initialized using supervised pretraining

Table 6.16: Summary of mean daily costs and paired-sample percentage difference in mean daily costs for the eight product scenario

6.8 Discussion

This chapter is, to the best of my knowledge, the first to frame the problem of platelet inventory management in a hospital blood bank as a multi-agent RL problem, jointly optimizing the replenishment and issuing policies. Using custom RL environments, I developed and evaluated both heuristic and neural network-based replenishment and issuing policies for managing platelet inventory, including all eight ABO/RhD blood types and their substitution relationships. The models were tested with realistic inputs for daily demand, demand per blood type, and the remaining useful life of platelet units upon arrival at the hospital blood bank.

To support this research, I developed Python classes for GPU-accelerated multi-agent RL environments capable of handling multiple agents. This included a fully-fledged multi-agent environment for the single product scenario and an adapted single-agent environment for the two and eight product scenarios. Both of these environments were specifically developed to handle an important, and unusual, feature of the problem: the fact that the replenishment and issuing agents act at different frequencies with a random number of steps between the successive steps by the replenishment agent depending on the daily demand. The adapted single-agent environment offers an efficient interface for applying neuroevolutionary methods to other problems with similar characteristics: where multiple agents share the same reward, and do not take their actions in parallel or

alternately.

Neuroevolutionary methods proved highly effective, with SimpleGA yielding lower costs compared to both heuristic methods and PPO for single product scenarios and lower costs than heuristic policies in the two and eight product scenarios. Incorporating existing knowledge from a heuristic policy was straightforward with SimpleGA. This approach led to improved results for the harder problems, with a longer lead time, in the single product case and was essential to achieving lower costs than the heuristic policies in the eight product case.

There was no disadvantage to jointly fitting the policies with SimpleGA compared to fitting the replenishment policy with the optimal issuing policy in the single product case. There are experimental settings for the two product scenario where jointly optimizing the policies gave lower costs than optimizing only the replenishment policy with the best heuristic issuing policy, and, for the base case of the scenario, the benefit from joint optimization was attributed to an improved issuing policy.

For the eight product case, with realistic inputs for platelet inventory management in a hospital blood bank, there was no benefit to jointly optimizing the replenishment and issuing policies and less benefit from using neural network replenishment policies. However, the multi-agent RL environments provided an efficient way to fit the parameters of the heuristic policies, by supporting the evaluation of multiple candidate policy parameters in parallel on a large number of episodes, and a modular interface that enabled different combinations of policies to be straightforwardly compared.

The results for the eight-product scenario are not sufficient to conclude that heuristic policies will generally be competitive with neural network policies for more realistic and complex problems. Sensitivity analyses for the two-product scenario indicate that the relative performance of the two approaches can vary significantly depending on the problem settings: the benefits of neural network policies become more pronounced as demand decreased and the lead times

increased. This suggests that the effectiveness of each approach may depend on the specific operational context, and further investigation is needed to fully understand their comparative advantages across a broader range of scenarios.

For the single and two product cases, I found that neuroevolutionary methods offered an effective way to estimate the Pareto frontier between wastage and service level. In both cases, an outer-loop approach using SimpleGA to jointly fit the replenishment and issuing policy offered better performance across the range of possible combinations of service level and wastage than heuristic policies.

The main limitation of this chapter is that, except for the simplest case when $m = 2$ and $L = 2$ for the single product scenario, the multi-dimensional observation space and black-box nature of neural networks mean that I have not been able to describe in detail how and when the neural network policies differ from the heuristic policies. Understanding these differences could support the development of new heuristic policies that incorporate the key differences in a transparent way and provide greater confidence to decision makers. I was able to establish, either by exhaustively evaluating the order-up-to parameters or plotting the order quantities against total stock for each product, that the neural network replenishment policies are not alternative parameterizations of the base stock policy and so are taking additional information into account.

Additionally, I did not consider multi-objective optimization for the eight product case. Multi-objective optimization was a relatively straightforward extension of the existing work for the single and two product scenarios, but there are additional challenges to the eight product case. Firstly, due to the larger number of products (and demand, for the high demand case), it was not feasible to exhaustively evaluate the parameters for the base stock policy and an alternative method would be needed for searching the heuristic policy space. Additionally, the results using a cost function to balance priorities showed that it was challenging to fit neural network policies that achieved better performance than the heuristic policies without supervised pretraining. Therefore, in order for it to be worthwhile running the multi-objective optimization for the neural network policies, I would

need to investigate how to perform supervised pretraining from a range of heuristic policies covering trade-offs between the objectives. This additional work is beyond the scope of the chapter.

Another limitation is that I have only considered one, basic, multi-agent RL method (IPPO). The tooling and software libraries in Python for multi-agent RL are much less mature than those for single-agent RL and therefore it is not straightforward to apply standard algorithms to new custom environments, even for CPU-based environments. Additionally these methods require step-level information, and I found that my fully-fledged GPU-accelerated multi-agent RL environments were slow for the problems with more than one product. I therefore chose to focus on the neuroevolutionary methods, specifically SimpleGA, because it had the strongest performance on the single product scenario and, because it learns from performance at the level of episode, could be used with my adapted single-agent environment.

Since the start of the work on this chapter, other researchers have developed JAX-based multi-agent RL environments. A team including the author of gymnax [119], which I used as the basis for environments in this chapter, introduced JaxMARL in November 2023 which includes GPU-accelerated multi-agent environments and compatible implementations of popular multi-agent RL algorithms. Unlike the environments I developed, the environments assume that the agents act in parallel, so each agent is assumed to act at each timestep. Dummy actions can be used if the agents act in turn or at different intervals (as with the scenarios considered in this chapter). This was followed in December 2023 by Mava [312], which provides multi-agent RL algorithms written in JAX that are compatible with JAX-based environments implemented using the Python library Jumanji [313]. A library for creating a more specialized set of environments, modelling games inspired by Multiplayer Online Battle Arena games, has also been released [314]. The availability of more standardized environment classes and compatible implementations of multi-agent RL algorithms will make it more straightforward to compare a variety of methods on new environments, and to

compare new methods on a variety of environments.

6.9 Conclusion

In this chapter, I explored the joint optimization of replenishment and issuing policies for managing perishable inventory, up to and including platelet inventory control across all eight ABO/RhD blood types and their substitution relationships. I developed custom GPU-accelerated RL environments to reflect the decisions of multiple agents, including a fully-fledged multi-agent environment and an adapted single-agent environment, which was more computationally efficient for larger problems. Neural network replenishment policies, optimized using a simple genetic algorithm, consistently incurred lower costs than heuristic benchmark policies. In some cases, jointly optimizing both replenishment and issuing policies resulted in the lowest costs. As interest in DRL algorithms grows in operational research, this chapter demonstrates that neuroevolution offers a compelling alternative. It achieved strong performance, benefited from large-scale parallelization using GPUs, easily incorporated existing knowledge through supervised pretraining to initialize parameters, and was used to estimate the Pareto frontier of solutions under different priorities by directly optimizing KPIs over whole simulated episodes.

Chapter 7

Conclusion

In this chapter I summarize the findings of the thesis, set out the strengths and weaknesses of the work as a whole, and suggest directions for future research.

7.1 Summary of findings

In Chapter 3, I found that DRL methods can be used to learn effective policies for platelet replenishment. The DRL algorithm PPO gave the lowest mean daily costs of two DRL and four heuristic policies, with the parameters for the heuristics fit using stochastic mixed integer linear programming, when using both simulated demand and observed demand trajectories from our partner hospital. For the simulated demand case, on which the optimal policy was computed using Q-value iteration, PPO performed near optimally.

Subsequently, in Chapter 4, I established that advances in GPU hardware and software have increased the size of perishable inventory problems for which it is practical and feasible to find the optimal policy. Using the Python library JAX and consumer-grade GPU hardware, I computed the optimal policy with value iteration for large cases of three scenarios which were recently described as infeasible or impractical in the literature. Additionally, I developed GPU-accelerated simulators which I used to fit well-performing heuristic replenishment policy parameters for each of the scenarios, and for the scenario considered in Chapter 3.

In Chapter 5, I developed an updated model of the platelet inventory management workflow to incorporate returns, platelet units issued to patients but not

transfused, to investigate how hospital blood banks can modify their issuing policies to mitigate wastage caused by this discrepancy between requests and transfusions. Using a simulation-first approach, enabled by the GPU-accelerated simulators developed in Chapter 4, I estimated the potential utility of a novel, ML-guided issuing policy, for different levels of predictive model quality. This analysis showed that the proposed policy, supported by a sufficiently good predictive model, could reduce wastage relative to a standard OUFO policy and I subsequently trained and evaluated a sufficiently good supervised learning model.

Finally, in Chapter 6 I investigated jointly optimizing replenishment and issuing policies for managing multiple perishable products with substitution. I built custom, GPU-accelerated environments to model the task as a multi-agent problem which provide a modular platform for comparing the performance of different replenishment and issuing policies. In some scenarios jointly optimizing the policies gave the lowest costs. There was no clear benefit to neural network policies or jointly optimizing the policies when managing platelets of eight different blood types, but there is not enough evidence to suggest that this holds in general for larger, more realistic problems: sensitivity analyses on a smaller scenario showed the benefits of neural networks policies depend on multiple problem settings including lead time and demand. I explored the advantages of fitting neural network policies using evolutionary algorithms, including the ability to optimize the policies using KPIs directly, and to incorporate knowledge from heuristic policies into the policy search process.

Overall, the results show that ML and GPU-accelerated computing can be used to find improved policies for both platelet replenishment and issuing in a hospital blood bank, supporting decision-making for larger problems that incorporate challenging, previously neglected, aspects of reality.

7.2 Strengths and limitations

I explain the strengths and limitations of individual components of the work in the corresponding chapters. In this section I focus, in turn, on strengths and limitations

of the thesis as a whole.

7.2.1 Strengths

One strength of the work is the engagement with subject matter experts and practitioners - the people whose day-to-day work this research is ultimately intended to support. I spent a day shadowing a Senior Biomedical Scientist in the UCLH transfusion laboratory, have presented and discussed the findings of the work internally at multiple UCLH Hospital Transfusion Team meetings and externally at conferences for blood transfusion and haematology. I also collaborated with the National Institute for Health and Care Research (NIHR) Blood Transfusion Research Unit in Data Driven Transfusion Practice to review current applications of ML applied to transfusion medicine. Chapters 5 and 6 address aspects of the real-world problem identified through discussions with staff at the UCLH transfusion laboratory and they are considering how best to implement an issuing policy based on my proposed ML-guided policy. I have reported the results in terms of KPIs relevant to the practitioners and in Chapter 6 used multi-objective optimization methods to present a range of feasible options in terms of these KPI for perishable inventory management problems.

I have sought to tackle realistic sized problems, using data for daily demand, remaining useful life on arrival, and the distribution of blood types from previously reported work and from UCLH. I invested effort into improving benchmark methods so that they can be applied to larger problems with additional aspects of reality, rather than focusing solely on ML methods. This included developing a GPU-accelerated implementation of value iteration that was able to compute the optimal policy for the realistic setting of the platelet inventory management problem described by Mirjalili [166]. I also developed GPU-accelerated RL environments, simulators for MDPs, and used them to fit heuristic policy parameters efficiently using simulation optimization, achieving better results than fitting parameters for the same policies using stochastic mixed integer linear programming.

Making use of the GPU-accelerated RL environments to simulate a workflow, I adopted a simulation-first approach in Chapter 5. This approach enabled me to

investigate the potential benefits of an ML model despite challenges with access to data and computational resources, and could be widely applicable to gain an initial understanding of the potential real-world benefits on an ML model.

Finally, I have released my experimental code open source on GitHub. Many previous studies in this area model very similar scenarios from scratch, duplicating effort and increasing the likelihood of errors. By sharing the code I hope that other researchers will be able to build on it and that, in the future, it will form part of a reference set of problems enabling different methods to be readily compared, such as OR-Gym [147].

7.2.2 Limitations

A major limitation, compared to the original vision for the project, is the limited use of patient-level data to inform blood product inventory management decisions. This resulted from two related difficulties: access to patient-level data, and the availability of computational resources in a secure research environment. The ethical approval and data access requests were delayed by the COVID-19 pandemic, but I eventually obtained access to data from our partner hospital through a virtual desktop with limited computational resource, and no GPU access. I was granted approval to transfer the data to the UCL Data Safe Haven (DSH), but the GPUs available on the DSH were subsequently taken out of service. As a result, it has only been possible to perform limited ML modelling work on the patient-level data (for example, the XGBoost model to predict returned platelet units in Chapter 5) but not, as was originally planned, to incorporate them into the observations of an RL model. The thesis therefore focuses more heavily on simulated data, with inputs drawn from the literature and UCLH, and I developed the simulation-first approach in Chapter 5 as an alternative to a more common ML model development-first approach which may be a useful alternative for other researchers facing similar challenges.

In Chapters 3, 4 and 5 I balanced competing inventory management objectives, such as service level and wastage, using notional costs. I initially adopted this approach because it was frequently used in the related literature and provides a single scalar cost that could be used as a reward signal for RL. While the same

notional costs have been used in previous work, they do not have an empirical basis and therefore may not reflect the true preferences of practitioners. As noted above, in Chapter 6 I considered how to understand and illustrate the trade-offs between different objectives while retaining the flexibility of a neural network policy,

I used NSGA-II to search the heuristic policy parameter spaces when evaluating every combination of parameters was not possible in Chapters 4, 5 and 6. I selected this method based on preliminary experiments during the work for Chapter 4, in which I found that it was more efficiently able to deal with batch updates (i.e. for multiple combinations of policy parameters tested in parallel) than the competing methods available in the Python library Optuna. However, I did not perform an exhaustive evaluation of alternative search strategies and it may be possible to obtain better performance, both in terms of the policy parameters and the efficiency of searching the space, using an alternative method.

The scenarios considered in this thesis build on recently published scenarios and include elements of the real problem that have previously received relatively little attention, including the fact of returns and placing replenishment orders for platelets taking into account ABO/RhD blood groups. However, I did not consider a scenario that includes both of these elements, and there are other elements of the real problem, such as orders for patients with special transfusion requirements and the possibility of multiple routine replenishment orders per day, that may be important to include to provide useful decision support in real life. My RL environments could be straightforwardly modified to account for multiple routine orders per day, but incorporating patients with special transfusion requirements would be more challenging.

7.3 Future research directions

In light of the research set out in this thesis, I propose future directions for research on using ML methods to support the management of blood product inventory and how the software and approaches I have developed may be applied more broadly.

7.3.1 Blood product inventory management

7.3.1.1 Explaining policies

In Chapters 3, 4 and 6 I have found cases where the optimal replenishment policy computed using value iteration or a policy fit using a neural network gives better performance than a heuristic replenishment policy. The downside of these policies is that they are more complex and, when the observation space is large, difficult to visualize and interpret. The example provided by Quinn *et al.* [90] shows that hospital blood banks may be willing to adopt a system that recommends order quantities, instead of using simple heuristic rules. However, it would be helpful for both researchers and practitioners to have a better understanding of the complex policies. It may help researchers to develop new methods, including new heuristics. For decision makers, it could help to build confidence in the system and awareness about when the recommendations may not be suitable. One method for improving understanding would be to develop new visualizations, which may benefit from interactivity due to the large number of dimensions, that can help users to explore the complex policies, including the impact of different inputs on the recommended order quantities and when and how they differ from well-understood heuristic policies.

7.3.1.2 Multi-objective optimization

In Chapter 6 I used two multi-objective optimization approaches (an outer-loop approach and NSGA-II) to investigate the possible trade-offs between different objectives, instead of balancing the objectives using notional costs. Given the difficulty of estimating the notional costs for wastage and shortage, where the costs are not primarily financial, framing the platelet inventory management problem as a multi-objective approach may help to develop policies that are better suited to the preferences of decision makers. While Blake *et al.* [37] emphasized the potential benefits of this approach over 10 years ago, the majority of work in this area still uses notional costs to balance different priorities. Future work could explore the many alternative evolutionary algorithms for multi-objective optimization, or the

problem could be modelled as a multi-objective MDP [295], with a vector valued reward function. A Python library with a new API for modelling multi-objective MDPs, and a collection of compatible algorithms, was released in 2023 [315], which may help to facilitate work in this direction. These methods may be more effective and efficient at estimating the Pareto frontier between KPIs of interest to decision makers, and support the extension of the multi-objective optimization work to the inventory management of all eight ABO/RhD blood types.

7.3.1.3 Improved prediction of transfusion based on a request for platelets

In Chapter 5 I developed the first, to the best of my knowledge, supervised learning model to predict whether platelet units requested by clinicians and issued by the hospital blood bank would be transfused to patients. The performance was sufficiently good to reduce wastage due to returns using my ML-guided YUPR issuing policy but, with an AUROC of 0.74, there is room for improvement. It may be possible to improve the predictive performance by including additional features representing diagnoses, procedures, and the indication for the planned transfusion if they are available at the time the issuing decision is made. A similar predictive model could also be used to support the decision about whether to issue a platelet unit in response to a request, not just which unit to issue, supporting the work of a platelet coordinator [274] to ensure that blood products are used appropriately and to reduce wastage. This could reduce wastage even in settings where units are not generally able to be reissued or are outside the hospital blood bank for a shorter period of time but would bring the use of predictive models closer to decisions on the clinical management of individual patients. Such use would, appropriately, come with additional ethical, professional and regulatory considerations.

7.3.1.4 Using implicit or explicit demand forecasting to support replenishment decisions

Part of the original motivation for this project was that real-time data from EHRs could be used to forecast demand, and these forecasts could be used to support

replenishment decisions. While it was not ultimately possible to follow that line of research due to challenges with access to data and computational resources, other research groups have recently used ML methods and EHR data to forecast demand for PRBCs [62, 91, 316] and platelets [61, 93, 94, 317] concurrently with my work and Quinn *et al.* [90] reported on the successful implementation of a system to recommend PRBC order quantities based on haemoglobin test results. The integration of a forecast, or features relevant to forecasting the demand, into the observation space of a policy trained using DRL has not yet been investigated. The eight product scenario considered in Chapter 6 shows a steep trade-off between low wastage and a high exact match percentage - a forecast of the demand for each blood group could help to improve both of these metrics. Dumkreiger [63] investigated the potential benefit of demand forecasts for PRBC inventory management using a simulation-first approach to support a heuristic replenishment policy. A similar simulation-first study may be a useful first step for investigating the benefits of demand forecasts for platelets: incorporating simulated forecasts with different levels of predictive accuracy and time horizons into the observation space for neural network replenishment policies.

7.3.1.5 Benefits of sharing data through the supply chain

An extension of using real-time data to predict demand at the hospital level would be to share information (both stock levels and aggregated data from EHRs that could support demand forecasting) through the supply chain, to support forecasting demand at higher levels. I understand from discussions held with member of the NIHR Blood Transfusion Research Unit in Data Driven Transfusion that NHSBT, who supply blood products to hospital blood banks in the UK, have very limited transparency of stock holding and activity within hospitals. Sharing information could help to create aggregate demand forecasts, which in turn could support collection and distribution efforts. However, developing and implementing the connections required for data sharing may be expensive and require agreement between multiple parties with different interests. Therefore this idea may also be well suited to a simulation-first approach, and modelled using a multi-agent RL

environment with agents representing decision makers at different echelons of the supply chain. Bhandawat *et al.* [318] simulated the use of a distributed ledger to share limited information between different participants in a regional blood supply chain but a more straightforward, centralized system may be feasible in the UK where the participants are part of single healthcare system. In the context of a retail supply chain, Meisheri *et al.* [155] investigated how the quality of forecasts affected the performance of replenishment policies fit using DRL. In addition to considering the benefits of information sharing, simulating multiple echelons of the blood chain could also help to understand the potential challenges if some participants start to use data-driven methods to support decision making - Meneses *et al.* [55] highlighted the importance of considering multiple echelons of the blood supply chain to avoid excessive sub-optimization which may lead to negative effects for other participants or at other levels.

7.3.1.6 Implementation studies

From my scoping review [21] and wider reading it is clear that the implementation of ML models, and other optimization methods [1], to support the management of blood product inventory is very limited. During the work on this thesis I have experienced many of the challenges associated with developing models based on patient-level data (e.g. approvals processes and the availability of appropriate computational resources on secure research computing environments). Implementation studies bring a range of additional challenges beyond those faced when making batch predictions on historical data including the need for a platform that can provide access to features in near-real time from multiple EHR systems, developing a suitable user interface for the outputs and system maintenance. The simulation-first approach can help to identify candidate decisions and models likely to lead to real-world benefits, but ultimately realizing those benefits for healthcare systems and patients will require additional work to shadow-test models and then evaluate their performance when used to inform real decisions.

7.3.1.7 Packed red blood cells

All of the above future research could also be applied to the inventory management of PRBCs. The differences between platelets and PRBCs, particularly the longer shelf life and increased importance of compatibility of PRBCs, will require different approaches to implementation but the core ideas are applicable to both products. In hospitals that do not have routine daily deliveries, but where both PRBCs and platelets are provided by the same supplier, there may also be benefits to considering a joint replenishment task for both products to minimize the number of deliveries and staff time spent processing orders.

7.3.2 Wider applications

Three elements of this work may help to support research into a range of problems beyond blood product inventory management: adopting a simulation-first approach to ML modelling in healthcare, using GPU-accelerated dynamic programming and simulation optimization, and GPU-accelerated multi-agent RL environments.

7.3.2.1 Simulation-first approaches to developing machine learning models in healthcare

In Chapter 5, I adopted a simulation-first approach to investigate a novel use case for ML: predicting whether requests for platelets would result in transfusion. This approach enabled me to understand how good a predictive model for this task would need to be in order to achieve improvements in KPIs of interest to decision makers when utilized in a workflow, and required only high-level summary data.

The path to realizing value from ML applications in healthcare requires the deployment of models inside existing clinical or operational workflows, or the development of new workflows based on their predictions. In cases where the clinical workflow can be readily modelled this approach can provide early information about whether model development is likely to be beneficial. This could be particularly beneficial when access to patient-level or other sensitive data is a challenge: if even a perfect predictive model cannot support meaningful improvements in KPIs then the often time-consuming process of obtaining

approvals and access to data will not be worthwhile. It would also, by necessity, encourage engagement at an early stage with staff who understand current practice and whose work would be supported by model predictions.

In addition, results from early workflow simulations can be used to inform model selection. In Chapter 5 I used the partial-AUROC, calculated for a range of false-positive rate values determined by the simulation results, when tuning the hyperparameters of the predictive model in an effort to ensure that improvements in model performance would correspond to improvements in KPIs.

7.3.2.2 GPU-accelerated dynamic programming and simulation optimization

In Chapter 4, I developed a versatile Python class for running value iteration on one or more GPUs, supporting by the Python library JAX, to estimate the optimal policy for an MDP. The general Python class was adapted for four scenarios and, using this approach, I was able to extend the size of problems for which value iteration was feasible and practical compared to recent studies.

As I set out in Chapter 4, the ability to determine the optimal policy for larger, more realistic problems has two major benefits. The first is that, where real problems of interest are now feasible, the optimal policy can be estimated and used in practice. The second is that the optimal policies can be used to benchmark heuristic and approximate approaches on a larger range problems.

My Python implementation for value iteration could be extended from its current state, as research code with some automated tests and an interactive demonstration notebook, into a fully-fledged software library with additional tests, a wider range of examples for different problems, and full documentation to support future research efforts.

In the same chapter, I used GPU-accelerated RL environments implemented using the Python library gymnax [119] to simulate MDPs and fit the policies of heuristic policy parameters using simulation optimization. This has wide applicability and could support work on open problems within perishable inventory management, including replenishment in small, rural blood banks with low demand,

and on a range of problems in other research areas that can be solved using dynamic programming. Implementing meta-heuristic methods for searching integer parameter spaces in the API of the the Python library evosax [121], which was built to be compatible with gymnax RL environments, could facilitate the benchmarking and comparison of different methods for fitting heuristic policy parameters when the performance of a large number of candidate solutions can be evaluated on a large number of episodes in parallel.

More broadly, these serve as examples of how software libraries from the ML community can provide an accessible way for researchers in other disciplines, without extensive experience in GPU programming, to take advantage of modern GPU hardware.

7.3.2.3 Further development of GPU-accelerated multi-agent environments

In Chapter 6 I developed a GPU-accelerated multi-agent RL environment based on the single agent environments from the Python library gymnax [119]. The environment handles a crucial feature of our problem of interest (heterogeneous agents acting at different frequencies) but does not currently scale well to larger problems due to the memory required to track information for multiple agents.

Concurrent to my work, a group including the author of the gymnax library released a Python library including a general GPU-accelerated multi-agent environment and compatible implementations of popular multi-agent RL algorithms [319]. The agents are assumed to act in parallel and therefore for our problem, a dummy action would need to be included at each step depending on whether the step was a replenishment step or an issuing step. Building on both of these to develop a general JAX-based multi-agent RL environment that scales to larger problem sizes and does not assume that all agents act in parallel, would enable the benefits of GPU-accelerated environments to be extended to a wider range of problems that can be modelled using multiple agents.

The adapted single-agent environment I developed as part of the work in Chapter 6 may be more immediately useful for work on multi-agent problems where

the agents share the same reward function, but act at a different frequencies.

7.4 Concluding remarks

This thesis highlights how advanced computational techniques and ML can support decision-making in blood product inventory management, contributing to more efficient and effective healthcare logistics: reducing wastage and helping to ensure the right unit is available for the right patient at the right time.

Appendix A

Background information on packed red blood cells

This appendix is a counterpart to the description of platelets in Section 2.2. While I have not considered the inventory management of packed red blood cells (PRBCs) directly in this thesis, much of the relevant work on blood product inventory management is concerned with PRBCs.

Red blood cells deliver oxygen to cells throughout the body, where it is required to generate energy via aerobic respiration. This process is facilitated by haemoglobin, a protein contained in red blood cells to which oxygen binds for transportation. PRBC units are red blood cells that have been processed for transfusion by separating them from other blood components including platelets and plasma.

A PRBC transfusion is one of the most common medical procedures administered to hospital inpatients [320]. Patients may receive a transfusion if they are actively bleeding or if they they are otherwise suffering from anaemia, where there are not enough healthy red blood cells to deliver sufficient oxygen to their cells. The goal of a transfusion to is prevent hypoxia: tissue damage caused by a lack of oxygen [321].

In the most recent annual report available on their website the Blood Stocks Management Scheme, run by NHSBT, reported that total demand for RBCs from English hospitals in 2022/2023 was 1.4M units. Wastage as a percentage of units

issued was 2.2% during this period, and the main cause of this wastage was time expiry [3]. As of June 2024, a standard unit of adult PRBCs in England cost £158 [16].

In the UK, PRBC units have a useful life of 35 days from donation if kept refrigerated. PRBC units may be irradiated within 14 days of donation and then have a remaining useful life of 14 days from irradiation [13, p.20]. PRBC units may also be washed to remove most of the the plasma - if this is done in an automated, closed system them they have a remaining useful life of 14 days from washing, if it is done manually then they have a remaining useful life of 24 hours from washing [13, p.20].

In the past, if a patient was likely to require an PRBC transfusion then a serological crossmatch test would be performed to test for potential incompatibility, physically mixing a sample of the patient's plasma with a sample of red blood cells from the units intended to be used for transfusion. This process was time-consuming, and therefore when compatible units were identified they were assigned to patients for a fixed period (the cross-match release period). Now a group and screen test is used to determine the patient's ABO and RhD blood type, and screen for blood group antibodies. In most cases, based on the results from this test, compatible units can be identified by remote issue (computer cross-matching) systems without any further tests and there is no need to hold inventory for specific patients unless they have special transfusion requirements that make remote issue unsuitable [13, p.10-11].

A "major" ABO incompatibility occurs when the donor's ABO antigens are incompatible with the recipient's antibodies and is the main considerations for PRBC transfusions. An exact match is preferred for PRBC transfusions, but is not essential. Transfusions of RhD+ blood to RhD- girls or women of childbearing age is avoided unless there is no alternative due to the potential risk to a RhD+ fetus carried by an RhD- mother who has developed anti-D antibodies [13, p.9]

Some patients require regular transfusions throughout their lives, due to conditions such as sickle cell anaemia and thalassemia. These frequent transfusions

increase the risk of the patients developing antibodies, and therefore of an adverse transfusion reaction. Special care is taken to match the blood types of these patients with the blood type of the donors as closely as possible, and they are normally transfused with units that have been specially collected and/or ordered for them.

Appendix B

Blood product inventory management at UCLH

B.1 Introduction

This appendix describes the operations relevant to my work at our partner site, the transfusion laboratory serving UCLH. I explain the role of the transfusion laboratory and their current practices, based on a site visit and discussions with staff. I also present descriptive analysis covering key aspects of blood product use for the period 1 January 2015 to 31 December 2017, and comment on the importance of these findings for my subsequent research.

Unless otherwise cited, the information in this chapter about the role and operations of the transfusion laboratory was obtained during a site visit on 30 July 2021, during which I discussed the topics with Ian Longair and Philip Russell, both Senior Biomedical Scientists at the laboratory.

B.2 The role of the transfusion laboratory

The transfusion laboratory is run by Health Services Laboratories and serves the six UCLH hospitals in central London. The transfusion laboratory is a hospital blood bank responsible for ordering, storing, processing, managing and issuing fresh blood products. It also performs blood group, antibody and antigen testing.

The UCLH transfusion laboratory faces some of the highest demand in the country, and is also in a location that allows multiple routine deliveries per day.

Findings from UCLH may therefore not be generalizable to lower-demand sites, or to sites in remote locations.

B.3 Inventory management practice

I focus on PRBCs and platelets, but the laboratory also manages other blood products including fresh frozen plasma.

PRBCs that have not yet been allocated to a specific patient are stored in fridges in the transfusion laboratory, and in 10 electronic remote issue fridges located across the UCLH estate. Unallocated platelets are stored in a platelet agitator located in the transfusion laboratory. PRBC and platelet units ordered from NHSBT for patients with special requirements are also stored in the fridges or agitator in the transfusion laboratory until they are required.

All stock is ordered from NHSBT using the NHS Online Blood Ordering System (OBOS). In the transfusion laboratory, inventory is tracked using two main systems: Bank Manager and BloodTrack Manager. Bank Manager is a laboratory information management system that stores information about the blood products in stock, patients, test results and requests for blood products. BloodTrack Manager is used to track activity and monitor stock levels in the remote issue fridges.

B.3.1 Orders and deliveries

The transfusion laboratory receives three routine deliveries each weekday, and one routine delivery each day at weekends. The approximate delivery and order cut-off times for these deliveries are set out in Table B.1. There is no additional fixed order cost for these deliveries because they are covered by an annual service charge.

	Order cut-off time	**Delivery time**
Weekday	0500	0830
	1130	1430
	1630	2100
Weekend	0930	1330

Table B.1: Approximate order cut-off and delivery times for routine daily deliveries to the UCLH transfusion laboratory.

For platelets, the transfusion laboratory has a standing order of units to be delivered in the 0830 slot on weekdays and the 1330 slot on weekends. In July 2021, when I visited the transfusion laboratory, this order was for 19 units.

For PRBCs, the number of units ordered in the routine deliveries is determined using 'order-up-to' target stock levels. The laboratory staff use pro-forma that specifies the target stock of different blood types in each location. On weekdays, the unallocated stock in the transfusion laboratory fridges is reviewed twice daily, at approximately 0400 and 1600, and orders are placed for the 0830 and 2100 delivery slots to bring the stock up to the target stock level. A similar process is performed once per day using BloodTrack Manager to determine the current stock in the 10 remote issue fridges - an order that will bring the stock levels up to the target stock (taking into account any excess stock held in the laboratory fridges that could be transferred to them) will then be made for either the 1430 or 2100 delivery slot. The same processes are followed at the weekend, but with the reviews performed so that an order can be made for the single 1330 routine delivery slot.

If patients have special requirements, and the transfusion laboratory is given advance notice of a planned transfusion, the required units will normally be added to a routine order so that they arrive on the morning they are expected to be needed.

In addition to routine orders, ad hoc and emergency orders can also be made to NHSBT. Placing an ad hoc order costs £10, plus approximately £50 for delivery, and are received within 1-2 hours depending on traffic. Emergency orders incur similar costs but arrive more quickly because they are delivered by paramedics using blue lights and sirens. Ad hoc orders are generally made for patients with special requirements, or when there has been unusually high usage and additional units are expected to be required before the next routine delivery slot. Emergency orders are very rarely made because UCLH is not a trauma centre.

When an order is received, the stocks manager in the transfusion laboratory ticks off each unit against the delivery note and scans it into Bank Manager using the product barcode.

B.3.2 Issuing units

The majority of PRBC units are taken directly from remote issue fridges by nurses. The fridges automatically select the units to be issued based on the patient's blood type and other requirements. In general, the oldest unit that is ABO and RhD compatible with the patient will be issued. When the unit is issued by a remote fridge, the event is recorded in BloodTrack Manager, and subsequently in Bank Manager via the interface between those two systems. A minority of patients cannot safely receive blood by remote issue, this includes patients where there is a discrepancy in their blood group tests, where they have current or historic antibodies to blood group antigens, and those who have other special requirements. In these cases, clinical staff will make a request for units to the transfusion laboratory via telephone or email. Staff in the transfusion laboratory register the request in Bank Manager, and search the stock list in Bank Manager for a compatible unit. If a potentially compatible unit is identified, a physical cross-match will be performed. If the physical cross-match confirms that the unit can be safely transfused to the recipient then a courier will deliver the unit(s) to a fridge near the patient, or to the patient's bedside, ready for transfusion. If a suitable unit is not available in stock, then an ad hoc or emergency order will be placed through OBOS.

Platelet units are all kept in an agitator in the transfusion laboratory until issued to meet a request. Similar to the process for PRBCs that are not remote issued, clinical staff place a request with the transfusion laboratory via phone or email, and the transfusion laboratory staff register the request in Bank Manager. The transfusion laboratory staff will identify a compatible unit using Bank Manager, or place an ad hoc order if required. The unit(s) are then delivered by courier to a local platelet agitator or to the patient's bedside for the requested time. In general, the transfusion laboratory staff try to ensure that the allocated units are the same ABO blood group as the patient, and then select the oldest unit. If there are units in stock that expire at the end of the current day, they will be issued to meet the next request for which they are an acceptable match.

Products ordered for specific patients are held in the transfusion laboratory for

their use. However, if they are expected to expire before being transfused to the patient for whom they were requested (due to a no show or cancellation), then they will enter the pool of units that may be allocated to other patients.

UCLH's electronic health record system Epic, which came into use in 2019, interfaces with Bank Manager to record the administration of transfusions. In the past, the fate of transfused units was recorded manually on Bank Manager by transfusion laboratory staff based on paper forms completed in the wards and clinics.

B.3.3 Stock movement

Couriers move units from the transfusion laboratory to a) top up the PRBC stock of the remote issue fridges and b) move issued units of PRBC and platelets to fridges and agitators near a patient, or directly to a patient's bedside.

Individual remote issue fridges have 'alarm levels' for low stock. When these alarms trigger on BloodTrack Manager, the stocks manager in the transfusion laboratory will dispatch PRBC units via courier to at least bring the stock level in the fridge to above the alarm level. This process helps to reduce the number of local shortages.

Couriers and transfusion laboratory staff visit fridges and agitators outside the laboratory between 1000 and 1300 each day. They return all platelets back to the laboratory, and any PRBCs issued by the laboratory more than 24 hours ago. Additionally, PRBCs in low volume remote issue fridges with less than 7 days of useful life are moved to high volume fridges to maximize the chance of them being transfused before expiry.

B.3.4 Performance monitoring

Weekly numbers on wastage, extracted from Bank Manager, are reported to consultant responsible for the transfusion laboratory. The Hospital Transfusion Team hold a monthly meeting which includes a detailed review of usage and wastage statistics from the previous month in the context of prior periods. There is no fixed review period for considering ordering policies (e.g. the size and

composition of the standing order for platelets and the order-up-to levels for PRBCs). Instead, these policies are considered as and when an issue is identified in the monthly management information prepared for the Hospital Transfusion Team meeting.

NHSBT will contact the transfusion laboratory if they have concerns about recent orders and usage, for example using products that meet special requirements where not strictly necessary, and this may trigger a targeted audit, conducted by transfusion laboratory staff, into the issue to identify the source of the problem and potential solutions.

B.4 Descriptive analysis

B.4.1 Data source

I received pseudonymized records of requests, products and tests extracted from an archive copy of the laboratory information system, Bank Manager, covering the period 1 January 2015 to 31 December 2017. Requests were included if the request was required during the period. Products were included in the original extract if they were received after 1 November 2014 to ensure products received before the start of the period, but with a missing fate date, were captured. Tests were included if the result was available on or after 1 November 2014, to account for the fact that some requests and transfusions early in the period would have been based on group and screen results from prior to the start of the period.

B.4.2 Red blood cells

I identified a total of 92,887 PRBC units received between 1 January 2015 and 31 December 2017. 2,566 (2.8%) of these units were neonatal units. In the rest of this analysis I consider only adult units.

In Table B.2 I present a breakdown of the fate of adult units received in the period by year. I include 132 units with no fate recorded as miscellaneous wastage. The categories for wastage are defined in Joint UK Blood Transfusion and Tissue Transplantation Services Professional Advisory Committee (JPAC) guidelines [322]. The wastage levels, and time expiry as the main source of

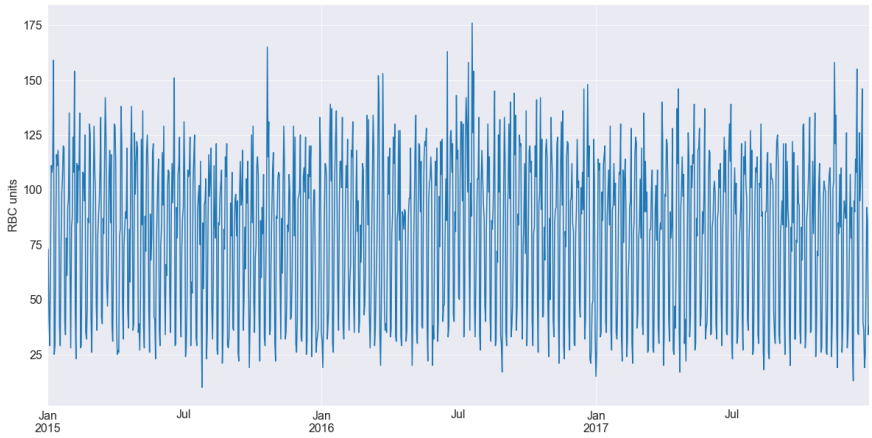
waste, are generally consistent with the rate of 2.4% reported by the Blood Stocks Management Scheme annual report for hospitals in England in the period 2017/2018 [2], although there is an increase in time expired units of approximately 30% between 2016 and 2017.

In Figure B.1 I present a plot of the daily units of adult PRBC units transfused during the period, and boxplots showing the distribution of daily PRBC units received and transfused by year, month and weekday. Figures B.1a and B.1d show that there is a clear weekly seasonality to demand, with far fewer transfusions at the weekend than during weekdays. This is due to scheduled procedures taking place during the week. Demand appears relatively stable over the three years from Figure B.1b, and there is no clear annual seasonality pattern in Figure B.1c. During this period, there was no routine delivery on Sundays, which is part of the reason why the number of units received on a Sunday is so low in Figure B.1d - all the units received would have been from ad hoc or emergency orders. This weekly pattern is consistent with previous work [62].

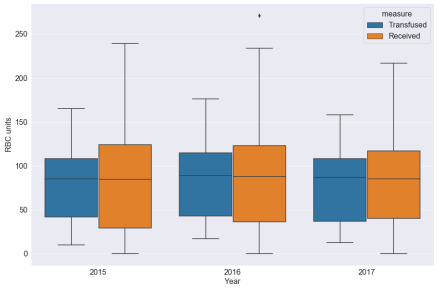
Fate	**2015**		**2016**		**2017**	
Transfused	28,850	97.7%	30,252	97.5%	28,682	96.5%
Time expiry	369	1.2%	475	1.5%	626	2.1%
Out of temperature control outside laboratory	161	0.5%	170	0.5%	157	0.5%
Fridge failure	9	0.0%	0	0.0%	79	0.3%
Miscellaneous	128	0.4%	142	0.5%	178	0.6%
Wasted	667	2.3%	787	2.5%	1,040	3.5%
Transferred or returned to NHSBT	18	0.0%	11	0.0%	14	0.0%
Total	**29,535**		**31,050**		**29,736**	

Table B.2: Fate of adult PRBC units by year received at the UCLH transfusion laboratory.

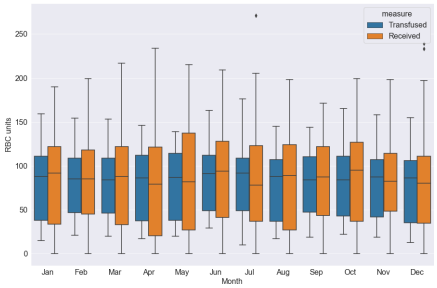
In Figure B.2a I present the number of days in advance that a request for PRBC units was made. In this plot, an order is considered to have been made one day in advance if registered any time before midnight on the day before it was required. Approximately 74% of requests are made on the same day. This suggests that daily forecasting could be beneficial - if most demand was registered before the end of the preceding day then the requests could be used to plan orders instead of using forecasts. The units that are requested and required on the same day include those



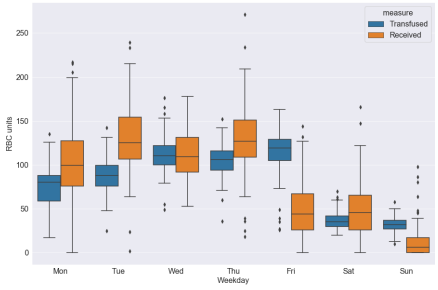
(a) Daily adult PRBC units transfused.



(b) Daily adult PRBC units received and transfused by year.

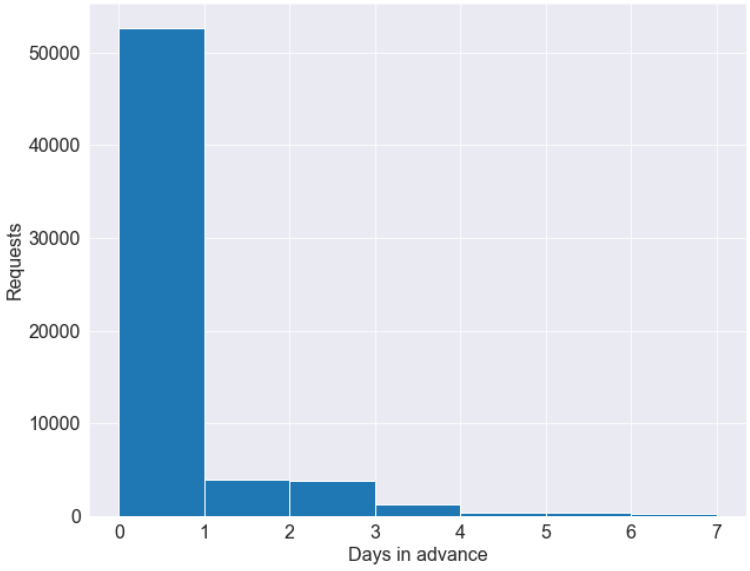


(c) Daily adult PRBC units received and transfused by month.

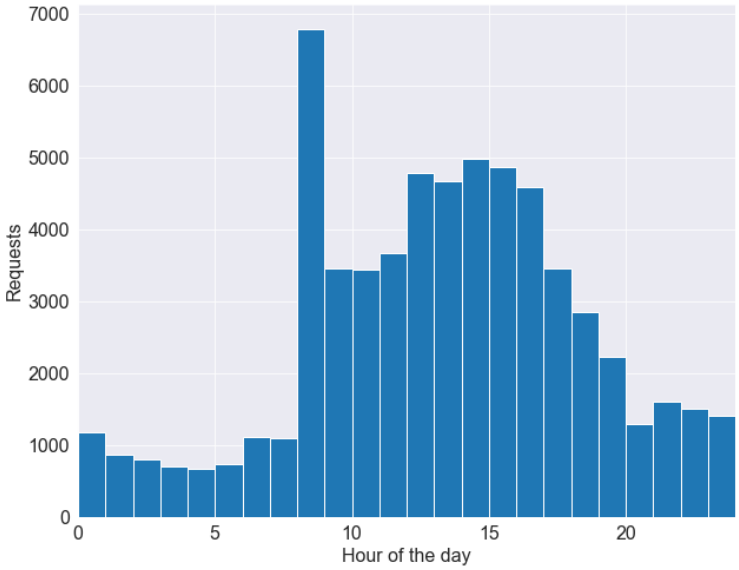


(d) Daily adult PRBC units received and transfused by weekday.

Figure B.1: Adult PRBC units received and transfused at UCLH between 2015 and 2017.



(a) Days between request for adult PRBC units being recorded and required.



(b) Hour of the day that request for adult PRBC units is required.

Figure B.2: Timing of requests for adult PRBCs at UCLH between 2015 and 2017.

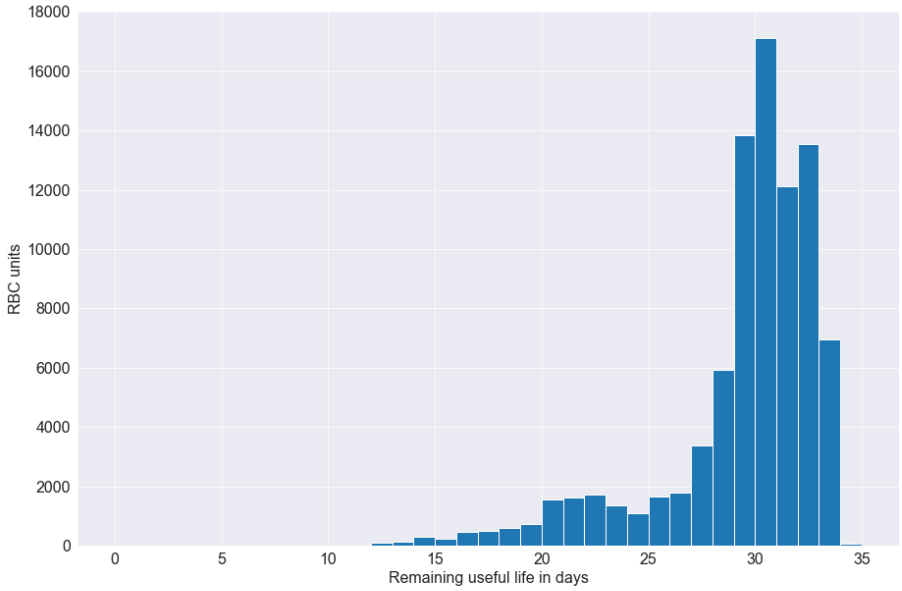
requested from remote fridges, rather than through the transfusion laboratory, in which case the time that the request is registered and required are the same.

In Figure B.2b I present the hour at which requests for adult PRBCs are required. The peak between 0800 and 0900 is due to orders placed at least day in advance. In the system these are most often recorded as being required at exactly 0800, indicating that they will be needed at some point during that day during normal operating hours.

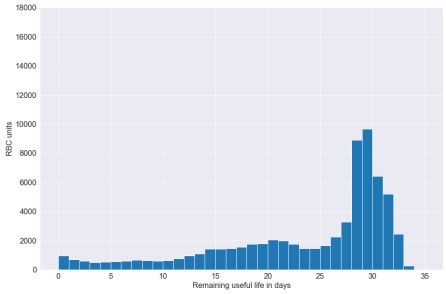
96% of the adult PRBC units received during the period were standard units of PRBCs in additive solution, which have a useful life of 35 days from donation. Approximately 19% of these units are irradiated after being received in the transfusion laboratory, from which point their remaining useful life is 14 days. The majority of the remainder are adult PRBC units that have been washed using an automated process before arrival at the transfusion laboratory, which have a remaining useful life of 14 days from the point of washing. In Figure B.3a I present the useful life at arrival for adult PRBC units in additive solution received during the period. In Figures B.3b and B.3c I present the remaining useful life on transfusion for adult PRBC units in additive solution that were not and were irradiated, respectively, after arrival in the transfusion lab. Figure B.3c shows that irradiated units are most commonly used the day after irradiation.

In Table B.3 I present the proportion of adult PRBC units received by ABO/RhD blood type and, for each blood type, the proportion of units received that were transfused. In general a lower percentage of units that are compatible with a small percentage of patients were transfused - for example AB- units can only be safely transfused to patients who have blood types AB+ and AB-, and B+ units can only be safely transfused to patient who have blood types B+ and AB+.

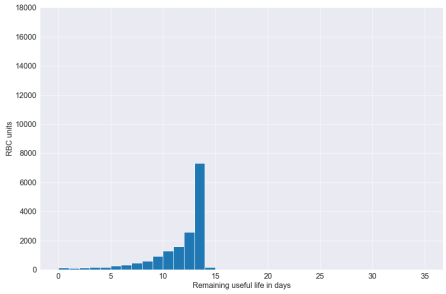
In Table B.4 I present compatibility between the blood type of the patient receiving the blood and the donor's blood type for requests where the units were required during the period. Transfusions are only included in this table where I was able to identify a group and screen result for the patient where the results were available before the time that the request was required, but this represents over 99%



(a) Remaining useful life in days at arrival for adult PRBC units in additive solution.



(b) Remaining useful life in days at transfusion for adult PRBC units in additive solution that were not irradiated.



(c) Remaining useful life in days at transfusion for adult PRBC units in additive solution that were irradiated.

Figure B.3: Remaining useful life of adult PRBC units in additive solution at receipt and transfusion at UCLH between 2015 and 2017.

of adult PRBC units transfused in the period. RhD- patients almost always receive PRBCs from RhD- donors, and substitution is often used where safe (for example, patients with ABO group AB patients can be safely transfused with any ABO group, and have the lowest ABO match percentages).

Blood type	Percentage of total units received	Percentage transfused
A+	23.3%	98.6%
A-	7.7%	95.2%
B+	8.0%	90.0%
B-	5.5%	94.5%
AB+	1.7%	93.7%
AB-	0.4%	62.2%
O+	33.0%	98.5%
O-	20.4%	95.7%
	100.0%	

Table B.3: Proportion of PRBC units received at the UCLH transfusion laboratory by blood type and, for each blood type, the proportion of units received that were transfused.

Recipient blood type	**Percentage of all transfused units transfused to patients with this blood type**	**Percentage of units transfused to this type**		
		Exact ABO and RhD match	ABO match	RhD match
A+	28.6%	80.5%	91.0%	85.2%
A-	4.5%	85.9%	85.9%	99.9%
B+	15.6%	50.3%	72.1%	65.1%
B-	2.4%	70.2%	70.2%	100.0%
AB+	4.0%	41.9%	46.9%	75.5%
AB-	0.2%	26.4%	26.4%	100.0%
O+	39.0%	76.2%	100.0%	76.2%
O-	5.6%	100.0%	100.0%	100.0%
	100.0%			

Table B.4: Proportion of PRBC units transfused at the UCLH transfusion laboratory by recipient blood type and, for each blood type, the proportion of units transfused that were an exact ABO/RhD match, an ABO group match, and an RhD group match.

B.4.3 Platelets

I identified a total of 27,473 platelet units received between 1 January 2015 and 31 December 2017. 409 (1.5%) of the units were neonatal units which are not kept in stock and are ordered on an ad hoc or emergency basis when required. In the rest of this analysis I consider only adult units.

In Table B.5 I present a breakdown of the fate of adult units received in the

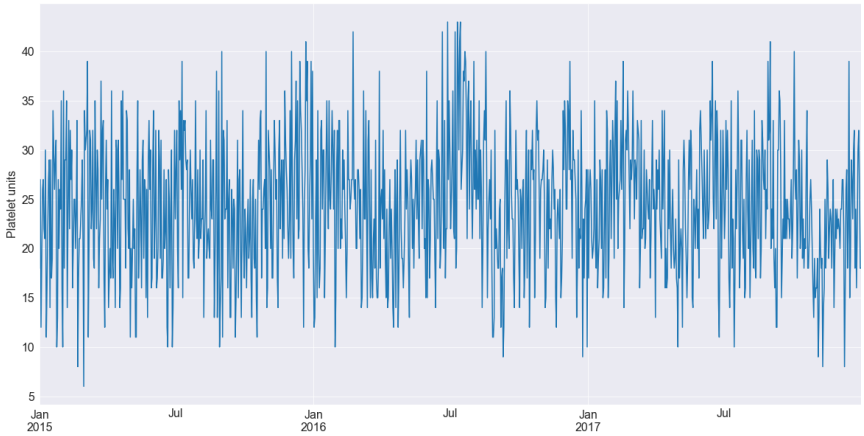
period by year. I include 14 units with no fate recorded as miscellaneous wastage. The categories for wastage are defined in the JPAC guidelines [322]. The wastage levels, and time expiry as the main source of waste, are consistently lower than the rate of 4.3% reported by the Blood Stocks Management Scheme annual report for hospitals in England in the period 2017/2018 [2] although, as with PRBCs, wastage is higher in 2017 compared to the earlier years.

In Figure B.4 I present a plot of the daily units of adult platelets transfused during the period, and boxplots showing the distribution of daily platelet units received and transfused by year, month and weekday. Figures B.4a and B.4d show a similar weekly demand pattern to that observed for PRBCs, with far fewer transfusions at the weekend. The weekly seasonality is consistent with prior research including Guan *et al.* [11] and Motamedi *et al.* [61]. As with PRBCs, annual demand appears relatively stable over the three years from Figure B.1b. There appear to be periods of higher and lower demand in Figure B.4, but demand does not exhibit a clear seasonal pattern.

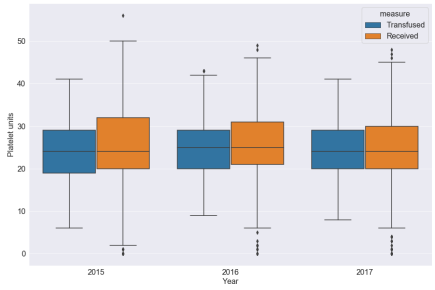
Fate	**2015**		**2016**		**2017**	
Transfused	8,796	98.7%	9,057	98.5%	8,776	97.9%
Stock time expired	26	0.3%	47	0.5%	58	0.6%
Medically ordered, not used	55	0.6%	55	0.6%	77	0.9%
Surgically ordered, not used	1	0.0%	4	0.0%	5	0.1%
Wasted outside laboratory	4	0.0%	6	0.1%	5	0.1%
Miscellaneous	25	0.3%	19	0.3%	31	0.3%
Wasted	111	1.2%	131	1.5%	176	2.0%
Transferred or returned to NHSBT	7	0.1%	2	0.0%	8	0.1%
Total	**8,914**		**9,190**		**8,960**	

Table B.5: Fate of adult platelet units by year received at the UCLH transfusion laboratory.

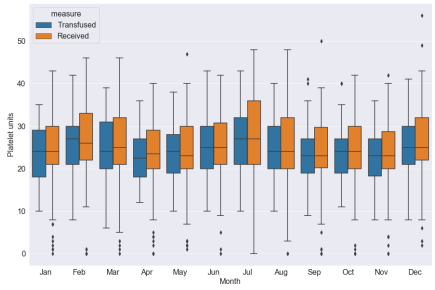
In Figure B.5a I present the number of days in advance that a request for adult platelet units was made. Approximately 90% of requests are made on the same day. As above, this suggests a role for forecasting because relatively little of the demand is known before routine order must be placed. In Figure B.5b I present the hour at which requests for adult platelet are required, showing that the majority of demand occurs during normal working hours.



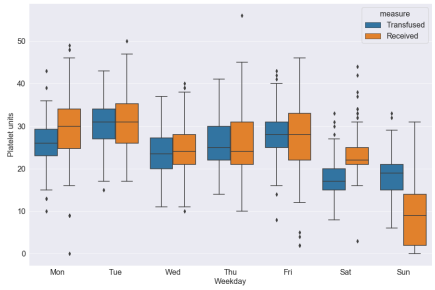
(a) Daily adult platelet units transfused.



(b) Daily adult platelet units received and transfused by year.

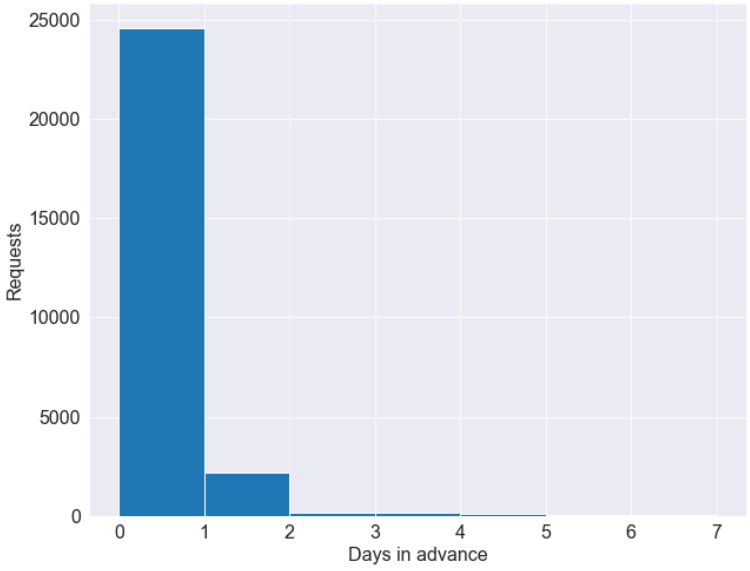


(c) Daily adult platelet units received and transfused by month.

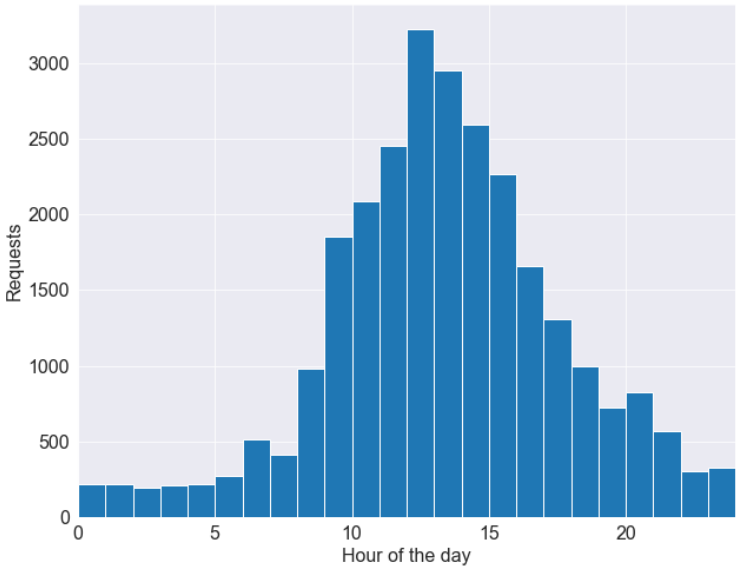


(d) Daily adult platelet units received and transfused by weekday.

Figure B.4: Adult platelet units received and transfused at UCLH between 2015 and 2017.



(a) Days between request for adult platelet units being recorded and required.



(b) Hour of the day that request for adult platelet units is required.

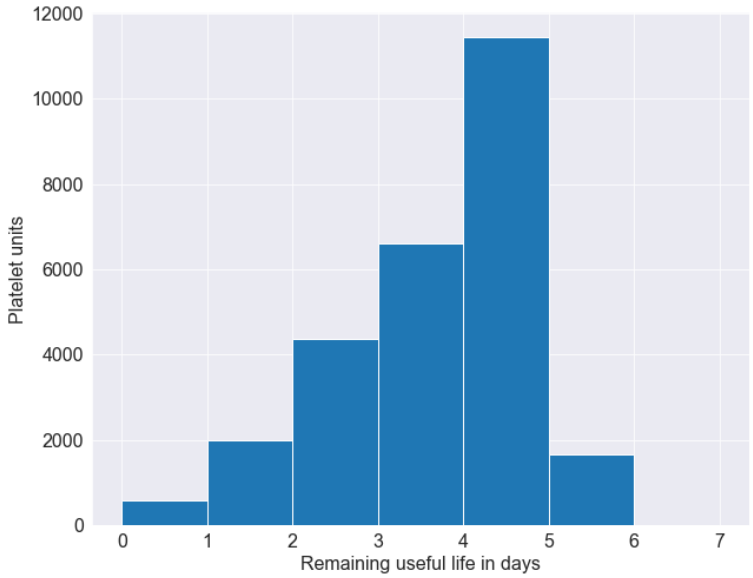
Figure B.5: Timing of requests for adult platelet units at UCLH between 2015 and 2017.

99% of the adult platelet units received during the period were extended-life platelets which have a useful life of 7 days from donation. In Figure B.6a I present the useful life at arrival for extended-life platelet units received during the period. In Figures B.6b I present the remaining useful life on transfusion for extended-life platelet units. Of the units that were transfused, 8% were transfused on the last day of their useful life.

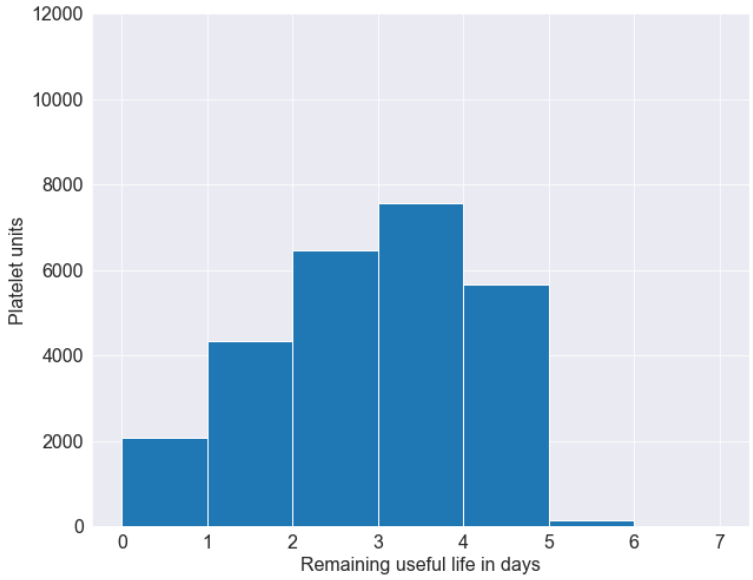
If a platelet unit is issued and then not transfused to a patient and returned to the transfusion laboratory, this back-and-forth can consume a significant period of the unit's useful life. In Figure B.7 I present a flow chart showing the the outcome of issuing adult platelet units. Only a very small minority, 0.6%, of platelet units received by the transfusion laboratory are never issued. The majority of units that are not transfused are issued only once. This raises two potential issues for my work: the first is that it may be important to include the concept of returns in my simulations because the hospital blood bank must have sufficient stock to meet all of the requests even if approximately 10% of the time those units are returned. The second is that it may be possible to reduce waste by creating a model that predicts which requests are unlikely to result in transfusion and prompts transfusion laboratory staff to discuss with the clinical team making the request whether it is actually necessary.

In Table B.6 I present the proportion of adult platelets received by ABO/RhD blood type and, for each blood type, the proportion of units received that were transfused.

In Table B.7 I set out the compatibility between the blood type of the patient receiving the blood and the donor's blood type for requests where the platelet units were required during the period. Transfusions are only included in this table where I was able to identify a group and screen result for the patient where the results were available before the time that the request was required, but this represents over 99% of adult platelet units transfused in the period.



(a) Remaining useful days at arrival for adult platelet units.



(b) Remaining useful days at transfusion for adult platelet units.

Figure B.6: Remaining useful life of adult platelet at receipt and transfusion at UCLH between 2015 and 2017.

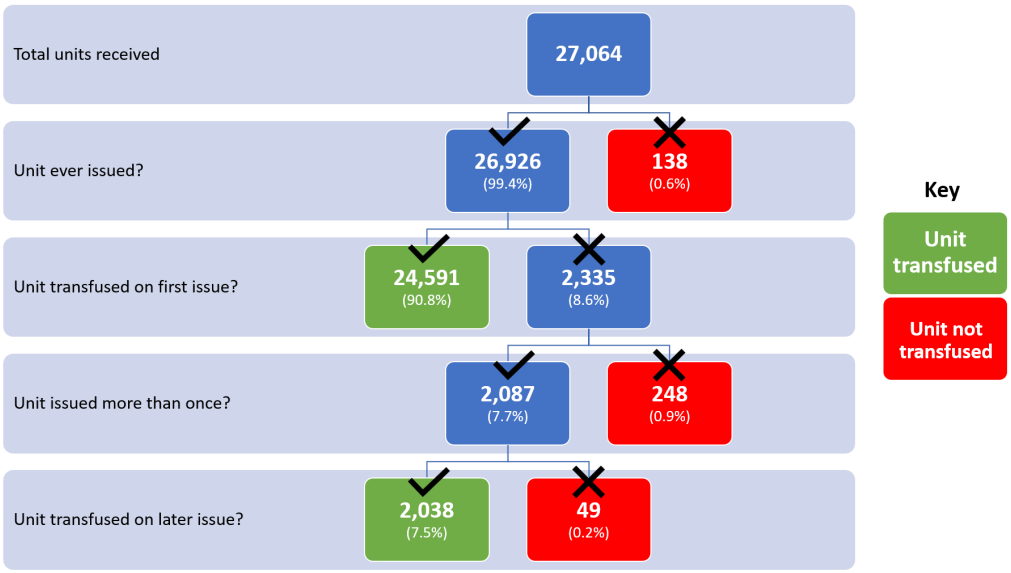


Figure B.7: Flow chart of how many times platelet units were issued and their subsequent fate at UCLH between 2015 and 2017.

Blood type	Percentage of total units received	Percentage transfused
A+	45.7%	98.7%
A-	24.2%	98.4%
B+	6.6%	98.4%
B-	1.6%	96.3%
AB+	4.3%	98.1%
AB-	1.0%	96.4%
O+	6.5%	96.6%
O-	10.0%	98.7%
	100.0%	

Table B.6: Proportion of adult platelet units received at the UCLH transfusion laboratory by blood type and, for each blood type, the proportion of units received that were transfused.

B.5 Conclusion

UCLH is not typical of transfusion laboratories in the United Kingdom due to its relatively high demand. The overall patterns of demand, with clear weekly seasonality but no obvious monthly seasonality, are consistent with previous reported results from other countries.

The UCLH transfusion laboratory performs well in terms of wastage, with similar wastage of PRBCs and lower wastage of platelets than reported for hospitals in England during the same period. This is perhaps to be expected given the relatively high demand. The relatively high wastage of platelet units that are issued

Recipient blood type	**Percentage of all transfused units transfused to patients with this blood type**	**Percentage of units transfused to this type**		
		Exact ABO and RhD match	ABO match	RhD match
A+	30.8%	71.1%	90.7%	78.7%
A-	5.0%	88.5%	92.1%	94.9%
B+	13.1%	33.6%	38.7%	80.2%
B-	2.2%	12.7%	12.7%	100.0%
AB+	3.6%	37.0%	42.1%	83.3%
AB-	0.4%	8.7%	8.7%	100.0%
O+	37.2%	12.6%	29.3%	66.7%
O-	7.8%	41.1%	42.7%	94.9%
	100.0%			

Table B.7: Proportion of adult platelet units transfused at the UCLH transfusion laboratory by recipient blood type and, for each blood type, the proportion of units transfused that were an exact ABO/RhD match, an ABO group match, and an RhD group match.

by the laboratory but not transfused suggests that this is a process that should be incorporated into my work, both to cover the complete demand seen by the transfusion laboratory and to investigate whether changes in issuing policy could reduce this wastage. This is not something that has been considered in prior work in the area.

The timing of requests for both PRBCs and platelets suggested that there could be a benefit to short-term forecasting: the vast majority of demand for both blood products is same-day. If much of the demand was known further in advance then that could be incorporated into orders with less need for forecasting. Additionally, there is room for improvement in the percentage of transfusions where there is an exact match between the patient and donor ABO/RhD blood types. This could be supported by type-specific forecasts and order recommendations like those produced by Quinn *et al.* [90] so that the units in stock more closely match the needs of the current inpatients.

Appendix C

Chapter 3 supplement: Evaluating previously reported heuristic policy parameters on the reinforcement learning environment

In order to properly evaluate the performance of DRL methods on the simplified platelet replenishment scenario described by Rajendran and Srinivas [45], it is necessary to establish relevant benchmark policies. I initially intended to use the policies described in Rajendran and Srinivas [45], using the parameters reported in the study, as a benchmark solution and therefore implemented these policies and applied them to my RL environment. In this Appendix I describe my approach to estimating the performance of the four policies with the reported parameters, and why I ultimately reimplemented the stochastic mixed integer linear programming approach to re-fit the parameters for these policies.

C.1 Methods

I created custom Python `Agent` classes compatible with my OpenAI Gym environment for each of the four heuristic ordering policies described in Rajendran and Srinivas [45]. The ordering rules for the four policies are set out in Equations 3.6 to 3.9 and the parameters reported for those policies in Rajendran and Srinivas

[45] are presented in Table C.1.

I applied these policies to the 1,000 evaluation rollouts (described in Section 3.4.5) and recorded the daily rewards (and reward components).

Policy	Parameter	Weekday						
		Mon	Tue	Wed	Thu	Fri	Sat	Sun
$(s, S)^{\dagger}$	s	10	8	17	9	15	13	5
	S	51	58	60	39	43	32	30
(s, Q)	s	7	9	20	10	21	18	7
	Q	50	34	45	33	30	48	34
(s, S, α, Q)	s	19	18	13	10	10	11	19
	S	50	48	40	54	31	30	45
	α	18	13	10	8	7	10	13
	Q	32	14	27	35	22	16	34
(s, S, β, Q)	s	18	15	12	12	22	9	5
	S	51	48	40	59	43	29	45
	β	1	1	0	3	14	0	1
	Q	33	31	0	46	21	0	35

Table C.1: Heuristic policy parameters for the baseline platelet replenishment scenario from Table 2 of Rajendran and Srinivas [45]

C.2 Results

I present the mean daily costs incurred by each policy over the 1,000 evaluation episodes, split by component, in Table C.2. The results are presented alongside those reported in Rajendran and Srinivas [45] as costs output by their stochastic programming optimization process for comparison.

C.3 Discussion

I did not expect the two sets of costs presented in Table C.2 to be exactly the same: the shorter trajectories (of 30 days) in the original paper mean that initial and terminal conditions will make a larger relative contribution to the mean values. I am interested in evaluating the long-term performance of the policies.

The two largest elements of the total cost (variable purchasing cost and shortage cost) are similar between both sets of results and the same relative performance can be observed for these measures: for example in both cases the (s, S, α, Q) policy has the lowest total cost, the lowest shortage cost and the second

Policy	Source	Mean daily cost					
		Fixed	Variable	Holding	Shortage	Wastage	Total
$(\mathtt{s}, \mathtt{S})^\dagger$	Ours	138	17,778	1,006	26,223	0	45,145
	[45]	146	17,565	4,524	27,101	2,106	51,442
$(\mathtt{s}, \mathtt{Q})$	Ours	156	18,619	947	22,019	0	41,742
	[45]	173	18,819	4,651	22,342	905	46,890
$(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$	Ours	188	20,754	819	11,342	0	**33,104**
	[45]	200	20,482	4,148	12,455	1,011	38,296
$(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$	Ours	195	20,060	601	14,811	0	35,668
	[45]	151	20,217	4,664	13,507	1,202	39,740

The daily cost components and total costs on the RL environment were calculated for each episode. The mean of the daily cost components and daily total cost over the 1,000 evaluation episodes is reported in the rows labelled "Ours".

Table C.2: Mean daily costs incurred on my RL environment using the heuristic replenishment policy parameters reported in Table 3 of Rajendran and Srinivas [45], presented alongside the reported performance from Table 2 of Rajendran and Srinivas [45] for the same parameters.

lowest variable purchasing cost while the $(\mathtt{s}, \mathtt{S})^\dagger$ policy has the highest total cost and the highest shortage cost.

There are differences between the two sets of results in the wastage and holding costs: both are much lower across all of the agents in my implementation.

It is surprising, given the relative value of the costs, that none of the policies make an order almost every day. If the policies were making an order every day, the mean daily fixed cost would be \$225. The fixed order cost is less than twice the holding cost per unit, and therefore I would not expect it to be cost effective to make larger orders (e.g. for two days' demand) at a lower frequency. This is particularly relevant given the high shortage costs observed in both sets of results - even the policy with the lowest shortage cost in my implementation, $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$, has an average daily shortfall of 3 units.

I want to use these heuristic policies as a benchmark to compare with the performance of DRL methods. If that comparison is to be valid, I need to be confident that the parameters I am using for those policies are good (if not optimal). Based on the results above, I considered that it may be possible to find parameters for these heuristic policies that perform better than the results reported by Rajendran and Srinivas [45] and my application of the parameterized policies to my RL environment. I therefore re-implemented the stochastic mixed integer programming

approach described in Rajendran and Srinivas [45] to investigate this, and to ensure that the performance of the DRL policies is not flattered by under-performing benchmark policies. The results of this subsequent work are reported in Section 3.4.5

Appendix D

Chapter 3 supplement: Stochastic programming formulation

In this appendix I reproduce the stochastic programming objective and constraints used to fit the policies described in Section 3.4.5. As I explain in that section, this is a reimplementation of the methods from Rajendran and Srinivas [45] but with amendments to

- explicitly constrain the policy parameter decision variables so that there is one per day of the week (instead of per timestep);
- add additional constraints to enforce the OUFO issuing policy; and
- correct a typographical error in their equation 9.

These modifications are described in Section D.4 below. The indices, sets, parameters, decision variables, objective function and other constraints are reproduced from Rajendran and Srinivas [45] using the original notation instead of the alternative notation used elsewhere in this thesis.

I did not include the review period in this reproduction of the constraints, because in both my analysis and in the original paper it is not changed and the stock is reviewed each day.

The formulation below assumes that the weekdays are numbered at 0 to 6, with Sunday as 0. In practice, when implementing these models, I used the common Python convention that Monday is the weekday with index 0 and therefore replaced

$t\%7$ with $(t-1)\%7$ to maintain the fact that the weekday of the first timestep should be Monday. I use $x\%y$ to represent $x \mod y$, because the latter is difficult to format in subscripts. The $\%$ symbol is used to represent the modulo operation in Python.

I represent the set of scenarios as Ω and a specific scenario as ω. Most variables (excluding the policy parameters) can have a different value in each scenario, for example $D_t(\omega)$ represents the demand on day t in scenario ω. Each of the 20 scenarios is equally likely, and therefore $p(\omega)$ for each scenario is $\frac{1}{20}$.

The starting inventory is always the same: 36 units with a remaining useful life of two days (as if they had been delivered on the preceding Sunday), with no order arriving on the Monday morning. This value was calculated following private correspondence with the authors of the original paper [45], in which they explained that they had used a starting inventory equal to the mean daily demand over the seven weekdays.

D.1 Indices and Sets

In Table D.1 I present the indices and sets common to all of the stochastic programming approaches.

Notation	Description
$a \in \mathcal{A}$	Set of platelet age (shelf life in days)
$t \in \mathcal{T}$	Set of time periods (in days)
$k \in \mathcal{K}$	Set of days of the week
$\omega \in \Omega$	Set of scenarios

Table D.1: Indices and sets for stochastic programming.

D.2 Parameters

In Table D.2 I present the parameters common to all of the stochastic programming approaches.

D.3 Common decision variables

In Table D.3 I present the decision variables common to all of the stochastic programming approaches. The binary decision variables $\phi_{t,a}(\omega)$ were added to

Notation	Description
$D_t(\omega)$	Platelet demand for day t and scenario ω
$I_a^0(\omega)$	Initial inventory with a shelf life of a days
$p(\omega)$	Probability of occurrence of scenario ω
LT	Constant lead time (in days)
C^F	Fixed transportation costs of procuring platelets ($/shipment)
C^P	Platelet purchasing cost ($/unit)
C^H	Inventory holding cost ($/unit/day)
C^W	Wastage cost ($/unit)
C^E	Emergency procurement cost ($/unit)
M	A large positive number

Table D.2: Parameters for stochastic programming.

enforce the OUFO issuing policy, as described below in Section D.4.

D.4 Modifications

As noted in Section 3.4.5 I modified the constraints on the policy parameter decision variables to make it explicit that one of each parameter should be learned for each weekday. The terms for these decision variables in the constraints were amended to replace index t with $t\%7$.

In my initial experiments, I found that the constraints as set out in Rajendran and Srinivas [45] were insufficient to enforce the OUFO rule for issuing units. The problematic constraints presented here as Equations D.5 and D.6 are their Equations 7 and 8 respectively.

These constraints only enforce an OUFO policy if only one term on the right hand side of each equation is allowed to be non-zero. If there is remaining demand after using all stock of a certain age, there cannot be any stock of that age remaining. If there is stock of a certain age remaining then there cannot be unfulfilled demand after using the stock of that age. I therefore introduced additional binary decision variables and constraints, based on those from Pauls-Worm *et al.* [173], to ensure that the intended OUFO issuing policy was followed. In these additional constraints $\phi_{t,a}(\omega)$ is a binary decision variable that is 1 if there is remaining demand on day t after using product with shelf life of a days for scenario ω, and M is a large

Notation	Description
$OQ_t(\omega)$	Units ordered at the end of day t for scenario ω
$X_{t,a}(\omega)$	Total units received by the hospital at the beginning of day t with shelf life of a days for scenario ω
$D^R_{t,a}(\omega)$	Remaining demand on day t after using product with shelf life of a days for scenario ω
$I^B_{t,a}(\omega)$	On-hand inventory at the beginning of day t with shelf life of a days for scenario ω
$I^E_{t,a}(\omega)$	Inventory at the end of day t with shelf life of a days for scenario ω
$IP_t(\omega)$	Inventory position at the end of day t for scenario ω
$E_t(\omega)$	Number of units obtained through emergency procurement on day t for scenario ω
$W_t(\omega)$	Number of units wasted (expired) at the end of day t for scenario ω
$\Delta_t(\omega)$	1 if $IP_t(\omega) < s_{t\%7}$ and 0 otherwise
$F_t(\omega)$	1 if a platelet or is placed on day t for scenario ω
$\phi_{t,a}(\omega)$	1 if $D^R_{t,a} > 0$ and 0 otherwise
s_k	Re-order point for weekday k, $k = t\%7$

Table D.3: Common decision variables for stochastic programming.

positive number. The additional constraints are presented below as Equations D.17 and D.18.

The addition of these decision variables and corresponding constraints enlarges problem and makes it more difficult to solve: each of these constraints has to be enforced for each day of remaining useful life, for each day, for each scenario and includes an additional integer decision variable. However, if an OUFO issuing policy is not enforced in optimization, but is used when the replenishment policy deployed, the fitted parameters found by the stochastic programming method will have been optimized to solve a different problem to that intended and may not

perform well.

I also used a corrected version of Equation 9 from Rajendran and Srinivas [45] so that the summation over the $X_{t,a}(\omega)$ terms is not over all $t \in \mathcal{T}$. This was confirmed to be a typographical error in the published paper in private correspondence with the authors. The corrected version is presented here as Equation D.7. There is no lead time in the baseline scenario: orders placed in the evening of day $t-1$ are received on the morning of day t and therefore the second two terms will always cancel out to the simplified version equivalent to Equation 3.3.

D.5 Objective function and common constraints

D.5.1 Objective function

$$\text{cost} = \sum_{\omega\in\Omega}\sum_{t\in T} p(\omega)\left(C^F F_t(\omega) + C^P OQ_t(\omega) + C^H \sum_{a\in\mathcal{A}\ni a>1} I^E_{t,a}(\omega) + C^W W_t(\omega) + C^E E_t(\omega)\right) \tag{D.1}$$

D.5.2 Common constraints

$$OQ_t(\omega) \le M \times F_t(\omega) \tag{D.2}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$\sum_{a\in\mathcal{A}} X_{t,a}(\omega) = OQ_{t-LT}(\omega) \tag{D.3}$$
$$t \in \mathcal{T} \ni t > LT, \quad \omega \in \Omega$$

$$\sum_{a\in\mathcal{A}} X_{t,a}(\omega) = 0 \text{ or known constants} \tag{D.4}$$
$$t \in \mathcal{T} \ni t \le LT, \quad \omega \in \Omega$$

$$D_t(\omega) - I^B_{t,a}(\omega) - X_{t,a}(\omega) = D^R_{t,a}(\omega) - I^E_{t,a}(\omega) \tag{D.5}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A} \ni a = 1, \quad \omega \in \Omega$$

$$D^R_{t,a-1}(\omega) - I^B_{t,a}(\omega) - X_{t,a}(\omega) = D^R_{t,a}(\omega) - I^E_{t,a}(\omega) \tag{D.6}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A} \ni a > 1, \quad \omega \in \Omega$$

$$IP_t(\omega) = \sum_{a \in \mathcal{A} \ni a > 1} I^E_{t,a}(\omega) + \sum_{t' \in \mathcal{T} \ni t' \leq t-1} OQ_{t'}(\omega) - \sum_{t' \in \mathcal{T} \ni t' \leq t} \sum_{a \in \mathcal{A}} X_{t,a}(\omega) \tag{D.7}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$IP_t(\omega) \leq (s_{t\%7} - 1) + M \times (1 - \Delta_t(\omega)) \tag{D.8}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$IP_t(\omega) \geq s_{t\%7} - M \times \Delta_t(\omega) \tag{D.9}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$I^B_{t+1,a}(\omega) = I^E_{t,a+1}(\omega) \tag{D.10}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A} \ni a < |\mathcal{A}|, \quad \omega \in \Omega$$

$$E_t(\omega) = D^R_{t,a}(\omega) \tag{D.11}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A} \ni a = |\mathcal{A}|, \quad \omega \in \Omega$$

$$W_t(\omega) = I^E_{t,a}(\omega) \tag{D.12}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A} \ni a = 1, \quad \omega \in \Omega$$

$$I^B_{t,a}(\omega) = 0 \tag{D.13}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A} \ni a = |\mathcal{A}|, \quad \omega \in \Omega$$

$$I^B_{t,a}(\omega) = I^0_a(\omega) \tag{D.14}$$
$$t = 1, \quad a \in \mathcal{A} \ni a < |\mathcal{A}|, \quad \omega \in \Omega$$

$$\Delta_t(\omega),\ F_t(\omega), \phi_{t,a}(\omega) \in \{0, 1\} \tag{D.15}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A}, \quad \omega \in \Omega$$

$$s_k,\ OQ_t(\omega),\ X_{t,a}(\omega),\ D^R_{t,a}(\omega),\ I^B_{t,a}(\omega),\ I^E_{t,a}(\omega),\ IP_t(\omega),\ W_t(\omega),\ E_t(\omega) \geq 0 \tag{D.16}$$
$$t \in \mathcal{T}, \quad k \in \mathcal{K}, \quad a \in \mathcal{A}, \quad \omega \in \Omega$$

In addition to the common constraints above from [45], I introduce two new constraints to enforce the OUFO issuing policy as explained in Section D.4:

$$M \times \phi_{t,a}(\omega) \geq D^R_{t,a}(\omega) \tag{D.17}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A}, \quad \omega \in \Omega$$

$$M \times (1 - \phi_{t,a}(\omega)) \geq I^E_{t,a}(\omega) \tag{D.18}$$
$$t \in \mathcal{T}, \quad a \in \mathcal{A}, \quad \omega \in \Omega$$

D.6 Additional decision variables and constraints for the $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy

I present the additional decision variables for the $(\mathtt{s}, \mathtt{S})^{\dagger}$ replenishment policy in Table D.4 and the additional constraints in Equations D.19 to D.23.

Notation	Description
S_k	Order-up-to-level for weekday k, $k = t\%7$

Table D.4: Additional decision variables for the $(\mathtt{s}, \mathtt{S})^{\dagger}$ policy.

$$S_k \geq s_k + 1 \qquad k \in \mathcal{K} \tag{D.19}$$

$$OQ_t(\omega) \leq (S_{t\%7} - IP_t(\omega)) + M \times (1 - \Delta_t(\omega)) \qquad t \in \mathcal{T}, \quad \omega \in \Omega \tag{D.20}$$

$$OQ_t(\omega) \geq (S_{t\%7} - IP_t(\omega)) - M \times (1 - \Delta_t(\omega)) \qquad t \in \mathcal{T}, \quad \omega \in \Omega \tag{D.21}$$

$$OQ_t(\omega) \leq M \times \Delta_t(\omega) \qquad t \in \mathcal{T}, \quad \omega \in \Omega \tag{D.22}$$

$$S_k \geq 0 \qquad k \in \mathcal{K} \tag{D.23}$$

D.7 Additional decision variables and constraints for the $(\mathtt{s}, \mathtt{Q})$ policy

I present the additional decision variables for the $(\mathtt{s}, \mathtt{Q})$ replenishment policy in Table D.5 and the additional constraints in Equations D.24 to D.27.

Notation	Description
Q_k	Fixed order quantity for weekday k, $k = t\%7$

Table D.5: Additional decision variables for the $(\mathtt{s}, \mathtt{Q})$ policy.

$$OQ_t(\omega) \leq Q_{t\%7} + M \times (1 - \Delta_t(\omega)) \qquad \text{(D.24)}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \geq Q_{t\%7} - M \times (1 - \Delta_t(\omega)) \qquad \text{(D.25)}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \leq M \times \Delta_t(\omega) \qquad \text{(D.26)}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$Q_k \geq 0 \qquad \text{(D.27)}$$
$$k \in \mathcal{K}$$

D.8 Additional decision variables and constraints for $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ policy

I present the additional decision variables for the $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ replenishment policy in Table D.6 and the additional constraints in Equations D.28 to D.35.

Notation	Description
δ_t	1 if $IP_t(\omega) < \alpha_{t\%7}$ and 0 otherwise
α_k	Ordering quantity threshold parameter for weekday k, $k = t\%7$
S_k	Order-up-to-level for weekday k, $k = t\%7$
Q_k	Fixed order quantity for weekday k, $k = t\%7$

Table D.6: Additional decision variables for $(\mathtt{s}, \mathtt{S}, \alpha, \mathtt{Q})$ policy.

$$IP_t(\omega) \leq (\alpha_{t\%7} - 1) + M \times (1 - \delta_t(\omega)) \tag{D.28}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$IP_t(\omega) \geq \alpha_{t\%7} - M \times \delta_t(\omega) \tag{D.29}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) = \max\left([Q_{t\%7} \times (1 - \delta_t(\omega)) \times \Delta_t(\omega)], [(S_{t\%7} - IP_t(\omega)) \times \delta_t(\omega) \times \Delta_t(\omega)]\right) \tag{D.30}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

The constraint on the order quantity in Equation D.30 is non-linear. It is therefore replaced with the linear constraints in Equations D.31a to D.31e.

$$OQ_t(\omega) \leq Q_{t\%7} + M \times \delta_t(\omega) + M \times (1 - \Delta_t(\omega)) \tag{D.31a}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \geq Q_{t\%7} - M \times \delta_t(\omega) - M \times (1 - \Delta_t(\omega)) \tag{D.31b}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \leq (S_{t\%7} - IP_t(\omega)) + M \times (1 - \delta_t(\omega)) + M \times (1 - \Delta_t(\omega)) \quad \text{(D.31c)}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \geq (S_{t\%7} - IP_t(\omega)) - M \times (1 - \delta_t(\omega)) - M \times (1 - \Delta_t(\omega)) \quad \text{(D.31d)}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \leq M \times \Delta_t(\omega) \quad \text{(D.31e)}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$s_k \geq \alpha_k + 1 \quad \text{(D.32)}$$
$$k \in \mathcal{K}$$

$$S_k \geq s_k + 1 \quad \text{(D.33)}$$
$$k \in \mathcal{K}$$

$$\delta_t(\omega) \in \{0, 1\} \quad \text{(D.34)}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$S_k,\ \alpha_k,\ Q_k \geq 0 \quad \text{(D.35)}$$
$$k \in \mathcal{K}$$

D.9 Additional decision variables and constraints for the $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policy

I present the additional decision variables for the $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ replenishment policy in Table D.7 and the additional constraints in Equations D.36 to D.43.

Notation	Description
ν_t	1 if $IP_t(\omega) < \beta_{t\%7}$ and 0 otherwise
β_k	Ordering quantity threshold parameter for weekday k, $k = t\%7$
S_k	Order-up-to-level for weekday k, $k = t\%7$
Q_k	Fixed order quantity for weekday k, $k = t\%7$

Table D.7: Additional decision variables for the $(\mathtt{s}, \mathtt{S}, \beta, \mathtt{Q})$ policy.

$$IP_t(\omega) \leq (\beta_{t\%7} - 1) + M \times (1 - \nu_t(\omega)) \tag{D.36}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$IP_t(\omega) \geq \beta_{t\%7} - M \times \nu_t(\omega) \tag{D.37}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) = \max\left([Q_{t\%7} \times \nu_t(\omega) \times \Delta_t(\omega)], [(S_{t\%7} - IP_t(\omega)) \times (1 - \nu_t(\omega)) \times \Delta_t(\omega)]\right) \tag{D.38}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

The constraint on the order quantity in Equation D.38 is non-linear. It is therefore replaced with the linear constraints in Equations D.39a to D.39e.

$$OQ_t(\omega) \leq Q_{t\%7} + M \times (1 - \nu_t(\omega)) + M \times (1 - \Delta_t(\omega)) \tag{D.39a}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \geq Q_{t\%7} - M \times (1 - \nu_t(\omega)) - M \times (1 - \Delta_t(\omega)) \tag{D.39b}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \leq (S_{t\%7} - IP_t(\omega)) + M \times \nu_t(\omega) + M \times (1 - \Delta_t(\omega)) \tag{D.39c}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \geq (S_{t\%7} - IP_t(\omega)) - M \times \nu_t(\omega) - M \times (1 - \Delta_t(\omega)) \tag{D.39d}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$OQ_t(\omega) \leq M \times \Delta_t(\omega) \tag{D.39e}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$s_k \geq \beta_k + 1 \tag{D.40}$$
$$k \in \mathcal{K}$$

$$S_k \geq s_k + 1 \tag{D.41}$$
$$k \in \mathcal{K}$$

$$\nu_t(\omega) \in \{0, 1\} \tag{D.42}$$
$$t \in \mathcal{T}, \quad \omega \in \Omega$$

$$S_k,\ \beta_k,\ Q_k \geq 0 \tag{D.43}$$
$$k \in \mathcal{K}$$

Appendix E

Chapter 3 supplement: Configuration files for deep reinforcement learning training

E.1 DQN

This yaml hydra configuration file was used with the training script available at https://github.com/joefarrington/bloodbank_rl/blob/main/bloodbank_rl/rl_training/dqn/train_dqn.py to train the DQN policy.

```yaml
hydra_logdir: ${hydra:output_subdir}
device: cuda
env:
  _target_: bloodbank_rl.environments.platelet_bankSR.PlateletBankGym
  demand_provider:
    _target_: bloodbank_rl.environments.platelet_bankSR.PoissonDemandProviderSR
    sim_duration: 365
    one_hot_encode_weekday: true
  max_order: 60
  max_shelf_life: 3
  lead_time: 0
  shelf_life_at_arrival_dist:
  - 0
  - 0
  - 1
  fixed_order_cost: 225
  variable_order_cost: 650
  holding_cost: 130
  emergency_procurement_cost: 3250
```

```yaml
  wastage_cost: 650
vec_env:
  n_train_envs: 1
  train_env_seed: 11
  n_test_envs: 1
  test_env_seed: 12
model:
  _target_: bloodbank_rl.tianshou_utils.models.FCDQN
  n_hidden:
  - 128
  - 128
  - 128
optimiser:
  _target_: torch.optim.Adam
  lr: 1.0e-05
policy:
  _target_: tianshou.policy.DQNPolicy
  discount_factor: 0.75
  estimation_step: 1
  target_update_freq: 1000
  is_double: false
train_collector:
  _target_: tianshou.data.Collector
  buffer:
    _target_: tianshou.data.VectorReplayBuffer
    total_size: 1000000
    buffer_num: ${vec_env.n_train_envs}
  exploration_noise: true
test_collector:
  _target_: tianshou.data.Collector
  exploration_noise: true
logger:
  _target_: bloodbank_rl.tianshou_utils.logging.TianshouMLFlowLogger
  filename: train_dqn.py
  experiment_name: dqn_tianshou
  info_logger:
    _target_: bloodbank_rl.tianshou_utils.logging.InfoLogger
  model_checkpoints: true
  cp_path: ${hydra:output_subdir}/model_checkpoints
n_steps_before_learning: 10000
trainer:
  _target_: tianshou.trainer.offpolicy_trainer
  max_epoch: 200
  step_per_epoch: 5000
  step_per_collect: 1
  update_per_step: 1
```

```
  episode_per_test: 100
  batch_size: 64
  verbose: false
exploration:
  _target_: bloodbank_rl.tianshou_utils.exploration.EpsilonScheduler
  max_epoch: ${trainer.max_epoch}
  step_per_epoch: ${trainer.step_per_epoch}
  eps_max: 1
  eps_min: 0.1
  exploration_fraction: 0.5
seed: 57
```

E.2 PPO

This yaml hydra configuration file was used with the training script available at https://github.com/joefarrington/bloodbank_rl/blob/main/bloodbank_rl/rl_training/ppo/train_ppo.py to train the PPO policy.

```
hydra_logdir: ${hydra:output_subdir}
device: cuda
env:
  _target_: bloodbank_rl.environments.platelet_bankSR.PlateletBankGym
  demand_provider:
    _target_: bloodbank_rl.environments.platelet_bankSR.PoissonDemandProviderSR
    sim_duration: 365
    one_hot_encode_weekday: true
  max_order: 60
  max_shelf_life: 3
  lead_time: 0
  shelf_life_at_arrival_dist:
  - 0
  - 0
  - 1
  fixed_order_cost: 225
  variable_order_cost: 650
  holding_cost: 130
  emergency_procurement_cost: 3250
  wastage_cost: 650
vec_env:
  n_train_envs: 5
  train_env_seed:
  - 15
  - 16
  - 17
```

```
  - 18
  - 19
  n_test_envs: 1
  test_env_seed: 12
actor_network:
  _target_: tianshou.utils.net.common.Net
  hidden_sizes:
  - 128
  - 128
  - 128
  device: ${device}
critic_network:
  _target_: tianshou.utils.net.common.Net
  hidden_sizes:
  - 128
  - 128
  - 128
  device: ${device}
optimiser:
  _target_: torch.optim.RMSprop
  lr: 0.0001
  eps: 1.0e-05
  alpha: 0.99
policy:
  _target_: bloodbank_rl.tianshou_utils.policies.PPOPolicyforMLFlow
  discount_factor: 0.95
  eps_clip: 0.2
  dual_clip: 5
  value_clip: true
  advantage_normalization: true
  recompute_advantage: false
  vf_coef: 0.75
  ent_coef: 0.01
  max_grad_norm: 0.5
  gae_lambda: 0.95
  reward_normalization: true
  max_batchsize: 256
  action_scaling: true
  action_bound_method: clip
  deterministic_eval: true
train_collector:
  _target_: tianshou.data.Collector
  buffer:
    _target_: tianshou.data.VectorReplayBuffer
    total_size: 20000
    buffer_num: ${vec_env.n_train_envs}
```

```yaml
  exploration_noise: true
test_collector:
  _target_: tianshou.data.Collector
  exploration_noise: true
logger:
  _target_: bloodbank_rl.tianshou_utils.logging.TianshouMLFlowLogger
  filename: train_ppo.py
  experiment_name: ppo_tianshou
  update_interval: 1
  info_logger:
    _target_: bloodbank_rl.tianshou_utils.logging.InfoLogger
checkpoints:
  save_checkpoints: true
  path: ./tmp_logs/model_checkpoints/${now:%Y-%m-%d}/${now:%H-%M-%S}
  training_checkpoint_interval: 1
trainer:
  _target_: tianshou.trainer.onpolicy_trainer
  max_epoch: 100
  step_per_epoch: 10000
  repeat_per_collect: 10
  episode_per_test: 100
  batch_size: 9999
  step_per_collect: 2000
  test_in_train: false
seed: 57
```

Appendix F

Chapter 4 supplement: Scenario descriptions in common notation

F.1 Scenario A

In this section I recast the problem formulated by De Moor *et al.* [40] into a consistent notation used for scenarios A, B and C.

The state of the system, S_t, comprises two components: the orders in-transit $\underline{\mathrm{X}}_t^{\mathrm{tr}}$ and the units in stock $\underline{\mathrm{X}}_t$:

$$\underline{\mathrm{X}}_t^{\mathrm{tr}} = \left[X_{L-1,t}^{\mathrm{tr}} = A_{t-1}, X_{L-2,t}^{\mathrm{tr}}, ..., X_{1,t}^{\mathrm{tr}}\right] \tag{F.1}$$

$$\underline{\mathrm{X}}_t = \left[X_{m,t} = X_{1,t-1}^{\mathrm{tr}}, X_{m-1,t}, ..., X_{1,t}\right] \tag{F.2}$$

for a total of $(m + L - 1)$ elements, with lead time $L \geq 1$. The total number of possible states is therefore $(A_{\max} + 1)^{m+L-1}$. In the state $S_t = \left[\underline{\mathrm{X}}_t^{\mathrm{tr}}, \underline{\mathrm{X}}_t\right]$, the entries are ordered by ascending age: the first element is the order placed on day $t - 1$ and the last element is the stock that will expire at the end of the current day. The total number of units in stock at the start of day t is $X_t = \sum_{i=1}^{m} X_{i,t}$, the total number of units in-transit at the start of day t is $X_t^{\mathrm{tr}} = \sum_{i=1}^{L-1} X_{i,t}^{\mathrm{tr}}$. The total number of units in stock or in-transit at the start of day t is $I_t = X_t + X_t^{\mathrm{tr}}$. If $L \leq 1$ there is no in-transit component to the state, and the first element of $\underline{\mathrm{X}}_t$ is A_{t-1}. In Table F.1 I present the input parameter values that are the same for all of the experiments for Scenario A.

	$D_{\max}$	$A_{\max}$	C_v	C_s	C_h	μ	$\frac{\mu}{\sigma}$	γ	ϵ
Value	100	10	3	5	1	4	0.5	0.99	1×10^{-4}

Table F.1: Input parameter values that are consistent for all of the experiments for Scenario A.

Daily demand, the stochastic element in the transition, is modelled using a truncated gamma distribution. It does not depend on the state or the action. The demand for the product is discrete and the gamma distribution is continuous, so the probability that the daily demand is equal to $d \in \{0, 1, ..., D_{\max}\}$ is:

$$\begin{aligned} \text{Prob}(\Omega = \omega | S = s, A = a) &= P(\Omega = d) \\ &= P(D = d) \\ &= \begin{cases} F\left(d + \frac{1}{2}; \mu, \frac{\mu}{\sigma}\right) \\ \quad -F\left(d - \frac{1}{2}; \mu, \frac{\mu}{\sigma}\right), & \text{if } d \in \{0, 1, ..., D_{\max} - 1\} \\ 1 - F\left(D_{\max} - \frac{1}{2}; \mu, \frac{\mu}{\sigma}\right), & \text{if } d = D_{\max} \end{cases} \end{aligned} \tag{F.3}$$

where $F(x; \mu, \frac{\mu}{\sigma})$ is the cumulative distribution function of the gamma distribution parameterized by mean μ and coefficient of variation $\frac{\mu}{\sigma}$, and $F(x; \mu, \frac{\mu}{\sigma}) = 0$ when $x \leq 0$.

The reward function comprises four components: a holding cost per unit in stock at the end of the period (C_h), a variable ordering cost per unit (C_v), a shortage cost per unit of unmet demand (C_s) and a wastage cost per unit that perishes at the end of the period (C_w). The single-step reward after taking action A_t in state S_t with $\Omega_t = (D_t)$ is:

$$R_{t+1} = -C_v A_t - C_h \left[X_t - D_t - W_t\right]^+ - C_s \left[D_t - X_t\right]^+ - C_w W_t \tag{F.4}$$

where W_t is the number of units that expire at the end of period t.

Equations F.5 and F.6 set out how the number of expired units, W_t, is calculated

and how the elements of $\underline{X_t}$ are updated when following a FIFO issuing policy and a LIFO issuing policy, respectively.

$$
\begin{aligned}
W_t &= [X_{1,t} - D_t]^+ \\
X_{j,t+1} &= \left[X_{j+1,t} - \left[D_t - \sum_{k=1}^{j} X_{k,t}\right]^+\right]^+ \qquad \forall j \in \{1, 2, ..., m-1\} \\
X_{m,t+1} &= X^{\text{tr}}_{1,t} = A_{t-L+1}
\end{aligned}
\tag{F.5}
$$

$$
\begin{aligned}
W_t &= \left[X_{1,t} - \left[D_t - \sum_{k=2}^{m} X_{k,t}\right]^+\right]^+ \\
X_{j,t+1} &= \left[X_{j+1,t} - \left[D_t - \sum_{k=j+2}^{m} X_{k,t}\right]^+\right]^+ \qquad \forall j \in \{1, 2, ..., m-1\} \\
X_{m,t+1} &= X^{\text{tr}}_{1,t} = A_{t-L+1}
\end{aligned}
\tag{F.6}
$$

The scenario is an infinite horizon MDP with a discount factor and no periodicity in the state space. I therefore used a standard convergence test for the value function [97], evaluating:

$$\max_{s \in \mathbb{S}} |V_j(s) - V_{j-1}(s)| < \epsilon \tag{F.7}$$

after each iteration. The inequality tests for the convergence of the values themselves, and requires more iterations than the convergence tests used for the other scenarios which are testing for convergence of the change in value for each state. The test compares the current estimate of the value function with the estimate from the immediately preceding iteration and does not require previous checkpoints for evaluation. Therefore, to save storage space and writing time, I saved a checkpoint every 100 iterations.

F.2 Scenario B

In this section I recast the problem formulated by Hendrix *et al.* [71] into a consistent notation used for scenarios A, B and C.

The state of the environment, S_t comprises two components, one for each product type. In the combined state $S_t = \left[\underline{\mathrm{X}}_t^a, \underline{\mathrm{X}}_t^b\right]$, the elements in each component are ordered by ascending age:

$$\underline{\mathrm{X}}_t^a = [X_{m,t}^a = A_{t-1}^a, X_{m-1,t}^a, ..., X_{1,t}^a] \tag{F.8}$$

$$\underline{\mathrm{X}}_t^b = [X_{m,t}^b = A_{t-1}^b, X_{m-1,t}^b, ..., X_{1,t}^b] \tag{F.9}$$

for a total number of $2m$ elements. The total number of possible states is therefore $(A_{\max}^a + 1)^m + \left(A_{\max}^b + 1\right)^m$. The total number of units in stock at the start of period t is $I_t^a = X_t^a = \sum_{i=1}^m X_{i,t}^a$ for product A and $I_t^b = X_t^b = \sum_{i=1}^m X_{i,t}^b$ for product B. In Table F.2 I present the parameter values that are the same for all of the experiments for Scenario B.

	C_v^a	C_v^b	C_r^a	C_r^b	ρ	γ	ϵ
Value	0.5	0.5	1.0	1.0	0.5	1.0	1×10^{-4}

Table F.2: Input parameter values that are consistent for all of the experiments for Scenario B.

The stochastic element of the transition is the number of products of each type issued, (H^a, H^b). The number of units of product B issued only depends on the demand for product B and the total stock of product B, but the number of units of product A that are issued depends on the demand for product A, the total stock of product A, and any excess demand for product B for which the customer is willing to accept product A.

Let the demand for product A be D^a, the demand for product B be D^b, the excess demand for product B where the customer is willing to accept product A be D^u and the total demand for product A including any substitution be $D^z = D^a + D^u$. To calculate the probability of a combination $\omega = (h^a, h^b)$ given a particular state s, it is necessary to consider five possible cases:

$$
\begin{aligned}
\text{Prob}\left(\Omega=\omega|S=s,A=a\right) &= P\left(\Omega=\left(h^a,h^b\right)|S=s\right) \qquad \text{(F.10)}\\
&= P\left(H^a=h^a,H^b=h^b|S=s\right)\\
&= \begin{cases}
0, & \text{if } h^a>I^a \text{ or } h^b>I^b\\
P(D^a=h^a)P(D^b=h^b), & \text{if } h^a<I^a \text{ and } h^b<I^b\\
P(D^a\geq I^a)P(D^b=h^b), & \text{if } h^a=I^a \text{ and } h^b<I^b\\
P(D^z=h^a|S=s)P(D^b\geq I^b), & \text{if } h^a<I^a \text{ and } h^b=I^b\\
P(D^z\geq I^a)P(D^b\geq I^b), & \text{if } h^a=I^a \text{ and } h^b=I^b
\end{cases}\\
&= \begin{cases}
0, & \text{if } h^a>I^a \text{ or } h^b>I^b\\
P(h^a;\mu^a)P(h^b;\mu^b), & \text{if } h^a<I^a \text{ and } h^b<I^b\\
\left[1-F\left(I^a-1;\mu^a\right)\right]P(h^b;\mu^b), & \text{if } h^a=I^a \text{ and } h^b<I^b\\
P(D^z=h^a|S=s)\left[1-\left(F(I^b-1;\mu^b\right)\right], & \text{if } h^a<I^a \text{ and } h^b=I^b\\
\left[1-\sum_{d=0}^{I^a-1}P(D^z=d|S=s)\right]\left[1-\left(F(I^b-1;\mu^b\right)\right], & \text{if } h^a=I^a \text{ and } h^b=I^b
\end{cases}
\end{aligned}
$$

For the fourth and fifth cases there may be substitution, and therefore it is necessary to consider the distribution of the total demand for product A and the distribution of the demand for substitution:

$$
\begin{aligned}
P(D^z=d^z|S=s) &= P(D^z=d^z|I^b=y)\\
&= \sum_{k=0}^{d^z}P(D^a=k)P(D^u=d^z-k|I^b=y,D^b\geq y) \qquad \text{(F.11)}\\
&= \sum_{k=0}^{d^z}P(k;\mu^a)P(D^u=d^z-k|I^b=y,D^b\geq y)
\end{aligned}
$$

$$P(D^u = d^u | I^b = y, D^b \geq y) = \begin{cases} \sum_{c=0}^{\infty} P(D^b = c + y)(1-\rho)^c, & \text{if } d^u = 0 \\ \sum_{c=d^u}^{\infty} P(D^b = c + y) P(d^u; c, \rho), & \text{if } d^u > 0 \end{cases} \tag{F.12}$$

$$= \begin{cases} \sum_{c=0}^{\infty} P(c + y; \mu^b)(1-\rho)^c, & \text{if } d^u = 0 \\ \sum_{c=d^u}^{\infty} P(c + y; \mu^b) P(d^u; c, \rho), & \text{if } d^u > 0 \end{cases}$$

where $P(x; c, \rho)$ is a binomial probability mass function representing the probability that there are x units of excess demand for product B willing to accept product A out of a total of c units of excess demand for product B and the probability of being willing to accept the substitution is ρ. $P(x; \mu^a)$ and $P(x; \mu^b)$ are the probability mass functions of independent Poisson distributions for the daily demand of product A and B parameterized by mean daily demands μ^a and μ^b respectively, and $F(x; \mu^a)$ and $F(x; \mu^b)$ are the corresponding cumulative distribution functions.

I calculated the values of $P(D^u = d^u | I^b = y, D^b \geq y)$ and $P(D^z = d^z | S = s)$, for $d^u \in \{0, 1, ..., D_{\max}\}$ and $d^z \in \{0, 1, ..., D_{\max}\}$, where $D_{\max} = \Big(\Big(m \max(A^a_{\max}, A^b_{\max})\Big) + 2\Big)$, at the start of value iteration following the MATLAB implementation of Hendrix *et al.* [71].

The reward function comprises two components which can be different for each product: a variable ordering cost per unit (C^a_v, C^b_v) and revenue per unit sold (C^a_r, C^b_r). The single step reward after taking action A_t in state S_t with $\Omega_t = \left(H^a_t, H^b_t\right)$ is:

$$R_{t+1} = -\left(C^a_v A^a_t + C^a_v A^b_t\right) + \left(C^a_r H^a_t + C^b_r H^b_t\right) \tag{F.13}$$

Equation F.14 shows how the elements of $\underline{\mathrm{X}^a_t}$ and $\underline{\mathrm{X}^b_t}$ are updated following a FIFO issuing policy.

$$X^a_{j,t+1} = \left[X^a_{j+1,t} - \left[H^a_t - \sum_{k=1}^{j} X^a_{k,t} \right]^+ \right]^+ \quad \forall j \in \{1, 2, ..., m-1\} \tag{F.14}$$

$$X^a_{m,t+1} = A^a_t$$

$$X^b_{j,t+1} = \left[X^b_{j+1,t} - \left[H^b_t - \sum_{k=1}^{j} X^b_{k,t} \right]^+ \right]^+ \quad \forall j \in \{1, 2, ..., m-1\}$$

$$X^b_{m,t+1} = A^b_t$$

The maximum order quantities for value iteration, $A^a_{\max}$ and $A^b_{\max}$ are calculated independently for each product following the newsvendor model [28]:

$$A^k_{\max} = \left\lceil F^{-1}\left(\frac{C^k_r - C^k_v}{C^k_r}; m\mu^k \right) \right\rceil^+, \quad \forall k \in \{a, b\} \tag{F.15}$$

where $F(x; m\mu^k)$ is the cumulative distribution function of a Poisson distribution parameterized by $m\mu^k$ and μ^k is the mean daily demand for product k.

The maximum order quantities reported for experiments P1 to P4 in Table 3 of Ortega *et al.* [196] are not consistent with the number of states reported in that table. I have assumed that they instead represent the number of actions (one higher than the maximum order quantity, due to the possibility of ordering zero units) as this is consistent with the number of states reported, with Table 1 of Ortega *et al.* [196] and, for experiment P1, with the corresponding experiment in Hendrix *et al.* [71].

The initial estimate for the value function is the expected one-step ahead sales revenue:

$$V_0(s) = \sum_{h^a=0}^{I^a_t} \sum_{h^b=0}^{I^b_t} P(H^a = h^a, H^b = h^b | S = s)(h^a C^a_r + h^b C^b_r) \tag{F.16}$$

I used the same convergence test as Hendrix *et al.* [71], evaluating:

$$\max_{s \in \mathbb{S}} \left[V_j(s) - V_{j-1}(s) \right] - \min_{s \in \mathbb{S}} \left[V_j(s) - V_{j-1}(s) \right] < \epsilon \tag{F.17}$$

after each iteration. The inequality tests for the convergence of the change in value for each state. When the value of each state is changing by the same amount, the best action for each state will not change and therefore the estimate of the optimal policy is stable. I saved a checkpoint after every iteration.

F.3 Scenario C

In this section I recast the problem formulated by Mirjalili [166] into a consistent notation used for scenarios A, B and C.

The state of the environment, S_t, comprises two components: $\tau \in \{0, 1, ..., 6\}$, representing the day of the week, and the units in stock at the start of the day $\underline{\mathrm{X}_t} = [X_{m-1,t}, X_{m-2,t}, ..., X_{1,t}]$. The lead time, L, is always zero which means that the units ordered on day t are received before any demand arises on day t. There are therefore only $m-1$ elements in $\underline{\mathrm{X}_t}$ and a total of m elements, including τ, in S_t.

In the previous problems, the maximum value of each element of $\underline{\mathrm{X}_t}$ was $A_{\max}$, because all units arrived with the same remaining useful life. All units received in the same period would be in the same element of $\underline{\mathrm{X}_t}$. In this scenario, the remaining useful life on arrival is stochastic and therefore, depending on the policy, a series of orders could be received such that an element of $\underline{\mathrm{X}_t}$ would exceed $A_{\max}$. I assume there is a maximum capacity of $A_{\max}$ for stock of each possible value of remaining useful life. Units received in excess of this limit are not accepted at the point of delivery. The total number of possible states is therefore $7 \times (A_{\max}+1)^{m-1}$. The entries in $\underline{\mathrm{X}_t}$ are ordered by ascending age: the first element represents stock with $m-1$ days before expiry, and the last element is the stock that will expire at the end of day t. In Table F.3 I present the input parameter values that are the same for all of the experiments for Scenario C.

	$D_{\max}$	$A_{\max}$	C_f	C_h	C_s	C_w	γ	ϵ
Value	20	20	10	1	20	5	0.95	1×10^{-4}

Table F.3: Input parameter values that are consistent for all of the experiments for Scenario C.

The stochastic elements in the transition are the daily demand D, and the age profile of the units received: $\underline{\mathrm{Y}} = [Y_m, Y_{m-1}, ..., Y_1]$.

The probability of a given random outcome ω is the product of the probability of the demand given the state, and the probability of receiving units with a specific age profile given the action:

$$\begin{aligned}\text{Prob}\left(\Omega=\omega|S=s,A=a\right) &= P\left(\Omega=\left(d,\underline{\mathrm{y}}\right)|S=s,A=a\right) \\ &= P\left(D=d,\underline{\mathbf{Y}}=\underline{\mathrm{y}}|S=s,A=a\right) \\ &= P\left(D=d|S=s\right)P\left(\underline{\mathbf{Y}}=\underline{\mathrm{y}}|A=a\right)\end{aligned} \tag{F.18}$$

Demand is modelled by truncated negative binomial distributions, one for each day of the week. The demand distribution therefore only depends on the weekday element of the state. The negative binomial distribution models the number of failures, x, in a series of repeated Bernoulli trials before achieving a specified number of successes. The probability that daily demand is equal to d on weekday τ is:

$$P(D=d|S=s)=\begin{cases}P(d;n^{\tau},\delta^{\tau}), & \text{if } d\in\{0,1,...,D_{\max}-1\}\\ 1-F(D_{\max}-1;n^{\tau},\delta^{\tau}), & \text{if } d=D_{\max}\end{cases} \tag{F.19}$$

where $P(x;n^{\tau},\delta^{\tau})$ is the probability mass function of a negative binomial distribution parameterized by a target number of successes n^{τ} and a mean δ^{τ} for weekday τ and $F(x;n^{\tau},\delta^{\tau})$ is the corresponding cumulative distribution function. The probability of success in an individual Bernoulli trial is $p^{\tau}=\frac{n^{\tau}}{n^{\tau}+\delta^{\tau}}$. The parameters for each day of the week are set out in Table F.4.

τ	0	1	2	3	4	5	6
n^{τ}	3.5	11.0	7.2	11.1	5.9	5.5	2.2
δ^{τ}	5.7	6.9	6.5	6.2	5.8	3.3	3.4

Table F.4: Parameters of the demand distribution for each weekday from Monday ($\tau=0$) to Sunday ($\tau=6$) for Scenario C.

The remaining useful life of units on arrival is modelled by a multinomial distribution with a number of trials equal to the order quantity a and a number of

events equal to the maximum useful life m. The probability mass function for the distribution is:

$$P(\underline{\mathrm{Y}} = \underline{\mathrm{y}}|A = a) = P(Y_m = y_m, ..., Y_1 = y_1|A = a) \tag{F.20}$$
$$= \begin{cases} \frac{a!}{y_m! y_{m-1}! ... y_1!} p_m(a)^{y_m} p_{m-1}(a)^{y_{m-1}} ... p_1(a)^{y_1}, & \text{if } a = \sum_{i=1}^{m} y_i \\ 0, & \text{if } a \neq \sum_{i=1}^{m} y_i \end{cases}$$

The parameters of the multinomial distribution are modelled by an affine function of the order quantity a:

$$\log\left(\frac{p_k(a)}{p_1(a)}\right) = c_0^k + c_1^k a, \quad \forall k \in \{2, 3, .., m\} \tag{F.21}$$

If the distribution of remaining useful life on arrival does not depend on order quantity, and therefore the uncertainty is exogenous, $c_1^k = 0 \quad \forall k \in \{2, 3, .., m\}$. The values of c_0^k and c_1^k for our experiments are set out in Table F.5. These represent a subset of the experiments run by Mirjalili [166]. The parameters for the two experiments where $m = 5$ were determined by Mirjalili [166] by fitting multinomial logistic regression models to observed data from a hospital system in Ontario, Canada.

m	Exp	c_0^2	c_0^3	c_0^4	c_0^5	c_0^6	c_0^6	c_0^8	c_1^2	c_1^3	c_1^4	c_1^5	c_1^6	c_1^7	c_1^8
3	1	1.0	0.5												
	2	1.0	0.5						0.40	0.80					
5	1	1.6	2.6	2.8	1.6										
	2	1.9	3.1	3.1	2.5				-0.03	-0.06	-0.03	-0.09			
8	1	0.8	1.4	1.9	2.3	1.7	1.2	0.8							
	2	0.8	1.4	1.9	2.3	1.7	1.2	0.8	-0.03	-0.04	-0.05	-0.06	-0.07	-0.08	-0.09

Table F.5: Parameters for the affine function used to model the parameters of the multinomial distribution of remaining useful life on arrival for each experiment for Scenario C. These are a subset of the experiments described by Mirjalili [166]

The reward function comprises four components: a holding cost per unit in stock at the end of the period (C_h), a shortage cost per unit of unmet demand (C_s), a

wastage cost per unit that perishes at the end of the period (C_w) and a fixed ordering cost which is incurred when $A_t > 0$ (C_f). The single-step reward function after taking action A_t in state $S_t = \left(\tau_t, \underline{\mathrm{X}_t}\right)$, and observing $\Omega_t = (D_t, \underline{\mathrm{Y}_t})$ is

$$\begin{aligned} R_{t+1} = &- C_f \mathbb{1}_{A_t>0} - C_h \left[Y_{m,t} + \sum_{i=1}^{m-1} \min\left(X_{i,t} + Y_{i,t}, A_{\max}\right) - D_t \right]^+ \\ &- C_s \left[D_t - Y_{m,t} - \sum_{i=1}^{m-1} \min\left(X_{i,t} + Y_{i,t}, A_{\max}\right) \right]^+ \\ &- C_w \left[\min\left(X_{1,t} + Y_{1,t}, A_{\max}\right) - D_t \right]^+ \end{aligned} \tag{F.22}$$

Equation F.23 shows how the elements of the state are updated, following a OUFO policy:

$$\begin{aligned} \tau_{t+1} &= (\tau_t + 1) \mod 7 \\ X_{j,t+1} &= \left[\min\left(X_{j+1,t} + Y_{j+1,t}, A_{\max}\right) \right. \\ &\quad \left. - \left[D_t - \sum_{k=1}^{j} \min\left(X_{k,t} + Y_{k,t}, A_{\max}\right) \right]^+ \right]^+ \qquad \forall j \in \{1, ..., m-2\} \\ X_{m-1,t+1} &= \left[Y_{m,t} - \left[D_t - \sum_{k=1}^{m-1} \min\left(X_{k,t} + Y_{k,t}, A_{\max}\right) \right]^+ \right]^+ \end{aligned} \tag{F.23}$$

The scenario is an infinite horizon MDP with a discount factor and periodicity because the demand depends on the day of the week. It is therefore possible to take advantage of the periodicity and use a convergence test based on those described by Su and Deininger [182], as in Section 3.4.7.1. Performing this convergence test requires retaining at least the last seven (as this is the periodicity) estimates of the value function, and it is only possible to test for convergence after performing at least seven iterations. I tested the following inequality at the end of each iteration once $j \geq 7$:

$$\Delta_{\text{max},j} = \max_{s \in \mathbb{S}} \left[\sum_{i=0}^{6} \frac{1}{\gamma^{j-i-1}} \left(V_{j-i}(s) - V_{j-i-1}(s) \right) \right] \tag{F.24}$$

$$\Delta_{\text{min},j} = \min_{s \in \mathbb{S}} \left[\sum_{i=0}^{6} \frac{1}{\gamma^{j-i-1}} \left(V_{j-i}(s) - V_{j-i-1}(s) \right) \right] \tag{F.25}$$

$$\Delta_{\text{max},j} - \Delta_{\text{min},j} \leq 2\epsilon \min \left[|\Delta_{\text{max},j}|, |\Delta_{\text{min},j}| \right] \tag{F.26}$$

When the inequality is met, the additional undiscounted reward being added to each state in one whole cycle (one week) is approximately the same, subject to our confidence level. In turn this means that for each weekday, every state is being increased by the same amount and therefore the best action for each state will not change. I therefore terminated value iteration when the inequality was met. If there were no discounting, and so $\gamma = 1$, the term in the square brackets would be equal to $V_j(s) - V_{j-7}(s)$: the total change in value from one cycle (in this case, one week). This convergence test relies on checkpoints from previous iterations, and I therefore saved a checkpoint every iteration.

Appendix G

Chapter 4 supplement: Additional results

G.1 Scenario A

I present additional results for Scenario A in Table G.1: the order-up-to level parameter S_{best} fit using simulation optimization and the mean and standard deviation of three KPIs calculated over 10,000 evaluation episodes for each policy.

m	Exp	S_{best}	Service level (%)				Wastage (%)				Holding (units)			
			VI		SO		VI		SO		VI		SO	
			Mean	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*
2	1	5	61.0	(1.4)	58.6	(1.3)	2.4	(0.6)	2.2	(0.6)	0.2	(0.0)	0.2	(0.0)
	2	7	72.7	(1.6)	76.6	(1.5)	0.7	(0.4)	1.5	(0.5)	0.5	(0.1)	0.8	(0.1)
	3	5	61.0	(1.4)	58.6	(1.3)	2.4	(0.6)	2.2	(0.6)	0.2	(0.0)	0.2	(0.0)
	4	6	71.7	(1.6)	68.6	(1.5)	0.7	(0.3)	0.7	(0.3)	0.5	(0.1)	0.5	(0.0)
	5	7	61.0	(1.4)	55.4	(1.3)	2.4	(0.6)	2.4	(0.7)	0.2	(0.0)	0.2	(0.0)
	6	9	73.5	(1.7)	69.4	(1.5)	0.9	(0.4)	1.1	(0.4)	0.6	(0.1)	0.6	(0.1)
	7	7	61.0	(1.4)	55.4	(1.3)	2.4	(0.6)	2.4	(0.7)	0.2	(0.0)	0.2	(0.0)
	8	9	72.3	(1.6)	69.4	(1.5)	0.8	(0.4)	1.1	(0.4)	0.6	(0.1)	0.6	(0.1)
3	1	6	69.5	(1.5)	68.3	(1.4)	1.3	(0.4)	1.4	(0.4)	0.5	(0.1)	0.5	(0.0)
	2	8	79.3	(1.5)	83.3	(1.4)	0.1	(0.1)	0.2	(0.2)	0.9	(0.1)	1.3	(0.1)
	3	6	65.2	(1.4)	68.3	(1.4)	0.7	(0.3)	1.4	(0.4)	0.4	(0.0)	0.5	(0.0)
	4	8	79.3	(1.5)	83.3	(1.4)	0.1	(0.1)	0.2	(0.2)	0.9	(0.1)	1.3	(0.1)
	5	8	65.6	(1.6)	62.6	(1.4)	1.7	(0.5)	1.4	(0.5)	0.3	(0.0)	0.4	(0.0)
	6	10	78.1	(1.6)	75.5	(1.5)	0.1	(0.1)	0.1	(0.1)	0.9	(0.1)	0.9	(0.1)
	7	8	65.6	(1.6)	62.6	(1.4)	1.7	(0.5)	1.4	(0.5)	0.3	(0.0)	0.4	(0.0)
	8	10	77.9	(1.6)	75.5	(1.5)	0.1	(0.1)	0.1	(0.1)	0.9	(0.1)	0.9	(0.1)
4	1	7	74.4	(1.4)	76.4	(1.5)	0.7	(0.3)	1.5	(0.4)	0.7	(0.1)	0.8	(0.1)
	2	8	79.3	(1.5)	83.3	(1.4)	0.0	(0.0)	0.0	(0.0)	0.9	(0.1)	1.3	(0.1)
	3	6	73.7	(1.4)	68.5	(1.4)	0.6	(0.3)	0.5	(0.3)	0.7	(0.1)	0.5	(0.1)
	4	8	79.3	(1.5)	83.3	(1.4)	0.0	(0.0)	0.0	(0.0)	0.9	(0.1)	1.3	(0.1)
	5	9	69.5	(1.5)	69.3	(1.5)	1.0	(0.4)	1.0	(0.4)	0.5	(0.1)	0.6	(0.1)
	6	10	78.9	(1.7)	75.5	(1.5)	0.0	(0.0)	0.0	(0.0)	1.0	(0.1)	0.9	(0.1)
	7	9	68.7	(1.5)	69.3	(1.5)	0.9	(0.4)	1.0	(0.4)	0.5	(0.1)	0.6	(0.1)
	8	10	78.9	(1.7)	75.5	(1.5)	0.0	(0.0)	0.0	(0.0)	1.0	(0.1)	0.9	(0.1)
5	1	7	76.3	(1.5)	76.6	(1.5)	0.4	(0.2)	0.7	(0.3)	0.8	(0.1)	0.8	(0.1)
	2	8	79.3	(1.5)	83.3	(1.4)	0.0	(0.0)	0.0	(0.0)	0.9	(0.1)	1.3	(0.1)
	3	7	75.6	(1.4)	76.6	(1.5)	0.3	(0.2)	0.7	(0.3)	0.8	(0.1)	0.8	(0.1)
	4	8	79.3	(1.5)	83.3	(1.4)	0.0	(0.0)	0.0	(0.0)	0.9	(0.1)	1.3	(0.1)
	5	9	71.9	(1.6)	69.5	(1.5)	0.6	(0.3)	0.4	(0.3)	0.6	(0.1)	0.6	(0.1)
	6	10	78.9	(1.7)	75.5	(1.5)	0.0	(0.0)	0.0	(0.0)	1.0	(0.1)	0.9	(0.1)
	7	9	71.5	(1.6)	69.5	(1.5)	0.5	(0.3)	0.4	(0.3)	0.6	(0.1)	0.6	(0.1)
	8	10	78.9	(1.7)	75.5	(1.5)	0.0	(0.0)	0.0	(0.0)	1.0	(0.1)	0.9	(0.1)

The KPIs were calculated for each episode and the mean and standard deviation of each KPI over the 10,000 evaluation episodes using policies fit with value iteration (VI) and simulation optimization (SO) is reported.

Table G.1: The best order-up-to level S_{best}, fit using simulation optimization, and KPIs for policies fit using value iteration and simulation optimization for each of the experimental settings for Scenario A from De Moor *et al.* [40].

G.2 Scenario B

I present additional results for the experimental settings from Hendrix *et al.* [71] in Table G.2 and for the experimental settings from Ortega *et al.* [196] in Table G.3. In each table I present the best combination of order-up-to level parameters $\left(\mathtt{S}^a, \mathtt{S}^b\right)_{\text{best}}$ fit using simulation optimization and the mean and standard deviation of three KPIs calculated over 10,000 evaluation episodes for each policy. Demand for product B was considered to be satisfied for the purposes of calculating the service level if filled by product A when substitution was acceptable.

The only difference between experiment 1 from Hendrix *et al.* [71] and experiment P1 from Ortega *et al.* [196] is that value iteration was run for 100 iterations for experiment P1: more than were required for convergence. The best parameters for the modified base stock policy and the KPIs from evaluating the policies are the same for these experiments, as expected.

				Service level (%)				**Wastage (%)**				**Holding (units)**			
m	Exp	Product	$\mathtt{S}_{\text{best}}$	VI		SO		VI		SO		VI		SO	
				Mean	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*
2	1	A	13	95.5	(0.8)	95.2	(0.8)	6.0	(1.0)	6.3	(1.0)	2.7	(0.1)	2.7	(0.1)
		B	12	94.9	(0.8)	95.5	(0.7)	4.2	(0.8)	5.3	(0.9)	2.1	(0.1)	2.3	(0.1)
	2	A	18	96.9	(0.6)	96.6	(0.6)	4.2	(0.8)	4.4	(0.7)	3.7	(0.2)	3.6	(0.2)
		B	7	91.5	(1.1)	92.5	(1.0)	6.5	(1.2)	8.3	(1.3)	1.2	(0.1)	1.3	(0.1)
3	1	A	15	98.3	(0.5)	98.3	(0.5)	2.2	(0.6)	2.4	(0.6)	4.9	(0.2)	4.8	(0.2)
		B	14	98.4	(0.4)	98.4	(0.4)	1.5	(0.5)	1.7	(0.5)	4.1	(0.2)	4.1	(0.2)
	2	A	21	99.1	(0.3)	99.2	(0.3)	1.2	(0.4)	1.3	(0.4)	6.7	(0.3)	6.7	(0.2)
		B	8	96.6	(0.7)	96.1	(0.7)	3.3	(0.9)	3.2	(0.8)	2.4	(0.1)	2.3	(0.1)
	3	A	15	98.3	(0.5)	98.3	(0.5)	2.2	(0.6)	2.4	(0.6)	4.9	(0.2)	4.8	(0.2)
		B	14	98.4	(0.4)	98.4	(0.4)	1.5	(0.5)	1.7	(0.5)	4.1	(0.2)	4.1	(0.2)
	4	A	21	99.1	(0.3)	99.2	(0.3)	1.2	(0.4)	1.3	(0.4)	6.6	(0.3)	6.7	(0.2)
		B	8	96.6	(0.7)	96.1	(0.7)	3.3	(0.9)	3.2	(0.8)	2.4	(0.1)	2.3	(0.1)

The KPIs were calculated for each episode and the mean and standard deviation of each KPI over the 10,000 evaluation episodes using policies fit with value iteration and simulation optimization is reported.

Table G.2: The best combination of order-up-to levels $\left(\mathtt{S}^a, \mathtt{S}^b\right)_{\text{best}}$, fit using simulation optimization, and KPIs for policies fit using value iteration and simulation optimization for each of the experimental settings for Scenario B from Hendrix *et al.* [71].

m	Exp	Product	S_{best}	Service level (%)				Wastage (%)				Holding (units)			
				VI		SO		VI		SO		VI		SO	
				Mean	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*
2	P1	A	13	95.5	(0.8)	95.2	(0.8)	6.0	(1.0)	6.3	(1.0)	2.7	(0.1)	2.7	(0.1)
		B	12	94.9	(0.8)	95.5	(0.7)	4.2	(0.8)	5.3	(0.9)	2.1	(0.1)	2.3	(0.1)
	P2	A	13	95.5	(0.8)	95.1	(0.8)	6.0	(1.0)	6.2	(0.9)	2.7	(0.1)	2.6	(0.1)
		B	14	95.9	(0.6)	95.5	(0.7)	3.5	(0.7)	3.7	(0.7)	2.5	(0.1)	2.5	(0.1)
	P3	A	16	96.2	(0.7)	96.8	(0.6)	4.9	(0.9)	6.0	(0.9)	3.2	(0.2)	3.4	(0.1)
		B	14	95.9	(0.6)	95.8	(0.6)	3.5	(0.7)	3.7	(0.7)	2.5	(0.1)	2.5	(0.1)
	P4	A	18	96.8	(0.6)	96.7	(0.6)	4.2	(0.8)	4.5	(0.8)	3.7	(0.2)	3.7	(0.2)
		B	17	96.5	(0.6)	97.1	(0.5)	2.9	(0.6)	3.8	(0.7)	2.9	(0.2)	3.2	(0.1)

The KPIs were calculated for each episode and the mean and standard deviation of each KPI over the 10,000 evaluation episodes using policies fit with value iteration (VI) and simulation optimization (SO) is reported.

Table G.3: The best combination of order-up-to levels $\left(S^a, S^b\right)_{\text{best}}$, fit using simulation optimization, and KPIs for policies fit using value iteration and simulation optimization for each of the experimental settings for Scenario B from Ortega *et al.* [196].

G.3 Scenario C

In Table G.4 I present the best combination of parameters for the heuristic policy fit using simulation optimization. In Table G.5 I present the mean and standard deviation of three KPIs calculated over 10,000 evaluation episodes for each policy. For consistency with the calculation of the reward function in Mirjalili [166] the holding KPI includes the units that will expire at the end of the current day. These units are excluded from the calculation of the stock holding at the end of the day in the other scenarios.

			Weekday τ						
m	Exp	Parameter	0	1	2	3	4	5	6
3	1	$\mathtt{S}_{\text{best}}$	13	12	14	11	11	8	7
		$\mathtt{s}_{\text{best}}$	6	7	7	6	6	3	3
	2	$\mathtt{S}_{\text{best}}$	14	14	15	13	12	9	9
		$\mathtt{s}_{\text{best}}$	7	7	7	7	6	3	4
5	1	$\mathtt{S}_{\text{best}}$	16	17	16	13	13	10	14
		$\mathtt{s}_{\text{best}}$	7	8	8	7	7	3	3
	2	$\mathtt{S}_{\text{best}}$	17	16	16	13	13	11	14
		$\mathtt{s}_{\text{best}}$	7	7	9	8	8	3	4
8	1	$\mathtt{S}_{\text{best}}$	19	15	18	18	14	13	16
		$\mathtt{s}_{\text{best}}$	8	8	8	7	8	3	4
	2	$\mathtt{S}_{\text{best}}$	18	18	16	15	14	11	15
		$\mathtt{s}_{\text{best}}$	9	8	9	7	7	3	4

Table G.4: The best combination of parameters for the heuristic policy $\left(\left(\mathtt{s}^0, \mathtt{S}^0\right), ..., \left(\mathtt{s}^6, \mathtt{S}^6\right)\right)_{\text{best}}$ fit using simulation optimization for each of our experiments for Scenario C, a subset of the experiments run by Mirjalili [166].

m	Exp	Service level (%)				Wastage (%)				Holding (units)			
		VI		SO		VI		SO		VI		SO	
		Mean	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*
3	1	95.3	(0.9)	95.3	(0.9)	12.6	(1.3)	12.6	(1.4)	4.9	(0.1)	4.9	(0.1)
	2	96.6	(0.8)	96.2	(0.8)	7.0	(1.1)	7.2	(1.1)	5.8	(0.1)	5.7	(0.1)
5	1	97.4	(0.7)	97.0	(0.7)	3.2	(0.8)	3.0	(0.7)	6.8	(0.1)	6.7	(0.1)
	2	97.4	(0.7)	97.5	(0.7)	3.1	(0.7)	3.4	(0.8)	6.8	(0.1)	7.0	(0.2)
8	1	—	—	97.9	(0.6)	—	—	0.7	(0.3)	—	—	8.0	(0.2)
	2	—	—	97.7	(0.6)	—	—	1.0	(0.4)	—	—	7.5	(0.2)

The KPIs were calculated for each episode and the mean and standard deviation of each KPI over the 10,000 evaluation episodes using policies fit with value iteration (VI) and simulation optimization (SO) is reported. Value iteration was not feasible when $m = 8$.

Table G.5: KPIs for policies fit using value iteration and simulation optimization for each of our experiments for Scenario C, a subset of the experiments run by Mirjalili [166].

Appendix H

Chapter 5 supplement: Additional information on the simulated workflow and simulation experiments

H.1 Simulated workflow

H.1.1 Workflow as a Markov decision process

The state of the environment S_t comprises three components: the day of the week $\tau_t \in \{0, 1, ...6\}$, the stock on hand $\underline{\mathrm{X}}_t$ ordered by ascending age and the units that were issued on day $t-1$ but not transfused, $\underline{\mathrm{Z}}_t$ again ordered by ascending age.

$$\underline{\mathrm{X}}_t = [X_{m,t}, X_{m-1,t}, ..., X_{1,t}] \tag{H.1}$$

$$\underline{\mathrm{Z}}_t = [Z_{m-1,t}, Z_{m-2,t}, ..., Z_{0,t}] \tag{H.2}$$

I assume that $\underline{\mathrm{Z}}_t$ is not observable at the point the agent takes its decision, and therefore the agent observes $O_t = [\tau_t, \underline{\mathrm{X}}_t]$. This is therefore a POMDP [97, Chapter 17], in which the observation accurately captures part of the underlying state, and part of the underlying state is always unobservable.

The action in the MDP, A_t is the number of platelet units ordered at the start of day t. While the focus of this work is on issuing policies, this required

finding plausible replenishment policies, and I used the MDP to fit the parameters of heuristic replenishment policies as described in Sections 5.4.1.3 and H.2. The heuristic replenishment policies use, at most, the day of the week and the total stock on hand at the start of the day as input.

When order A_t is placed, the units are assumed to be delivered immediately, a lead time L of zero periods. The age profile of the units received is $\underline{Y}_t = [Y_{m,t}, Y_{m-1,t}, ..., Y_{1,t}]$. The remaining useful life of the units on arrival is modelled by a multinomial distribution with a number of trials equal to the order quantity A_t and a number of events equal to the maximum useful life m. The parameters of the multinomial distribution may depend on the day of the week, τ_t, and the parameters corresponding to weekday τ are represented as $\underline{\Delta}_\tau$. The transition dynamics of the system are defined implicitly by the simulation.

H.1.2 Order of events

There are six stages in the simulated workflow, as illustrated in Figure 5.1. Some variables are updated at multiple stages, and I indicate the value of a variable at the end of stage k using a superscript on the left hand side of the variable, e.g. ${}^k\underline{\mathrm{X}}_t$ is the vector of units in stock at the end of stage k on day t. Stage 0 is start of the day, before an order is placed and ${}^0\underline{\mathrm{X}}_t \equiv \underline{\mathrm{X}}_t$. The number of units for which an emergency order was placed due to a shortage is E_t (${}^0E_t = 0$) and the vector of units issued on day t but not transfused is $\underline{\mathrm{U}}_t$ (${}^0\underline{\mathrm{U}}_t = \underline{0}$).

Demand is sampled independently for the morning and afternoon using parameters $\mu_\tau^{\text{am}} = \mu_\tau^{\text{pm}} = \frac{\mu_\tau}{2}$ for day of the week τ. I use the function MEET_DEMAND in the equations below to represent the process of issuing units and simulating the true and predicted labels required for the YUPR issuing policy. To simulate predicted labels, I specified the sensitivity α and specificity β of the predictive model assumed to be available. I describe this approach in Section 5.4.1.4 and provide pseudocode for MEET_DEMAND in Algorithm 5.

Stage 1: At the start of each day t, the agent makes an observation O_t comprising the current inventory in stock (split by remaining useful life) and the day of the week. The agent places a replenishment order, A_t, following a replenishment

policy and the order, with age profile $\underline{\mathrm{Y}}_t$, is assumed to arrive immediately.

$$\underline{\mathrm{Y}}_t \sim \text{Multinomial}(A_t, \underline{\Delta}_\tau) \tag{H.3}$$

$${}^1X_{m,t} = Y_{m,t} \tag{H.4}$$

$${}^1X_{j,t} = {}^0X_{j,t} + Y_{j,t} \quad \forall j, 1 \leq j \leq m-1 \tag{H.5}$$

Stage 2: Total demand for the morning, D_t^{am}, is sampled from a Poisson distribution, and filled following the issuing policy. If there is a shortage, an emergency order is placed to fill the demand. For each request, a sample from a Bernoulli distribution with probability of success termed the return rate, ρ, is taken to determine whether the unit will be returned or transfused to a patient.

$$D_t^{\text{am}} \sim \text{Poisson}(\mu_\tau^{\text{am}}) \tag{H.6}$$

$${}^2\underline{\mathrm{X}}_t, {}^2\underline{\mathrm{U}}_t, {}^2E_t = \textsc{meet\_demand}({}^1\underline{\mathrm{X}}_t, {}^0\underline{\mathrm{U}}_t, {}^0E_t, D_t^{\text{am}}, \alpha, \beta, \rho, \underline{\Delta}_\tau) \tag{H.7}$$

Stage 3: At midday, the units that were issued on day $t-1$ but not transfused are returned to the blood bank. Returned units that were issued on their last day of remaining useful life will have expired. The number of returned units with remaining useful life i that are not in a fit state to be reissued, $N_{i,t}$, is sampled from a Binomial distribution with a number of trials equal to the number of returned units with remaining useful life i and a probability of success termed the slippage rate, ϕ. Units that have not expired or been wasted due to slippage are now available to fill demand that arises during the rest of the day.

$${}^3X_{m,t} = {}^2X_{m,t} \tag{H.8}$$

$$N_{i,t} \sim \text{Binomial}(Z_{i,t}, \phi) \quad \forall i, 1 \leq i \leq m-1 \tag{H.9}$$

$${}^3X_{i,t} = {}^2X_{i,t} + Z_{i,t} - N_{i,t} \quad \forall i, 1 \leq i \leq m-1 \tag{H.10}$$

Stage 4: As at stage 2, total demand for the afternoon, D_t^{pm} is sampled from a Poisson distribution, and filled following the issuing policy. If there is a shortage, an emergency order is placed to fill the demand. For each request, a sample from a Bernoulli distribution with probability of success termed the return rate, ρ, is taken to determine whether the unit will be returned or transfused to a patient.

$$D_t^{\text{pm}} \sim \text{Poisson}(\mu_\tau^{\text{pm}}) \tag{H.11}$$

$${}^4\underline{\mathbf{X}}_t, {}^4\underline{\mathbf{U}}_t, {}^4E_t = \text{MEET\_DEMAND}({}^3\underline{\mathbf{X}}_t, {}^2\underline{\mathbf{U}}_t, {}^2E_t, D_t^{\text{pm}}, \alpha, \beta, \rho, \underline{\Delta}_\tau) \tag{H.12}$$

Stage 5: The stock is aged one day, and any units that had a remaining useful life of one day at the start of day t are assumed to expire.

$${}^5X_{i,t} = {}^4X_{i+1,t} \quad \forall i, 1 \leq i < m \tag{H.13}$$

$${}^5X_{m,t} = 0 \tag{H.14}$$

$$W_t = {}^4X_{1,t} + \sum_{i=1}^{m-1} N_{i,t} \tag{H.15}$$

Stage 6: The state is updated for the start of the next day and the reward is calculated.

$$\tau_{t+1} = \tau_t + 1 \mod 7 \tag{H.16}$$

$$X_{i,t+1} = {}^5X_{i,t} \quad \forall i, 1 \leq i \leq m \tag{H.17}$$

$$Z_{i-1,t+1} = {}^4U_{i,t} \quad \forall i, 1 \leq i \leq m \tag{H.18}$$

$$R_{t+1} = -C_f \mathbb{1}_{A_t>0} - C_v A_t - C_h \sum_{i=1}^{m-1} {}^5X_{i,t} - C_s {}^4E_t - C_w \left[W_t + \frac{Z_{0,t}}{\gamma} \right] \tag{H.19}$$

The reward function component related to units that were issued on their last

day of life but were not transfused and therefore expired, $Z_{0,t}$ is divided by the discount factor γ because it is registered one timestep later than it would have been if it had expired in stock. This ensures that, if considering the discounted return G, there is no difference between a unit expiring in stock or in a remote agitator waiting to be returned.

In Algorithm 5 I set out how the YUPR issuing policy operates at stages 2 and 4 of the simulated workflow.

Algorithm 5 Meet demand for simulation stages 2 and 4

function ISSUE_YUFO($\underline{\mathrm{X}}$)
  for $i \leftarrow 0$ to length($\underline{\mathrm{X}}$) $- 1$ **do**
    if $\underline{\mathrm{X}}[i] > 0$ **then**
      return i
    end if
  end for
end function
function ISSUE_OUFO($\underline{\mathrm{X}}$)
  for $i \leftarrow$ length($\underline{\mathrm{X}}$) $- 1$ downto 0 **do**
    if $\underline{\mathrm{X}}[i] > 0$ **then**
      return i
    end if
  end for
end function
function MEET_DEMAND($\underline{\mathrm{X}}, \underline{\mathrm{U}}, E, D, \alpha, \beta, \rho, \underline{\Delta}$)
  for $d \leftarrow 1$ to D **do**
    $label \leftarrow$ Sample from Bernoulli(ρ)
    if sum($\underline{\mathrm{X}}$) > 0 **then**
      if $label = 1$ **then**
        $sample \leftarrow$ Sample from Uniform$(0, 1)$
        if $sample < \alpha$ **then**
          $prediction \leftarrow 1$
          $index \leftarrow$ Call ISSUE_YUFO($\underline{\mathrm{X}}$)
        else
          $prediction \leftarrow 0$
          $index \leftarrow$ Call ISSUE_OUFO($\underline{\mathrm{X}}$)
        end if
        $\underline{\mathrm{U}}[index] \leftarrow \underline{\mathrm{U}}[index] + 1$
      else
        $sample \leftarrow$ Sample from Uniform$(0, 1)$
        if $sample > \beta$ **then**
          $prediction \leftarrow 1$
          $index \leftarrow$ Call ISSUE_YUFO($\underline{\mathrm{X}}$)
        else
          $prediction \leftarrow 0$
          $index \leftarrow$ Call ISSUE_OUFO($\underline{\mathrm{X}}$)
        end if
      end if
      $\underline{\mathrm{X}}[index] \leftarrow \underline{\mathrm{X}}[index] - 1$
    else
      $E \leftarrow E + 1$
      $index \leftarrow$ Sample from Multinomial$(1, \underline{\Delta})$
      if $label = 1$ **then**
        $\underline{\mathrm{U}}[index] \leftarrow \underline{\mathrm{U}}[index] + 1$
      end if
    end if
  end for
  return $\underline{\mathrm{X}}, U, E$
end function

H.2 Replenishment policies

I considered two replenishment policies: a standing order policy and an $(\mathtt{s}, \mathtt{S})$ policy. The parameters for these heuristic policies were fit using simulation optimization. I evaluated each proposed set of parameters on 1,000 rollouts (the validation rollouts), each 365 days long following a warm-up period of 100 days. For each replenishment policy, the parameters(s) that achieved the highest mean return G over the 1,000 rollouts for a given scenario were used for subsequent evaluation.

The standing order policy (Equation H.20) has a single parameter, $\mathtt{Q}$, the number of units ordered each day. Under this policy, the same number of units are ordered on every day of the week, irrespective of the current stock. I used Optuna's grid sampler to evaluate all values of $\mathtt{Q}$ between 0 and the maximum order quantity, $A_{\max} = 100$. The standing order policy that achieved the highest mean return G over the 1,000 rollouts for a given scenario is characterized by the parameter $\mathtt{Q}_{\text{best}}$, and was used for subsequent evaluation.

$$A_t = \mathtt{Q} \tag{H.20}$$

The $(\mathtt{s}, \mathtt{S})$ is formally an $(\mathtt{R}, \mathtt{s}, \mathtt{S})$ policy with a fixed review period $\mathtt{R}$ equal to one day. The $(\mathtt{s}, \mathtt{S})$ policy has 14 parameters: an order-up-to level parameter $\mathtt{S}$ and a reorder point parameter $\mathtt{s}$ for each day of the week [323]. The order quantity on day t, given that the day of the week is τ and the total current stock on hand is X_t is:

$$A_t = \begin{cases} [\mathtt{S}^\tau - X_t]^+ & \text{if } X_t \leq \mathtt{s}^\tau \\ 0 & \text{if } X_t > \mathtt{s}^\tau \end{cases} \tag{H.21}$$

where $(\mathtt{s}^\tau, \mathtt{S}^\tau)$ is the pair of parameters for day of the week τ.

I used Optuna's NSGA-II sampler, a genetic algorithm, to suggest parameter combinations of $\mathtt{s}^\tau \in \{0, 1, ..., \mathtt{s}_{\max} = A_{\max}\} = 100$ and $\mathtt{S}^\tau \in \{0, 1, ..., \mathtt{S}_{\max} = A_{\max}\} = 100 \; \forall \tau \in \{0, 1, .., 6\}$. There is a hard constraint on the relative values of

$\mathtt{s}$ and $\mathtt{S}$ for each weekday: $\mathtt{s}^\tau < \mathtt{S}^\tau \ \forall \tau \in \{0, 1, .., 6\}$. It is not possible to restrict the search space in Optuna based on relative values of parameters, and therefore I enforced this constraint by forcing the policy to order zero units on each day of the simulation if it was violated. For each generation of the genetic algorithm, I ran 50 proposed combinations of parameters in parallel and ranked them based on the mean return G. The search procedure terminated when the best combination of parameters had not changed for a specified number generations, or after 200 generations had been completed. The $(\mathtt{s}, \mathtt{S})$ policy that achieved the highest mean return G over the 1,000 rollouts for a given scenario when the search procedure was terminated is characterized by parameters $((\mathtt{s}^0, \mathtt{S}^0), ..., (\mathtt{s}^6, \mathtt{S}^6))_{\text{best}}$ and was used for subsequent evaluation.

For experiments 1–4, I refit the parameters of the replenishment policy for each pair of values of sensitivity and specificity, using the best parameters identified for a policy that is equivalent to an OUFO policy as the first proposed combination for each subsequent simulation optimization run. I terminated the parameter search when the best policy parameters had not changed for 10 generations, or after a maximum of 200 generations.

For experiments 5–7, for each different level of an input setting in each experiment I fit the parameters for the replenishment policy under an OUFO issuing policy first and used these as the starting point when fitting the parameters for both the next input setting under an OUFO issuing policy, and for the corresponding input setting under a YUPR-PPM issuing policy. I terminated the parameter search when the best policy parameters had not changed for 50 generations, or after a maximum of 200 generations. Note that this early-stopping limit is more relaxed than in experiments 1–4 for computational reasons.

H.3 Simulation inputs

Four key simulation inputs were calculated based on information from Bank Manager, the laboratory information management system used by the UCLH transfusion laboratory. The mean daily demand, return rate, and slippage rate were calculated based on a common set of platelet requests: I queried all the requests for platelets where the units were required between 1 January 2015 and 31 December 2016 inclusive and to which a unit was allocated, and excluded any requests where the allocated unit was a neonatal unit or where the patient was a test patient used for system diagnostics.

H.3.1 Mean daily demand

H.3.1.1 Total demand for units (UCLH Tx+r)

The mean daily demand for each day of the week was calculated as the sum of units in the common set of platelet requests that were required on that day of the week divided by the total number of occurrences of that day of the week in the period. These values are set out in Table H.1.

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
μ_τ	28.8	33.4	26.2	28.4	30.8	18.6	19.6
μ_τ^{am}	14.4	16.7	13.1	14.2	15.4	9.3	9.8
μ_τ^{pm}	14.4	16.7	13.1	14.2	15.4	9.3	9.8

Table H.1: Mean daily demand per weekday including returns (UCLH Tx+r)

H.3.1.2 Demand for transfused units (UCLH Tx)

To estimate the demand for units that were transfused, I multiplied the values obtained for the total demand (UCLH Tx+r) by $(1 - \rho)$. These values are set out in Table H.2.

H.3.2 Return rate (ρ)

The return rate, ρ, was calculated as the proportion of requested units in the common set of requests where the final allocation status was not either "Transfusion" or "Transfusion Assumed".

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
μ_τ	26.4	30.6	24.2	26.0	28.4	17.0	18.0
μ_τ^{am}	13.2	15.3	12.1	13.0	14.2	8.5	9.0
μ_τ^{pm}	13.2	15.3	12.1	13.0	14.2	8.5	9.0

Table H.2: Mean daily demand per weekday adjusted by the return rate to include only units that will be transfused (UCLH Tx)

H.3.3 Slippage rate (ϕ)

I took the total number of units where reissue was possible as requested units in the common set of requests where the final allocation status was not "Transfusion" or "Transfusion Assumed" and where the allocated unit was not due to expire on the day the units were required. I estimated the slippage rate as the proportion of these units that were not subsequently reissued to another request. The estimate is an upper limit because there may be some units included in the numerator that were in a sufficiently good state to be reissued but were not due to a lack of demand. This is a conservative approach to estimating the benefits of the YUPR issuing policy because I expect it will have a greater advantage over an OUFO policy when the slippage rate is low.

H.3.4 Distribution of remaining useful life on arrival (UCLH)

I queried all platelet units received between 1 January 2015 and 31 December 2016 inclusive, and excluded units that were either neonatal or not extended-life (extended life units, with a maximum shelf life of seven days from donation comprised approximately 99% of the units received). The maximum useful life of extended-life units on arrival at the hospital blood bank is five days due to processing and testing required before distribution. To ensure the distributions represented the age of units available for routine morning deliveries I also excluded units not received within 90 minutes after the first routine delivery slot of the day. There were no routine deliveries on Sundays during the period of study, so I used the routine delivery time of Saturday as a proxy. The remaining useful life of a unit was calculated as the number of days between receipt of the unit and its expiry date (with 1 day corresponding to receipt on the day of expiry). For each day of

the week, I calculated the units received on that day of the week with a remaining useful life of one to five days as a percentage of the total number of units received on that day of the week meeting the criteria. The same process was repeated for units received between 1 January 2017 and 31 December 2017 inclusive to calculate the distribution for 2017 used as part of the evaluation of the trained predictive model within the simulated workflow. The parameters for the multinomial distribution of remaining useful life on arrival for 2015–2016 are set out in Table H.3 and the corresponding figures for 2017 in Table H.4. For simplicity, the same distributions were used for both the routine morning deliveries and any emergency orders placed in the event of a shortage. The distribution of remaining useful life on arrival may differ later in the day, but very few emergency orders were placed due to the high service levels achieved by the policies.

In Table H.5 I present the parameters for the multinomial distribution of remaining useful life on arrival used for experiment 9, which were generated from a binomial distribution with a changing probability of success as described in Section 5.4.1.5.

Weekday	**Remaining useful life on arrival (days)**				
	5	4	3	2	1
Mon	0.25	0.33	0.28	0.11	0.03
Tue	0.20	0.35	0.27	0.13	0.05
Wed	0.26	0.18	0.38	0.14	0.04
Thu	0.76	0.07	0.05	0.09	0.03
Fri	0.62	0.29	0.02	0.03	0.04
Sat	0.61	0.28	0.11	0.00	0.00
Sun	0.48	0.27	0.19	0.05	0.01

Table H.3: Parameters for the multinomial distribution of remaining useful life on arrival from UCLH in 2015–2016

H.4 Simulation experiments with no returns

Previous work has made the assumption that all requested platelet units are transfused, and focused on optimizing replenishment policies. I conducted experiments I–IV to determine how my simulated workflow performed under that

Weekday	**Remaining useful life on arrival (days)**				
	5	4	3	2	1
Mon	0.31	0.32	0.23	0.12	0.02
Tue	0.18	0.48	0.21	0.10	0.03
Wed	0.25	0.19	0.38	0.15	0.03
Thu	0.87	0.02	0.03	0.07	0.01
Fri	0.71	0.24	0.02	0.01	0.01
Sat	0.62	0.28	0.10	0.00	0.00
Sun	0.48	0.28	0.18	0.06	0.00

Table H.4: Parameters for the multinomial distribution of remaining useful life on arrival from UCLH in 2017

Probability of success	**Remaining useful life on arrival (days)**				
	5	4	3	2	1
0.0	0.00	0.00	0.00	0.00	1.00
0.1	0.00	0.00	0.05	0.29	0.66
0.2	0.00	0.03	0.15	0.41	0.41
0.3	0.01	0.08	0.26	0.41	0.24
0.4	0.02	0.15	0.35	0.35	0.13
0.5	0.06	0.25	0.38	0.25	0.06
0.6	0.13	0.35	0.35	0.15	0.02
0.7	0.24	0.41	0.26	0.08	0.01
0.8	0.41	0.41	0.15	0.03	0.00
0.9	0.66	0.29	0.05	0.00	0.00
1.0	1.00	0.00	0.00	0.00	0.00

Table H.5: Parameters for the multinomial distribution of remaining useful life on arrival for sensitivity analysis in experiment 9

assumption. Experiments I–IV are similar to experiments 1–4 respectively, but with no returns. I therefore set $\rho = 0$ and $\phi = 0$ and only used an OUFO issuing policy because I anticipated no benefit to using a predictive model or the YUPR policy when there are no returns. I estimated the demand for units that would be transfused (UCLH Tx) by multiplying the total demand (UCLH Tx+r) by a factor of $(1 - \rho)$ = 0.92, using the estimated return rate from UCLH during 2015-2016. The expected number of transfused units is therefore the same in experiments 1–4 and I–IV, and is set out in Table H.2.

In Table H.6 I summarize the key settings of experiments I–IV and in Table H.7 I report the results of experiments I–IV in terms of the mean daily cost, service level and wastage over 10,000 simulated years.

Exp	Distribution of remaining useful life on arrival	ρ	ϕ	Demand distribution	Replenishment policy	Issuing policy
I	UCLH	0%	0%	UCLH Tx	Standing order	OUFO
II	UCLH	0%	0%	UCLH Tx	`(s,S)`	OUFO
III	R&R [54]	0%	0%	UCLH Tx	Standing order	OUFO
IV	R&R [54]	0%	0%	UCLH Tx	`(s,S)`	OUFO

Table H.6: Summary of input settings for experiments I–IV using the simulated platelet inventory management workflow with returns.

Distribution of remaining useful life on arrival	Exp	Replenishment policy	Issuing policy	Daily cost		Service level (%)		Wastage (%)	
				Mean	*(s.d.)*	*Mean*	*(s.d.)*	*Mean*	*(s.d.)*
UCLH	I	Standing order	OUFO	20,202	(601)	98.0	(0.9)	0.1	(0.2)
	II	(s, S)	OUFO	17,541	(194)	99.6	(0.1)	0.0	(0.0)
R&R	III	Standing order	OUFO	19,831	(466)	97.1	(0.7)	1.0	(0.4)
	IV	(s, S)	OUFO	17,556	(200)	99.5	(0.2)	0.0	(0.0)

The daily cost and KPIs were calculated for each episode and the mean and standard deviation over the 10,000 evaluation episodes is reported.

Table H.7: Simulation results for experiments I–IV using the simulated platelet inventory management workflow with returns.

The results of experiments I–IV show that, if all requested units are transfused, then a simple `(s,S)` replenishment policy is sufficient to achieve no wastage and a very high (>99.5%) service level.

Appendix I

Chapter 5 supplement: Additional information on training the machine learning model

I.1 Training and test set request exclusion

In Table I.1 I set out the how the training and test sets were extracted from the requests recorded in Bank Manager, the laboratory information management system used in the UCLH transfusion laboratory.

	2015 - 2016	2017
Total requests	18,917	9,793
Less: no units assigned	(656)	(282)
Less: neonatal unit assigned	(236)	(152)
Less: request for test patient	–	–
Less: request required more than 30 minutes before registered	(5)	(6)
Less: requests removed so that all requests have 30 day look-back window	(723)	–
	17,297	9,353

Table I.1: Requests for platelet units used to train and evaluate the predictive model for platelet returns.

I.2 Input features for the machine learning model

In Table I.2 I summarize the input features for the predictive model. In Table I.3 I provide the proportion of missing values and the mean and standard deviation for the numeric and binary input features. In Table I.4 I provide the proportion of missing values and the three most frequent categories for the categorical features.

Feature name	Type (units)	Description
age	Integer (years)	Difference between year of request and year of birth
male	Binary	Patient sex is male
request_registered_hour	Integer (0-23)	Hour of day request was registered
request_registered_weekday	Integer (0-6)	Day of the week request was registered
request_required_hour	Integer (0-23)	Hour of day requested units are required
request_required_weekday	Integer (0-6)	Day of week requested units are required
registered_required_diff_hours	Float (hours)	Time between request being registered and the units being required
request_priority	Binary	Request has priority flag
requested_size	Integer (count)	Number of units requested
plt_count_value	Float (10^9L^{-1})	Value of the most recent platelet count available at the prediction point (up to 7 days ago)
plt_result_time_diff_hours	Float (hours)	Time between the result of the most recent platelet count and the prediction point
plt_specimen_time_diff_hours	Float (hours)	Time between the recorded collection time of the specimen used for the most recent platelet count and the prediction point
num_units_allocated_last_30_days	Integer (count)	Number of units allocated to previous requests for this patient in the last 30 days
last_transfused_unit_required_time_diff_hours	Float (hours)	Time between when the most recent request for which a unit was transfused was required and the prediction point
proportion_of_requests_last_30_days_not_all_transfused	Float (proportion)	Proportion of requests required in the last 30 days where not all units were transfused
num_requested_units_required_after_last_plt_sample	Integer (count)	Number of units allocated to requests required after the specimen time for the most recent platelet count
num_units_transfused_since_last_plt_sample	Integer (count)	Number of units recorded as transfused since the specimen time for the most recent platelet count
hospital	String	Hospital site the patient at which there patient is located
discipline	String	Discipline of the consultant responsible for the patient
ward_name	String	Ward on which the patient is located
ward_type	String	Type of ward on which the patient is located
plt_count_request_location	String	Location from which the request for the most recent platelet count (within 7 days) was made
required_location	String	Location to which the platelets are to be delivered

Table I.2: Summary of input features for the predictive model for platelet returns.

	2015 - 2016			2017		
Feature name	Missing (%)	*Mean*	*(s.d.)*	Missing (%)	*Mean*	*(s.d.)*
age	0.43	52.32	(18.79)	0.01	50.12	(19.06)
male	0.41	0.56		0.04	0.61	
request_registered_hour	–	12.21	(4.31)	–	12.18	(4.53)
request_registered_weekday	–	2.72	(1.93)	–	2.77	(1.93)
request_required_hour	–	12.88	(4.26)	–	12.82	(4.37)
request_required_weekday	–	2.75	(1.93)	–	2.79	(1.94)
registered_required_diff_hours	–	4.25	(17.09)	–	5.00	(21.09)
requested_priority	–	0.05		–	0.03	
requested_size	–	1.08	(0.28)	–	1.05	(0.22)
plt_count_value	2.43	27.20	(45.34)	3.27	24.74	(35.36)
plt_result_time_diff_hours	2.43	17.78	(29.70)	3.27	18.06	(28.14)
plt_specimen_time_diff_hours	2.43	21.50	(29.89)	3.27	22.64	(29.13)
num_units_allocated_last_30_days	–	8.02	(8.76)	–	7.20	(7.90)
last_transfused_unit_required_time_diff_hours	15.54	72.58	(99.07)	16.14	77.66	(106.42)
proportion_of_requests_last_30_days_not_all_transfused	15.10	0.07	(0.14)	15.72	0.09	(0.16)
num_requested_units_required_after_last_plt_sample	–	0.19	(0.49)	–	0.22	(0.53)
num_units_transfused_since_last_plt_sample	–	0.22	(0.27)	–	0.24	(0.55)

Table I.3: Additional information on numeric and binary input features for the predictive model for platelet returns.

	2015 - 2016			2017		
Feature name	Missing (%)	Top 3 categories		Missing (%)	Top 3 categories	
hospital	–	New UCLH	68.6%	–	New UCLH	74.0%
		UCH Macmillan Cancer Centre	14.5%		UCH Macmillan Cancer Centre	13.2%
		Harley Street on 15	12.2%		Harley Street on 15	9.2%
discipline	0.67	Haematology (Clinical)	73.0%	0.84	Haematology (Clinical)	74.4%
		Paed Clinical Haematology	5.3%		Paed Clinical Haematology	5.8%
		Medical Oncology	3.2%		Bone Marrow Transplant	2.9%
ward_name	–	UCH Tower 16th Fl. S	14.3%	–	Oncology Tower 16th Fl. N	11.8%
		UCH Tower 13th Fl. N	11.0%		UCH Tower 16th Fl. S	11.4%
		Harley Street on 15	10.6%		UCH Tower 13th Fl. N	10.0%
ward_type	0.33	Inpatient	58.9%	0.62	Inpatient	63.6%
		Day Case	14.6%		Day Case	13.2%
		Outpatient	12.7%		Outpatient	11.9%
plt_count_request_location	2.43	T16S	14.8%	3.27	T16N	12.0%
		T13N	12.0%		T16S	12.0%
		HS15	11.1%		T13N	10.0%
required_location	–	T16 Platelet Agitator	63.9%	–	T16 Platelet Agitator	68.4%
		Non-Fridge Location	26.0%		MCC Platelet Agitator	18.8%
		MCC Platelet Agitator	7.8%		Non-Fridge Location	10.6%

Table I.4: Additional information on categorical input features for the predictive model for platelet returns.

I.3 Hyperparameter tuning and threshold selection for the machine learning model

In Table I.5 I set out the search ranges for the hyperparameters of the ML pipeline I constructed using the Python library scikit-learn [324]. The search ranges are expressed using the syntax of the Python library hydra [325], which was used for configuration . I used the default parameter values for the `XGBClassifier` class from the Python library xgboost v1.7.5 for the hyperparameters not specified in the table

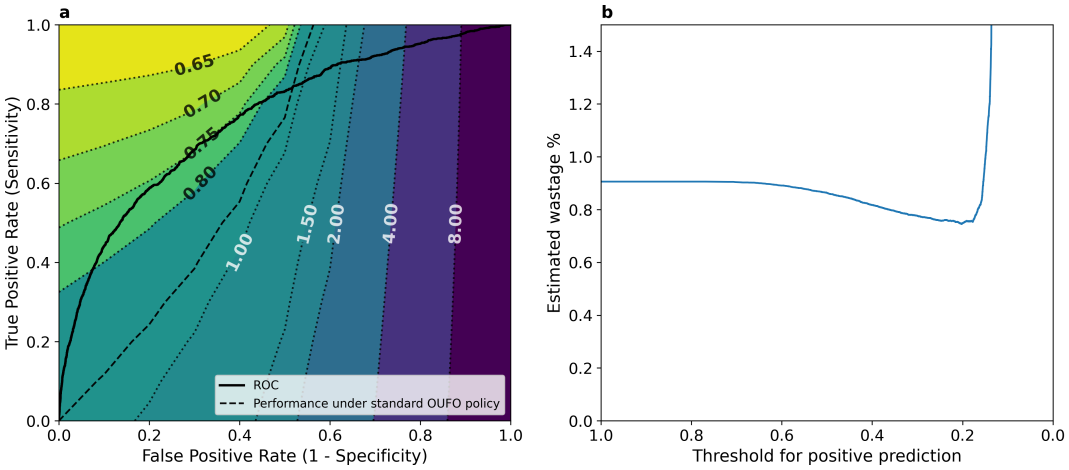
The categorical features discipline, ward_name and plt_count_request_location were all encoded using the `OneHotEncoder` class from the Python library scikit-learn. The parameter min_frequency is the minimum frequency required for a category to be encoded separately (categories below this threshold are collectively assigned to a separate 'infrequent' class).

	Parameter	Search range	Final value
Preprocessing	discipline - min_frequency	`interval(0.01, 0.5)`	0.15
	ward_name - min_frequency	`interval(0.01, 0.5)`	0.22
	plt_count_request_location - min_frequency	`interval(0.01, 0.5)`	0.02
XGBoost	gamma	`tag(log, interval(0.1, 100))`	38.11
	learning_rate	`tag(log, interval(0.01, 0.3))`	0.11
	max_depth	`int(interval(2,20))`	6
	min_child_weight	`tag(log, interval(0.1, 100))`	0.16
	n_estimators	`choice(50, 100, 200, 400, 800)`	100
	reg_alpha	`tag(log, interval(0.01, 100))`	0.01
	reg_lambda	`tag(log, interval(0.01, 100))`	0.80
	scale_pos_weight	`tag(log, interval(0.01, 100))`	3.15
	subsample	`interval(0.1, 1.0)`	0.83

Table I.5: Hyperparameter search ranges and final values for training the predictive model for platelet returns.

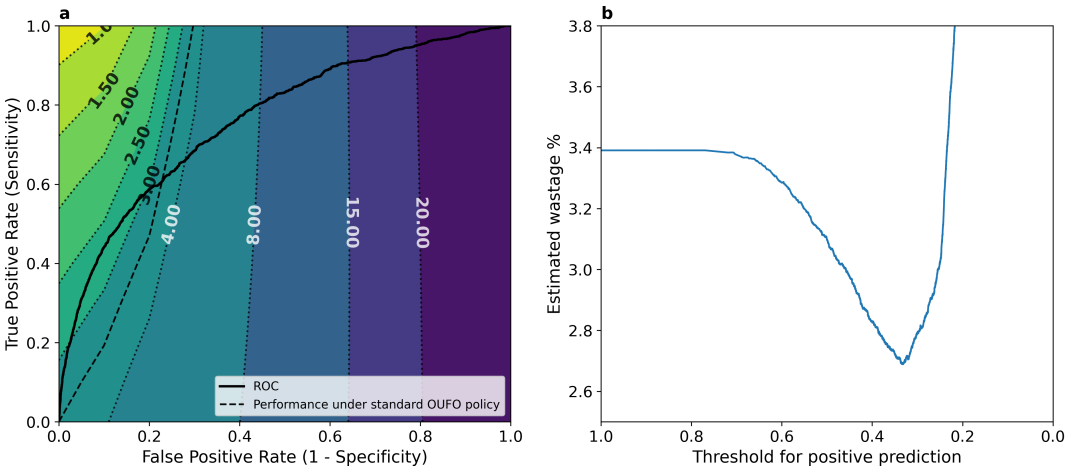
In order to apply the trained model within the simulated workflow, I needed to determine a threshold for distinguishing between positive and negative predictions. In Figure I.1a I plot the ROC curve for the training set over a contour plot of the wastage from experiment 4 and in Figure I.1b I plot the estimated wastage against the threshold used to determine a positive prediction. Figure I.2 presents the same information using the wastage results from experiment 8, in which the distribution of remaining useful life on arrival was taken from previous work by Rajendran and

Ravindran [54].



(a) The ROC on the training set overlaid on a contour plot of the wastage results from experiment 4 and **(b)** the estimated wastage based on the results from experiment 4 for different positive prediction thresholds on the training set (points along the ROC).

Figure I.1: Determining the threshold for the predictive model using the simulation results from experiment 4.



(a) The ROC curve on the training set overlaid on a contour plot of the wastage results from experiment 8 and **(b)** the estimated wastage based on the results from experiment 8 for different positive prediction thresholds on the training set (points along the ROC).

Figure I.2: Determining the threshold for the predictive model using the simulation results from experiment 8.

Appendix J

Chapter 6 supplement: Additional information on methods

J.1 Replenishment policies

J.1.1 Neural network replenishment policies

As set out in Section 6.4.4.2, I used the approach proposed by Vanvuchelen and Boute [293] to obtain order quantities for each product in the two and eight product scenarios. The process for mapping the continuous neural network output $\hat{a}_t^p$ for each product p into an integer value, $\hat{A}_t^{r,p}$, that can be used as an order-up-to level $\mathtt{S}$ or directly as the order quantity is set out in Equation J.1.

$$\hat{A}_t^{r,p} = \left\lceil \frac{\text{clip}(\hat{a}_t^p, \hat{a}_{\min}, \hat{a}_{\max}) - \hat{a}_{\min}}{\hat{a}_{\max} - \hat{a}_{\min}} (\hat{A}_{\max}^r - \hat{A}_{\min}^r) + \hat{A}_{\min}^r \right\rceil \tag{J.1}$$

where

$$\text{clip}(\hat{a}_t^p, \hat{a}_{\min}, \hat{a}_{\max}) = \begin{cases} \hat{a}_{\min} & \text{if } \hat{a}_t^p < \hat{a}_{\min} \\ \hat{a}_t^p & \text{if } \hat{a}_{\min} \leq \hat{a}_t^p \leq \hat{a}_{\max} \\ \hat{a}_{\max} & \text{if } \hat{a}_t^p > \hat{a}_{\max} \end{cases} \tag{J.2}$$

I set $\hat{a}_{\min} = -2$, $\hat{a}_{\max} = 2$ and $\hat{A}_{\min}^r = 0$ for each experiment. The value for $\hat{A}_{\max}^r$ was set depending on the properties of a problem.

For the single and two product scenarios, with $\hat{A}_{\max}^r = 10$ for each product. In

the sensitivity analyses for the two product scenario $\hat{A}^r_{\max}$ was increased to match the higher maximum order quantity $A^r_{\max}$ when the mean demand was increased.

For the eight product scenario, $\hat{A}^r_{\max} = 25$ for each product in the high demand case and $\hat{A}^r_{\max} = 5$ for each product in the low demand case. These are less than the maximum order quantities, 60 and 15 respectively. The values were set to include the order-up-to parameters identified when fitting the heuristic base stock policies because, during preliminary experiments, I found that allowing a wide range of order values led to poorer policy performance.

J.2 Fitting policies to optimize a single objective

J.2.1 Neuroevolution

In Algorithm 6 I present pseudocode for OpenAI ES and in Algorithm 7 I present pseudocode for SimpleGA, both as implemented in the Python library evosax [121].

The pseudocode for OpenAI ES uses a simple gradient update for clarity but in practice I used the default gradient-based optimizer in the evosax implementation of the algorithm: Adam [326].

J.3 Supervised pretraining of policies

I used supervised learning to train neural networks that matched the outputs of heuristic replenishment policies for the single product scenario, and heuristic replenishment and issuing policies for the eight product scenario. The parameters for the neural networks were used to initialize policy search using OpenAI ES and SimpleGA.

For OpenAI ES, the initial mean parameters were set equal to the pre-trained parameters. For SimpleGA, half of the initial set of elite solutions was populated with the pretrained parameters. In cases where the replenishment and issuing policies were jointly optimized, but only one policy was being initialized using parameters from supervised pretraining, a single set of parameters for the other policy was randomly sampled following the standard initialization procedure.

The initial value of the best set of parameters, θ_{best}, was set equal to the set of

Algorithm 6 OpenAI ES in evosax

Initialize neural network parameters θ
Initialize learning rate α
Initialize learning rate decay factor d_α
Initialize minimum learning rate $\alpha_{\min}$
Initialize mutation noise standard deviation σ
Initialize mutation noise standard deviation decay factor d_σ
Initialize minimum mutation noise standard deviation $\sigma_{\min}$
Initialize $f_{\text{best}} = \infty$
for $k = 1 \rightarrow K$ **do**
    Initialize new generation: $\mathbb{P} \leftarrow \emptyset$
    for $i = 1 \rightarrow N$ **do**
        Sample noise: $\epsilon_i \sim \mathcal{N}(0, I)$
        Add noise to parameters: $\theta_i = \theta + \sigma\epsilon_i$
        Evaluate fitness: $f_i = F(\theta_i)$
        $\mathbb{P} \leftarrow \mathbb{P} \cup \{(\theta_i, f_i)\}$
        if $f_i < f_{\text{best}}$ **then**
            $f_{\text{best}} \leftarrow f_i$
            $\theta_{\text{best}} \leftarrow \theta_i$
        end if
    end for
    Update parameters: $\theta \leftarrow \theta - \alpha \frac{1}{N\sigma} \sum_{i=1}^{N} f_i \epsilon_i$
    Apply learning rate decay: $\alpha \leftarrow \max(d_\alpha \alpha, \alpha_{\min})$
    Apply mutation noise decay: $\sigma \leftarrow \max(d_\sigma \sigma, \sigma_{\min})$
end for
return θ_{best}

parameters used for the initialization, and the initial value of the best fitness, f_{best}, was set equal to the fitness given by this set of parameters on a set of training episodes prior to starting the policy search process. When all of the policies being optimized have been pretrained, f_{best} should be approximately equal to the performance given by the heuristic policies used for the pretraining, however when the policies are jointly optimized and only one policy has been pretrained the initial performance is likely to be worse.

J.3.1 Single product scenario

J.3.1.1 Replenishment policy

The inputs to the neural network were observations for the replenishment agent. For each setting, I generated all of the possible observations. When $m = 5$,

Algorithm 7 SimpleGA in evosax

Initialize neural network parameters for a population of E elite individuals, $\mathbb{E} = \{\theta_e \; \forall e \in \{1, ..., E\}\}$
Initialize mutation noise standard deviation σ
Initialize mutation noise standard deviation decay factor d_σ
Initialize minimum mutation noise standard deviation $\sigma_{\min}$
Initialize $f_{\text{best}} = \infty$
for $k = 1 \rightarrow K$ **do**
    Initialize new generation: $\mathbb{P} \leftarrow \emptyset$
    for $i = 1 \rightarrow N$ **do**
        Select elite individual as parent: $e \sim \text{Uniform}(1, E)$
        Sample noise: $\epsilon \sim \mathcal{N}(0, I)$
        Add noise to parent parameters: $\theta_i = \theta_e + \sigma\epsilon$
        Evaluate fitness: $f_i = F(\theta_i)$
        $\mathbb{P} \leftarrow \mathbb{P} \cup \{(\theta_i, f_i)\}$
        if $f_i < f_{\text{best}}$ **then**
            $f_{\text{best}} \leftarrow f_i$
            $\theta_{\text{best}} \leftarrow \theta_i$
        end if
    end for
    $\mathbb{P} \leftarrow \mathbb{P} \cup \mathbb{E}$
    Sort $\mathbb{P}$ in ascending order by fitness
    Update elite population: $\mathbb{E} \leftarrow \mathbb{P}[1 : E]$
    Apply mutation noise decay: $\sigma \leftarrow \max(d_\sigma\sigma, \sigma_{\min})$
end for
return θ_{best}

observations where the total inventory on hand or in-transit was greater than 15 units were excluded. This limit was set to ensure there was some coverage of states where no stock should be ordered, but to prevent them from making up the majority of states which, in preliminary experiments, I found made it difficult to learn the heuristic policy.

Labels for the training set were created using the heuristic base stock policy, with order-up-to level $\texttt{S}$ set based on the simulation optimization experiments from Section 4.5, as listed in Table G.1 in Appendix G. For the order-up-to neural networks, the label for each observation in the training set was the order-up-to parameter $\texttt{S}$. For the direct-action neural networks, the label for the observation was $[\texttt{S} - I]^+$, where I was the total number of units in stock and in-transit for the observation.

The loss function was the ordinal categorical cross entropy loss [327]:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\left(1 + \frac{|\arg\max_k \mathbf{y} - \arg\max_k \hat{\mathbf{y}})|}{k-1}\right) \sum_k y_k \log \hat{y}_k \tag{J.3}$$

where k is the number of output categories (in this scenario, $A_{\max} + 1$), $\mathbf{y}$ is a one-hot vector representing the true label, $\hat{\mathbf{y}}$ is a vector of output values of un-normalized outputs from the neural network, y_k is the kth element of $\mathbf{y}$ and $\hat{y}_k$ is the kth element of $\hat{\mathbf{y}}$. This loss function was selected because the output is discrete but there is an ordinal relationship between the order quantities. The cross-entropy loss for each sample is weighted by the absolute difference between predicted and actual class labels, so a larger penalty was imposed for order quantities further from the target.

During training, performance was assessed every 10 epochs using two metrics: accuracy on the training set of observations, and the performance gap between the mean return given by the heuristic policy and the learned policy on 10,000 validation episodes. Hyperparameters for the supervised pretraining were manually tuned for each experiment to achieve a 0% performance gap. In every, the network also achieved 100% accuracy on the training set, demonstrating that the heuristic policy had effectively memorized the heuristic policy. The hyperparameters are presented in Table K.4.

J.3.2 Eight product scenario

J.3.2.1 Replenishment policy

The inputs to the neural network were observations for the replenishment agent. For each setting, I collected observations from 14,000 replenishment steps in the adapted single-agent environment, using the target heuristic replenishment policy and corresponding heuristic issuing policy that was selected for further evaluation and pretraining after conducting the simulation optimization experiments for the experimental setting. To encourage diversity in the observations, for each step the order quantity for each product had a 5% chance of being replaced by a random order quantity sampled from a Poisson distribution with a mean equal to the

order-up-to level for that product, $\mathbb{S}^p$. Duplicates observations were not removed.

Labels for the training set were created using the heuristic base stock policy, with order-up-to parameter $\mathbb{S}^p$ for each product previously fit using simulation optimization. The task was treated as a regression problem, using the mean squared error as the loss function, with the network trained to predict a continuous target for each of the eight products prior to the mapping process described in Equation J.1. Only direct-action neural network policies were used, and the continuous target for each product was calculated from the order-up-to level $\mathbb{S}^p$ and the total stock on hand and in-transit for product p, I^p using Equation J.4 which reverses the mapping process described in Equation J.1 so that the continuous target is in the middle of the range that corresponds to the target order quantity.

$$y^p = \left(\frac{[\mathbb{S}^p - I^p]^+ - \hat{A}_{\min}}{\hat{A}_{\max} - \hat{A}_{\min}} (\hat{a}_{\max} - \hat{a}_{\min}) \right) + \hat{a}_{\min} - \frac{\hat{a}_{\max} - \hat{a}_{\min}}{2(\hat{A}_{\max} - \hat{A}_{\min})} \tag{J.4}$$

During training, performance was assessed every 10 epochs using the performance gap between the mean return given by the heuristic policy and the learned policy on 10,000 validation episodes. Hyperparameters for the supervised pretraining were manually tuned for each experiment to achieve a performance gap of less than 0.2%. The hyperparameters are presented in Table K.11.

J.3.2.2 Issuing policy

The inputs to the neural network were observations for the issuing agent. For each setting, I collected observations from 20,000 steps in the multi-agent environment, using the target heuristic issuing policy and corresponding heuristic replenishment policy that was selected for further evaluation and pretraining after conducting the simulation optimization experiments. To encourage diversity in the observations, for each replenishment step the order quantity for each product had a 5% chance of being replaced by a random order quantity sampled from a uniform distribution over the integers between 0 and $A_{\max}$. Replenishment steps were removed from the 20,000 total steps but duplicate issuing observations were not removed.

Labels for the training set were created using the heuristic issuing policy. The task was treated as a classification problem, using the cross-entropy loss.

During training, performance was assessed every 10 epochs using two metrics: accuracy on the training set of observations, the performance gap between the mean return given by the heuristic policy and the learned policy on 10,000 validation episodes. Hyperparameters for the supervised pretraining were manually tuned for each experiment to achieve a performance gap of less than 0.2% performance gap and at least 99.1% accuracy on the training set. The hyperparameters are presented in Table K.12.

J.4 Multi-objective optimization

J.4.1 NSGA-II

In Algorithm 8 I present pseudocode for NSGA-II as implemented using the Python library pymoo [310].

Algorithm 8 NSGA-II in pymoo

Initialize neural network parameters and evaluate fitness for a population of P individuals: $\mathbb{P} = \{(\theta_i, F(\theta_i))\ \forall i \in \{1, ..., P\}\}$
Initialize mutation noise standard deviation σ
Initialize crossover probability p_c
Initialize crossover spread η
Perform non-dominated sorting on $\mathbb{P}$ to form fronts $\mathbb{F}_1, \mathbb{F}_2, \ldots$
Assign crowding distance to individuals in each front
for $k = 1 \rightarrow K$ **do**
    $\mathbb{H} \leftarrow N$ parents from $\mathbb{P}$ using binary tournament selection based on front rank and crowding distance
    for $i = 1 \rightarrow N$ **do**
        Generate a new individual using crossover: $\theta_i = \text{Crossover}(\mathbb{H}, \eta, p_c)$
        Sample noise: $\epsilon \sim \mathcal{N}(0, I)$
        Add noise to parameters: $\theta_i \leftarrow \theta_i + \sigma\epsilon$
        Evaluate fitness: $f_i = F(\theta_i)$
        $\mathbb{P} \leftarrow \mathbb{P} \cup \{(\theta_i, f_i)\}$
    end for
    Perform non-dominated sorting on $\mathbb{P}$ to form new fronts $\mathbb{F}_1, \mathbb{F}_2, \ldots$
    $\mathbb{P}' \leftarrow \emptyset$
    $j \leftarrow 1$
    while $|\mathbb{P}'| + |\mathbb{F}_j| \leq P$ **do**
        Assign crowding distance to individuals in $\mathbb{F}_j$
        $\mathbb{P}' \leftarrow \mathbb{P}' \cup \mathbb{F}_j$
        $j \leftarrow j + 1$
    end while
    Sort $\mathbb{F}_j$ by crowding distance in descending order
    Fill remaining slots in $\mathbb{P}'$: $\mathbb{P}' \leftarrow \mathbb{P}' \cup \mathbb{F}_j[1 : (N - |\mathbb{P}'|)]$
    $\mathbb{P} \leftarrow \mathbb{P}'$
end for
return $\mathbb{F}_1$

Appendix K

Chapter 6 supplement: Hyperparameters

Hyperparameters were tuned using the Python library wandb [307], and search distributions are reported using the names from that library, along with any additional parameters required for a given distribution.

K.1 Single product scenario

K.1.1 Main results

The hyperparameter settings and search ranges for PPO are set out in Table K.1. When the policies were jointly fit, separate values for each of the hyperparameters were included in the sweep for the replenishment and issuing policies.

Parameter	Fixed value	Search distribution	Min value	Max value	Choices
num_envs	200				
replenishment.num_steps	20				
replenishment.total_timesteps	1e6				
replenishment.num_minibatches	10				
issuing.num_steps	80				
issuing.total_timesteps	4e6				
issuing.num_minibatches	40				
anneal_lr	True				
dual_clip	False				
norm_adv	True				
value_clip	True				
partial_episode_bootstrapping	True				
update_epochs		categorical			[2,4,8,16]
lr		log_uniform_values	5e-6	1e-2	
gamma		uniform	0.7	0.99	
gae_lambda		uniform	0.7	0.99	
clip_eps		uniform	0.05	0.5	
ent_coef		log_uniform_values	0.0001	0.05	
vf_coef		uniform	0.1	1.0	
max_grad_norm		uniform	0.05	1.0	

Table K.1: Hyperparameter search ranges for PPO on the single product scenario

The hyperparameter settings and search ranges for OpenAI ES are set out in Table K.2.

Parameter	Fixed value	Search distribution	Min value	Max value	Choices
num_generations	100				
popsize	150				
num_train_rollouts	100				
init_params_abs_max	0				
lrate_init		log_uniform_values	0.001	0.5	
sigma_init		log_uniform_values	0.001	0.5	
sigma_decay		categorical			[1.0, 0.999, 0.995, 0.99, 0.95]
lrate_decay		categorical			[1.0, 0.999, 0.995, 0.99, 0.95]

Table K.2: Hyperparameter search ranges for OpenAI ES on the single product scenario

The hyperparameter settings and search ranges for SimpleGA are set out in Table K.3.

Parameter	Fixed value	Search distribution	Min value	Max value	Choices
num_generations	100				
popsize	150				
num_train_rollouts	100				
init_params_abs_max	0				
elite_ratio		q_uniform (q=0.05)	0.05	0.6	
sigma_init		log_uniform_values	0.001	0.5	
sigma_decay		categorical			[1.0, 0.999, 0.995, 0.99, 0.95]

Table K.3: Hyperparameter search ranges for SimpleGA on the single product scenario

K.1.2 Supervised pretraining of policies

The hyperparameter settings for supervised pretraining of the replenishment policy are set out in Table K.4.

Parameter	$m = 2, L = 2$	$m = 5, L = 2$
learning_rate	0.001	0.01
num_epochs	250	20
batch_size	16	64

Table K.4: Hyperparameter settings for supervised pretraining of the replenishment policy for the single product scenario

K.1.3 Estimating the Pareto frontier for service level and wastage

The hyperparameter settings for the outer-loop approach using SimpleGA are set out in Table K.5.

Parameter	Fixed value
num_generations	100
popsize	150
num_train_rollouts	100
init_params_abs_max	0
elite_ratio	0.50
sigma_init	0.07
sigma_decay	1.00

Table K.5: Hyperparameter settings for SimpleGA for estimating the Pareto frontier with the outer-loop method on the single product scenario

The hyperparameter settings and search ranges for NSGA-II are set out in Table K.6.

Parameter	Fixed value	Search distribution	Min value	Max value	Choices
num_generations	100				
pop_size	2,000				
n_offsprings	150				
num_train_rollouts	100				
mutation.sigma		log_uniform_values	0.01	2.0	
crossover.prob		uniform	0.1	1.0	
crossover.eta		log_uniform_values	0.5	50	
xu		uniform	0.0	10.0	

Table K.6: Hyperparameter search ranges for NSGA-II for estimating the Pareto frontier on the single product scenario

K.2 Two product scenario

K.2.1 Main results

The hyperparameter settings and search ranges for SimpleGA are set out in Table K.7.

Parameter	Fixed value	Search distribution	Min value	Max value	Choices
num_generations	100				
popsize	150				
num_train_rollouts	100				
init_params_abs_max	0				
elite_ratio		q_uniform (q=0.05)	0.05	0.6	
sigma_init		log_uniform_values	0.001	0.5	
sigma_decay		categorical			[1.0, 0.999, 0.995, 0.99, 0.95]

Table K.7: Hyperparameter search ranges for SimpleGA on the two product scenario

K.2.2 Sensitivity analyses

The hyperparameter settings for SimpleGA are set out in Table K.8.

Parameter	Fixed value
num_generations	100
popsize	150
num_train_rollouts	100
init_params_abs_max	0
elite_ratio	0.50
sigma_init	0.07
sigma_decay	1.00

Table K.8: Hyperparameter settings for SimpleGA for sensitivity analyses on the two product scenario

K.2.3 Estimating the Pareto frontier for service level and wastage

The hyperparameter settings for the outer-loop approach using SimpleGA are set out in Table K.9.

Parameter	Fixed value
num_generations	100
popsize	150
num_train_rollouts	100
init_params_abs_max	0
elite_ratio	0.50
sigma_init	0.07
sigma_decay	1.00

Table K.9: Hyperparameter settings for SimpleGA for estimating the Pareto frontier with the outer-loop method on the two product scenario

The hyperparameter settings and search ranges for NSGA-II are set out in Table K.10.

Parameter	Fixed value	Search distribution	Min value	Max value	Choices
num_generations	100				
pop_size	2,000				
n_offsprings	150				
num_train_rollouts	100				
mutation.sigma		log_uniform_values	0.01	2.0	
crossover.prob		uniform	0.1	1.0	
crossover.eta		log_uniform_values	0.5	50	
xu		uniform	0.0	10.0	

Table K.10: Hyperparameter search ranges for NSGA-II for estimating the Pareto frontier on the two product scenario

K.3 Eight product scenario

The hyperparameters used for supervised pretraining of the replenishment and issuing policies are set out in Table K.11 and Table K.12 respectively.

Parameter	**Low demand**		**High demand**	
	Max sub wastage	Max sub shortage	Max sub wastage	Max sub shortage
learning_rate	0.005	0.005	0.005	0.005
num_epochs	20	20	50	50
batch_size	64	64	64	64

Table K.11: Hyperparameter settings for supervised pretraining of the replenishment policy for the eight product scenario

Parameter	**Low demand**		**High demand**	
	Max sub wastage	Max sub shortage	Max sub wastage	Max sub shortage
learning_rate	0.005	0.005	0.005	0.005
num_epochs	50	30	30	30
batch_size	64	64	64	64

Table K.12: Hyperparameter settings for supervised pretraining of the issuing policy for the eight product scenario

The hyperparameter settings and search ranges for fitting policies with SimpleGA are set out in Table K.13. The parameter init_params_abs_max was only used for parameter initialization when a neural network policy had not be pretrained.

Parameter	Fixed value	Search distribution	Min value	Max value	Choices
num_generations	100				
popsize	150				
num_train_rollouts	100				
init_params_abs_max		uniform	0	5	
elite_ratio		q_uniform (q=0.05)	0.05	0.6	
sigma_init		log_uniform_values	0.001	0.5	
sigma_decay		categorical			[1.0, 0.999, 0.995, 0.99, 0.95]

Table K.13: Hyperparameter search ranges for SimpleGA on the eight product scenario

Appendix L

Chapter 6 supplement: Additional results

L.1 Two product scenario

L.1.1 Sensitivity analyses

In Table L.1 I present the full results for the sensitivity analyses when fitting replenishment policies with each of the three heuristic issuing policies.

L.2 Eight product scenario

L.2.1 Parameters for the heuristic base stock policies

In Table L.2 I present the order-up-to parameters, `S` for the base stock policies fit for the eight product scenario using simulation optimization. The performance of these policies is reported in Tables 6.12, 6.13, 6.14 and 6.15 with one table for each combination of demand and maximum substitution cost.

L.2.2 Policy plots for the neural network replenishment policies

In Figures L.1, L.2, L.3 and L.4 I present plots showing the orders made by the best-performing neural network replenishment policies for each of the four experimental settings. For each experimental setting, this was the replenishment policy fit using SimpleGA, with supervised pretraining, and a heuristic issuing policy. For each blood type, the plots visualize the number of units of that product

	Simulation optimization and heuristic issuing			**SimpleGA and heuristic issuing**		
	Exact	Priority	Oldest compatible	Exact	Priority	Oldest compatible
Base setting						
	17.41	**17.07**	17.09	16.97	**16.72**	16.72
Lead time L						
0	16.77	**16.44**	16.53	16.77	**16.39**	16.41
1	17.41	**17.07**	17.09	16.97	**16.72**	16.72
2	17.66	**17.34**	17.36	16.97	16.72	**16.69**
Maximum useful life m						
2	17.41	**17.07**	17.09	16.97	**16.72**	16.72
3	16.33	**16.17**	16.33	16.24	16.16	**16.12**
4	15.95	**15.88**	16.15	15.92	**15.82**	15.93
5	15.82	**15.79**	16.09	15.86	**15.76**	15.86
Product probabilities [A:B]						
[0.10, 0.90]	**16.08**	**16.08**	**16.08**	15.84	**15.84**	15.84
[0.25, 0.75]	17.11	**16.80**	16.81	16.87	**16.45**	16.47
[0.50, 0.50]	17.41	**17.07**	17.09	16.97	**16.72**	16.72
[0.75, 0.25]	17.11	**16.81**	**16.81**	16.87	**16.60**	16.60
[0.90, 0.10]	16.08	**16.00**	**16.00**	15.84	**15.75**	15.75
Shortage cost C_s : wastage cost C_w						
5:7	17.41	**17.07**	17.09	16.97	**16.72**	16.72
7:7	20.48	**19.24**	19.30	19.89	**18.73**	18.85
14:7	26.21	23.26	**23.16**	25.31	22.61	**22.37**
21:7	29.77	25.54	**25.30**	28.69	24.66	**24.50**
28:7	32.09	27.11	**26.79**	30.72	26.26	**25.81**
35:7	34.29	28.32	**27.91**	32.57	27.16	**26.89**
5:5	17.10	**16.92**	16.96	16.87	**16.51**	16.54
5:10	17.65	17.29	**17.28**	17.12	**16.84**	16.88
5:15	17.96	17.54	**17.54**	17.37	**17.15**	17.17
5:20	18.26	17.79	**17.77**	17.62	17.39	**17.27**
5:25	18.57	18.02	**17.95**	17.95	17.68	**17.62**
Substitution cost $C_m^{b,a}$						
0.50	17.41	16.12	**16.08**	16.97	**15.89**	15.89
1.0	17.41	16.53	**16.51**	16.97	16.17	**16.16**
2.0	17.41	**17.07**	17.09	16.97	**16.72**	16.72
4.0	**17.41**	17.68	17.84	**16.97**	17.42	17.52
Mean daily demand						
4	17.41	**17.07**	17.09	16.97	**16.72**	16.72
8	30.83	**30.60**	30.70	30.57	30.30	**30.28**
16	56.54	**56.41**	56.54	56.50	56.18	**56.14**
32	107.18	**107.10**	107.17	107.48	107.49	**107.08**

The daily cost was calculated for each episode and the mean over the 10,000 evaluation episodes is reported. The lowest cost, indicating the best performance, is highlight in **bold** for each method of fitting the replenishment policy in each row. The bold values are those reported in Table 6.7.

Table L.1: Mean daily cost for the sensitivity analyses on the two product scenario.

			S for ABO/RhD blood type							
Demand	Maximum substitution cost	Issuing policy	O-	O+	A-	A+	B-	B+	AB-	AB+
Low	650	Exact	1	3	1	3	0	1	0	0
Low	650	Priority	2	0	1	3	0	1	0	0
Low	650	Oldest compatible	2	0	2	2	0	1	0	0
Low	3,250	Exact	1	3	1	3	0	1	0	0
Low	3,250	Priority	1	2	1	3	1	0	0	0
Low	3,250	Oldest compatible	1	2	2	2	0	1	0	0
High	650	Exact	4	20	4	18	1	5	1	2
High	650	Priority	0	16	4	18	6	3	0	1
High	650	Oldest compatible	5	13	5	16	8	0	0	1
High	3,250	Exact	4	20	4	18	1	5	1	2
High	3,250	Priority	3	18	4	17	2	5	2	0
High	3,250	Oldest compatible	6	18	5	14	1	3	0	3

Table L.2: The best combinations of parameters for the base stock replenishment policies fit using simulation optimization for each of the experimental settings for the eight product scenario

ordered against the total number of units of that product in stock when the order was placed, recorded over 5,000 steps in the adapted single-agent reinforcement learning environment.

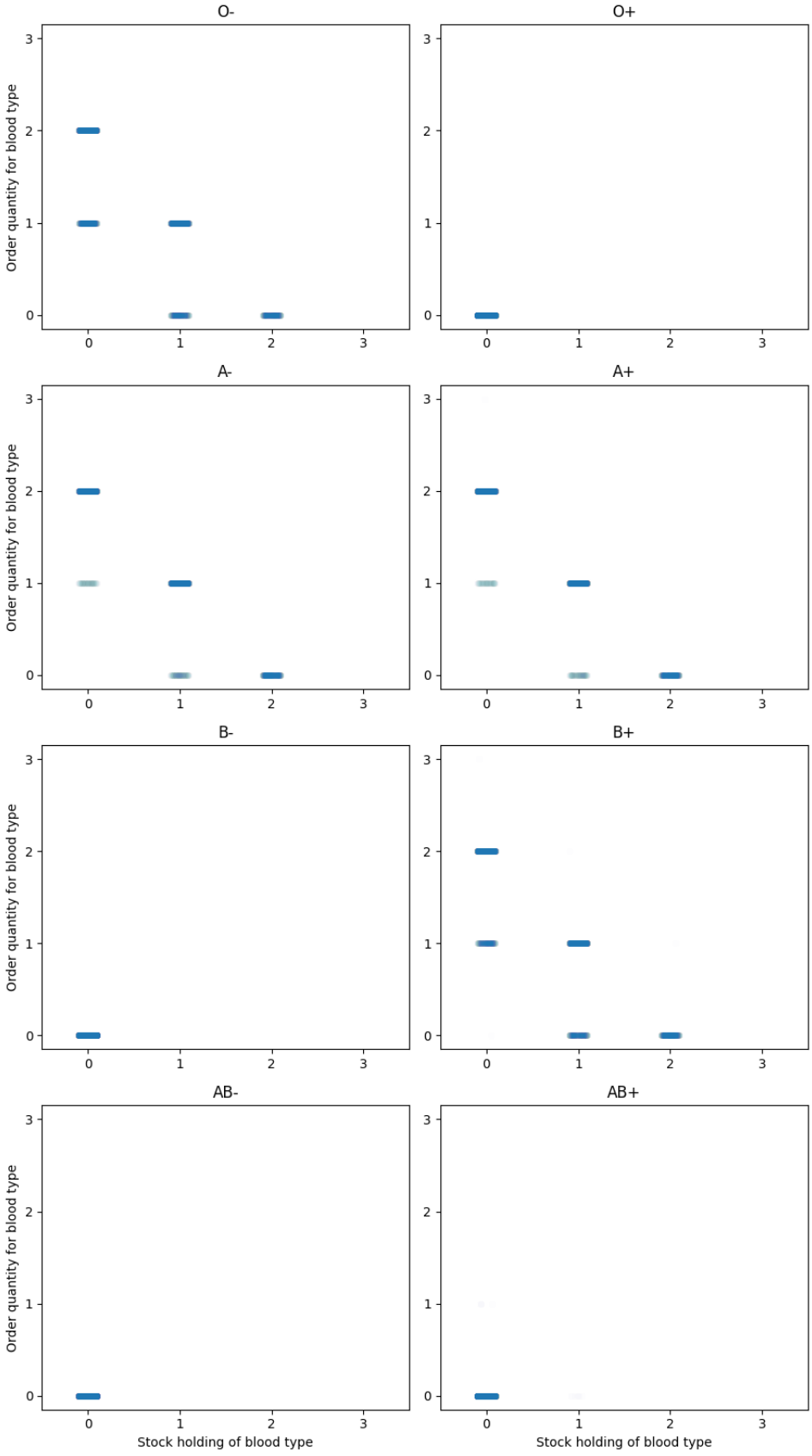


Figure L.1: Order quantity versus stock holding for each ABO/RhD blood type for the best replenishment policy fit using SimpleGA for the experiment setting with low demand and maximum substitution cost equal to the wastage cost

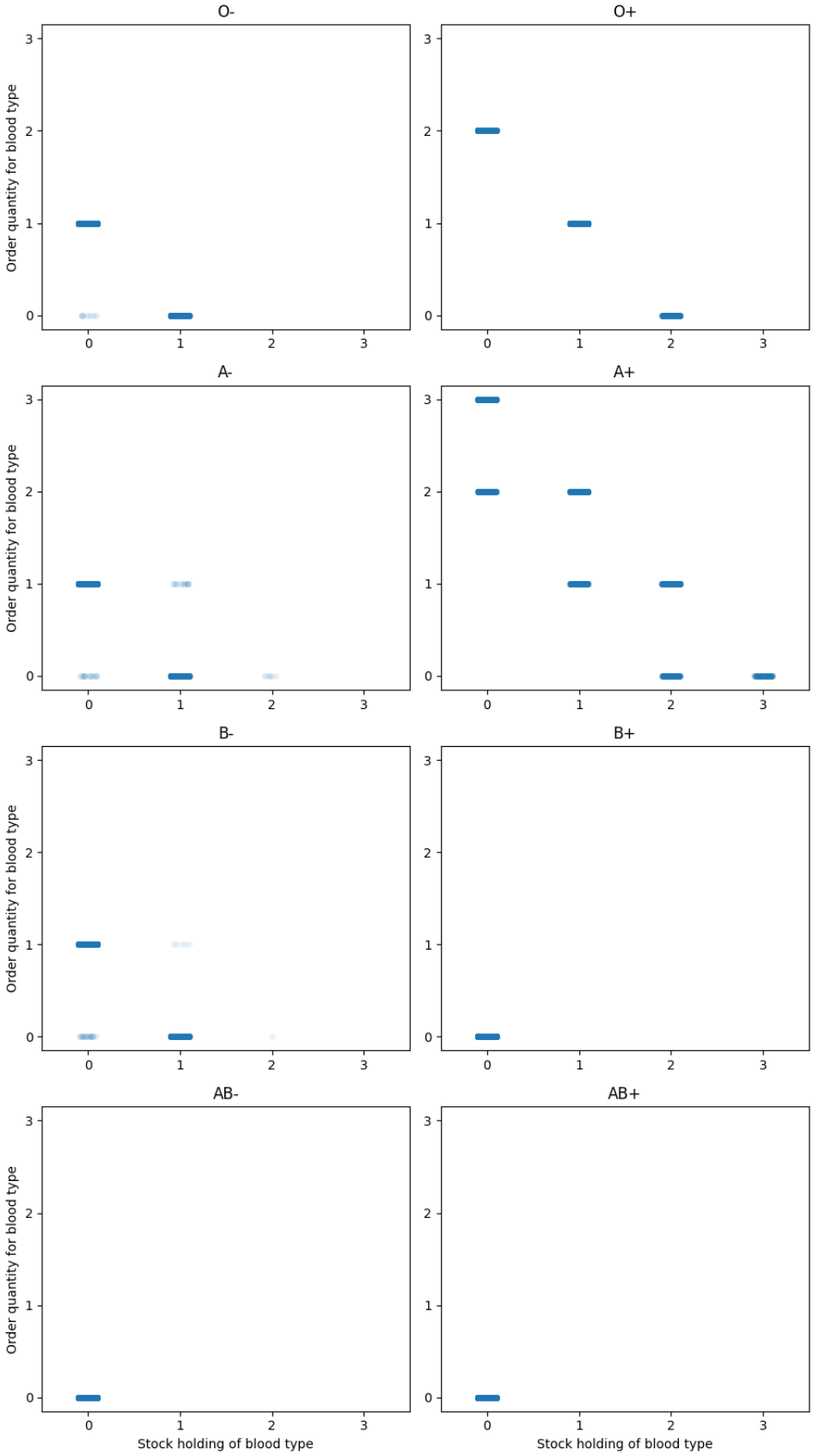


Figure L.2: Order quantity versus stock holding for each ABO/RhD blood type for the best replenishment policy fit using SimpleGA for the experiment setting with low demand and maximum substitution cost equal to the shortage cost

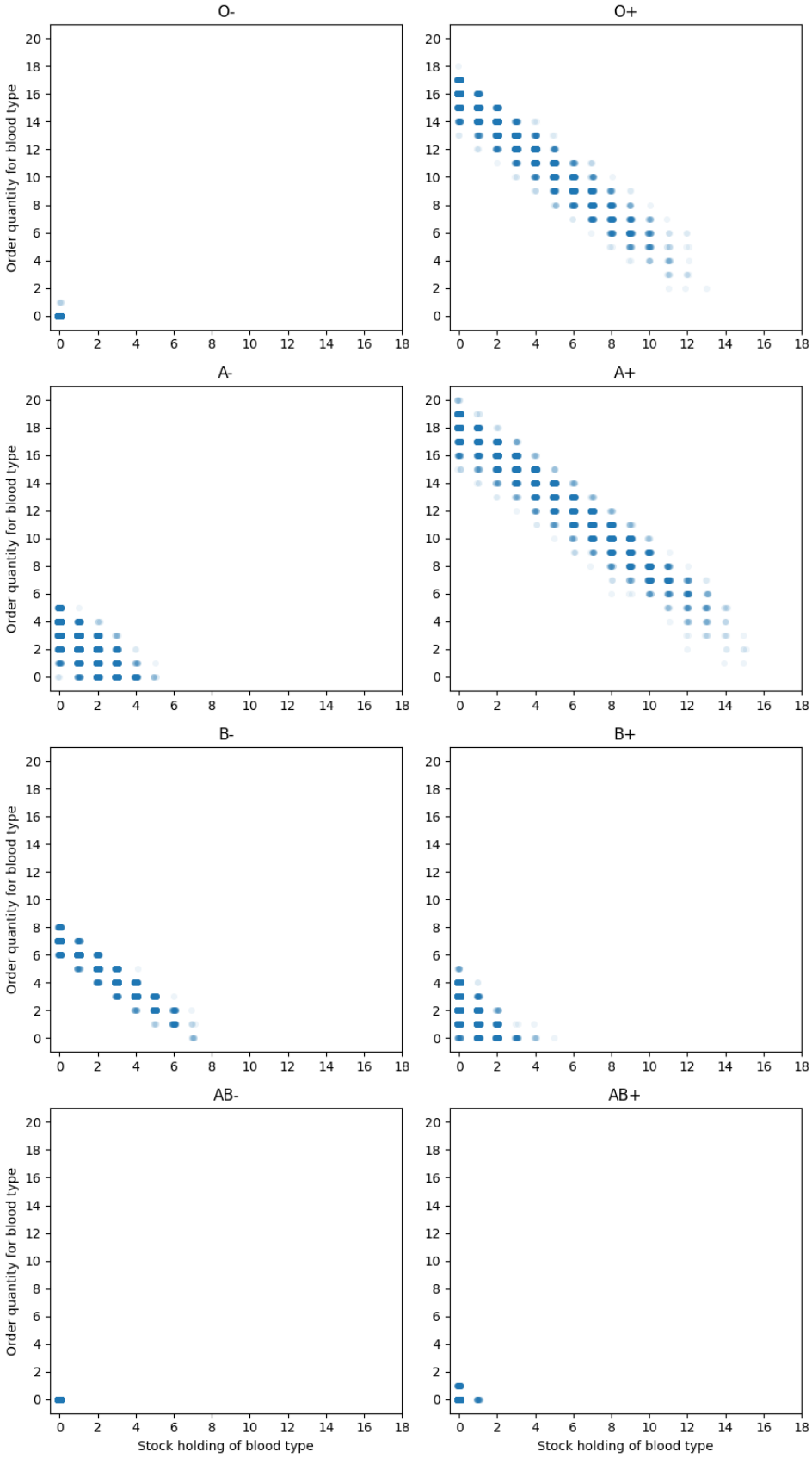


Figure L.3: Order quantity versus stock holding for each ABO/RhD blood type for the best replenishment policy fit using SimpleGA for the experiment setting with high demand and maximum substitution cost equal to the wastage cost

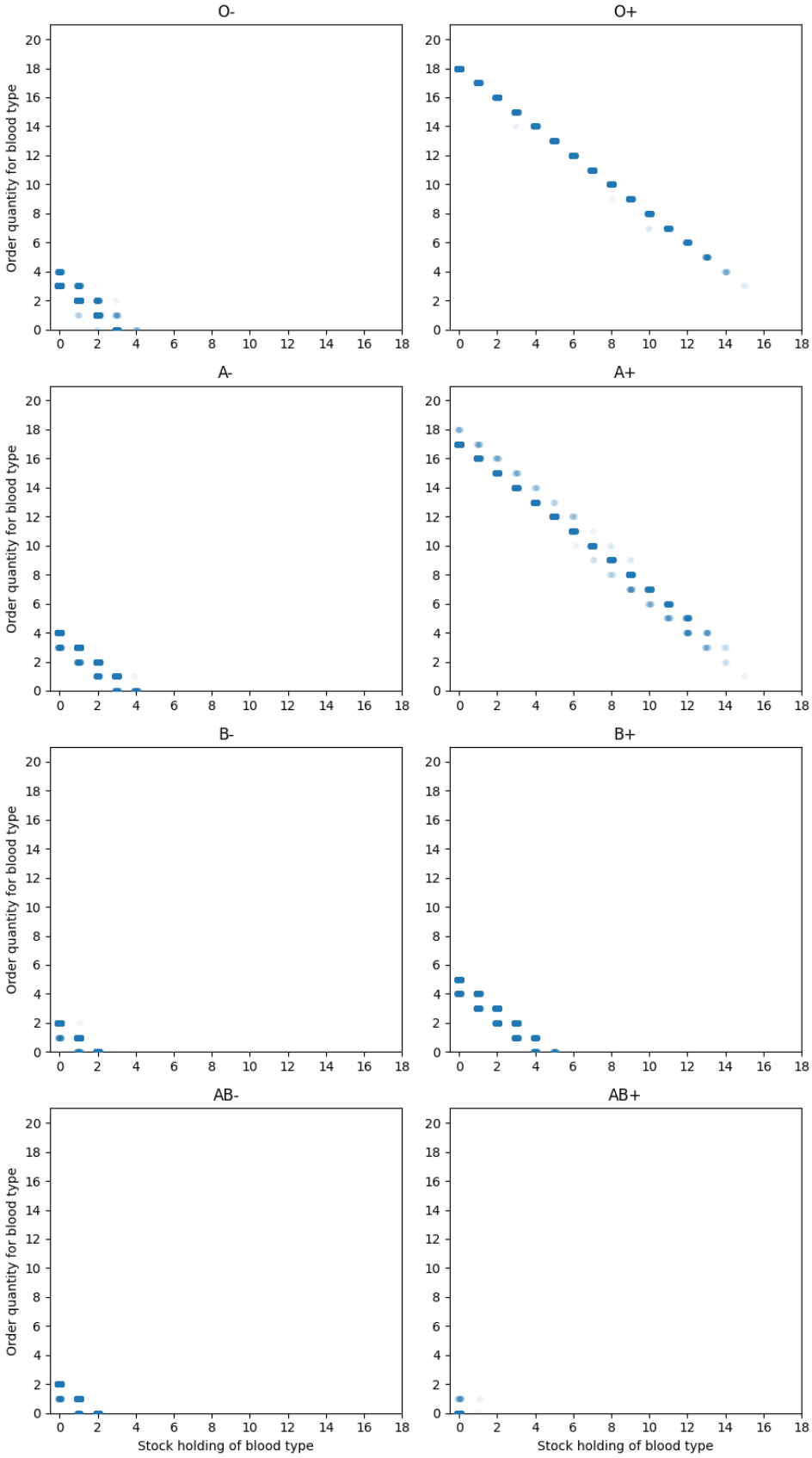


Figure L.4: Order quantity versus stock holding for each ABO/RhD blood type for the best replenishment policy fit using SimpleGA for the experiment setting with high demand and maximum substitution cost equal to the shortage cost

Bibliography

[1] A. W. Flint *et al.*, "Is platelet expiring out of date? A systematic review," *Transfusion Medicine Reviews*, vol. 34, no. 1, pp. 42–50, 2020. https://doi.org/10.1016/j.tmrv.2019.08.006.

[2] Blood Stocks Management Scheme, "NHSBT hospital blood supply chain annual report 2017/18," 2018. https://nhsbtdbe.blob.core.windows.net/umbraco-assets-corp/15951/nhsbt-annual-report-2017-18.pdf.

[3] Blood Stocks Management Scheme, "2023 BSMS 10 year component review," 2023. https://nhsbtdbe.blob.core.windows.net/umbraco-assets-corp/31272/bsms-10-year-component-review-2.pdf.

[4] J. Gottschall *et al.*, "The epidemiology of platelet transfusions: An analysis of platelet use at 12 US hospitals," *Transfusion*, vol. 60, no. 1, pp. 46–53, 2020. https://doi.org/10.1111/trf.15637.

[5] S. Valsami *et al.*, "ABO and RhD matching in platelet transfusions: Real-world data," *World Academy of Sciences Journal*, vol. 5, no. 6, pp. 1–5, 2023. https://doi.org/10.3892/wasj.2023.211.

[6] S. Nahmias, "Perishable inventory theory: A review," *Operations Research*, vol. 30, no. 4, pp. 680–708, 1982. https://doi.org/10.1287/opre.30.4.680.

[7] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. https://doi.org/10.1038/nature14236.

[8] Z. King *et al.*, "Machine learning for real-time aggregated prediction of hospital admission for emergency patients," *npj Digital Medicine*, vol. 5, no. 1, pp. 1–12, 2022. https://doi.org/10.1038/s41746-022-00649-y.

[9] A. F. Osorio *et al.*, "A structured review of quantitative models in the blood supply chain: A taxonomic framework for decision-making," *International Journal of Production Research*, vol. 53, no. 24, pp. 7191–7212, 2015. https://doi.org/10.1080/00207543.2015.1005766.

[10] O. Ejohwomu *et al.*, "A resilient approach to modelling the supply and demand of platelets in the United Kingdom blood supply chain," *International Journal of Management Science and Engineering Management*, vol. 16, no. 2, pp. 143–150, 2021. https://doi.org/10.1080/17509653.2021.1892548.

[11] L. Guan *et al.*, "Big data modeling to predict platelet usage and minimize wastage in a tertiary care system," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, no. 43, pp. 11 368–11 373, 2017. https://doi.org/10.1073/pnas.1714097114.

[12] G. P. Prastacos, "Blood inventory management: An overview of theory and practice," *Management Science*, vol. 30, no. 7, pp. 777–800, 1984. https://doi.org/10.1287/mnsc.30.7.777.

[13] D. Norfolk and United Kingdom Blood Services, *Handbook of Transfusion Medicine*, 5th ed. London, UK: Her Majesty's Stationery Office, 2013, ISBN: 978-0-11-706846-9.

[14] E. M. Wood *et al.*, "An update on indications for platelet transfusion," *ISBT Science Series*, vol. 11, pp. 170–176, S2 2016. https://doi.org/10.1111/voxs.12264.

[15] J. McCullough, "Overview of platelet transfusion," *Seminars in Hematology*, vol. 47, no. 3, pp. 235–242, 2010. https://doi.org/10.1053/j.seminhematol.2010.04.001.

[16] "NHS Blood and Transplant price list 2023/2024." (2023), https://nhsbtdbe.blob.core.windows.net/umbraco-assets-corp/29949/price_list_bc_nhs_contract_cost-per-item_2023-24.pdf (accessed 2024-06-13).

[17] K. M. Prioli *et al.*, "Economic implications of FDA platelet bacterial guidance compliance options: Comparison of single-step strategies," *Transfusion*, vol. 62, no. 2, pp. 365–373, 2022. https://doi.org/10.1111/trf.16778.

[18] Z. M. Szczepiorkowski and M. B. Pagano, "Platelet components and bacterial contamination: Hospital perspective 2022," *Hematology*, vol. 2022, no. 1, pp. 430–436, 2022. https://doi.org/10.1182/hematology.2022000402.

[19] M. F. Murphy *et al.*, Eds., *Practical Transfusion Medicine*, 5th ed., Hoboken, NJ, USA: Wiley, 2017, ISBN: 978-1-119-12941-7.

[20] N. M. Dunbar, "Does ABO and RhD matching matter for platelet transfusion?" *Hematology*, vol. 2020, no. 1, pp. 512–517, 2020. https://doi.org/10.1182/hematology.2020000135.

[21] S. Maynard *et al.*, "Machine learning in transfusion medicine: A scoping review," *Transfusion*, vol. 64, no. 1, pp. 162–184, 2024. https://doi.org/10.1111/trf.17582.

[22] D. Ben-Israel *et al.*, "The impact of machine learning on patient care: A systematic review," *Artificial Intelligence in Medicine*, vol. 103, p. 101 785, 2020. https://doi.org/10.1016/j.artmed.2019.101785.

[23] J. Epah *et al.*, "From unit to dose: A machine learning approach for precise prediction of hemoglobin and iron content in individual packed red blood cell units," *Advanced Science*, vol. 9, no. 36, p. 2 204 077, 2022. https://doi.org/10.1002/advs.202204077.

[24] M. Zhang *et al.*, "Improving the quantitative analysis accuracy of bagged liquid components with strong scattering by multi-pathlength data fusion," *Infrared Physics & Technology*, vol. 99, pp. 39–44, 2019. https://doi.org/10.1016/j.infrared.2019.04.006.

[25] J. Rad *et al.*, "An AI-driven predictive modelling framework to analyze and visualize blood product transactional data for reducing blood products' discards," in *Artificial Intelligence in Medicine: 18th International Conference on Artificial Intelligence in Medicine (AIME 2020)*, Minneapolis, MN, USA, 25–28 August, 2020, pp. 192–202. https://doi.org/10.1007/978-3-030-59137-3_18.

[26] R. F. Xiang *et al.*, "Application of unsupervised machine learning to identify areas of blood product wastage in transfusion medicine," *Vox Sanguinis*, 2021. https://doi.org/10.1111/vox.13089.

[27] N. Li *et al.*, "Blood demand forecasting and supply management: An analytical assessment of key studies utilizing novel computational techniques," *Transfusion Medicine Reviews*, p. 150 768, 2023. https://doi.org/10.1016/j.tmrv.2023.150768.

[28] L. V. Snyder and Z.-J. Shen, *Fundamentals of Supply Chain Theory*, 2nd ed. Hoboken, NJ, USA: Wiley, 2019, ISBN: 978-1-119-02484-2.

[29] V. Chaudhary *et al.*, "State-of-the-art literature review on inventory models for perishable products," *Journal of Advances in Management Research*, vol. 15, no. 3, pp. 306–346, 2018. https://doi.org/10.1108/JAMR-09-2017-0091.

[30] S. Nahmias, *Perishable Inventory Systems*. New York, NY, USA: Springer, 2011, ISBN: 978-1-4419-7998-8.

[31] R. C. Elston, "Simulation d'un processus stochastique impliqué dans la gestion d'une banque de sang," *Biométrie-praximétrie*, vol. 3, pp. 129–140, 1962.

[32] R. C. Elston and J. C. Pickrel, "A statistical approach to ordering and usage policies for a hospital blood bank," *Transfusion*, vol. 3, no. 1, pp. 41–47, 1963. https://doi.org/10.1111/j.1537-2995.1963.tb04602.x.

[33] J. Beliën and H. Forcé, "Supply chain management of blood products: A literature review," *European Journal of Operational Research*, vol. 217, no. 1, pp. 1–16, 2012. https://doi.org/10.1016/j.ejor.2011.05.026.

[34] A. Pirabán *et al.*, "Survey on blood supply chain management: Models and methods," *Computers and Operations Research*, vol. 112, 2019. https://doi.org/10.1016/j.cor.2019.07.014.

[35] M. Cohen and W. Pierskalla, "Target inventory levels for a hospital blood bank or a decentralized regional blood banking system," *Transfusion*, vol. 19, no. 4, pp. 444–454, 1979. https://doi.org/10.1046/j.1537-2995.1979.19479250182.x.

[36] M. Dillon *et al.*, "A two-stage stochastic programming model for inventory management in the blood supply chain," *International Journal of Production Economics*, vol. 187, pp. 27–41, 2017. https://doi.org/10.1016/j.ijpe.2017.02.006.

[37] J. T. Blake *et al.*, "Simplified platelet ordering using shortage and outdate targets," *International Journal of Health Management and Information*, vol. 1, no. 2, pp. 145–166, 2010. https://www.researchgate.net/publication/268373290_Simplified_Platelet_Ordering_Using_Shortage_and_Outdate_Targets.

[38] B. E. Fries, "Optimal ordering policy for a perishable commodity with fixed lifetime," *Operations Research*, vol. 23, no. 1, pp. 46–61, 1975. https://doi.org/10.1287/opre.23.1.46.

[39] S. Nahmias, "Optimal ordering policies for perishable inventory—II," *Operations Research*, vol. 23, no. 4, pp. 735–749, 1975. https://doi.org/10.1287/opre.23.4.735.

[40] B. J. De Moor *et al.*, "Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management," *European Journal of Operational Research*, vol. 301, no. 2, pp. 535–545, 2022. https://doi.org/10.1016/j.ejor.2021.10.045.

[41] J. T. Blake, "On the use of operational research for managing platelet inventory and ordering," *Transfusion*, vol. 49, no. 3, pp. 396–401, 2009. https://doi.org/10.1111/j.1537-2995.2008.02061.x.

[42] R. Haijema and S. Minner, "Improved ordering of perishables: The value of stock-age information," *International Journal of Production Economics*, vol. 209, pp. 316–324, C 2019. https://ideas.repec.org/a/eee/proeco/v209y2019icp316-324.html.

[43] S. Nahmias, "A comparison of alternative approximations for ordering perishable inventory," *INFOR*, vol. 13, no. 2, pp. 175–184, 1975. https://doi.org/10.1080/03155986.1975.11731604.

[44] J. T. Blake *et al.*, "Optimizing the platelet supply chain in Nova Scotia," in *Proceedings of the 29th meeting of the EURO Working Group on Operational Research Applied to Health Services (ORAHS 2003)*, Prague, Czech Republic, 27 July – 1 August, 2003, pp. 47–66. http://orahs.di.unito.it/docs/2003-ORAHS-proceedings.pdf#page=47.

[45] S. Rajendran and S. Srinivas, "Hybrid ordering policies for platelet inventory management under demand uncertainty," *IISE Transactions on Healthcare Systems Engineering*, vol. 10, no. 2, pp. 113–126, 2020. https://doi.org/10.1080/24725579.2019.1686718.

[46] Q. Duan and T. W. Liao, "A new age-based replenishment policy for supply chain inventory optimization of highly perishable products," *International Journal of Production Economics*, vol. 145, no. 2, pp. 658–671, 2013. https://doi.org/10.1016/j.ijpe.2013.05.020.

[47] Q. Duan and T. W. Liao, "Optimization of blood supply chain with shortened shelf lives and ABO compatibility," *International Journal of Production Economics*, vol. 153, pp. 113–129, 2014. https://doi.org/10.1016/j.ijpe.2014.02.012.

[48] D. Dalalah *et al.*, "Platelets inventory management: A rolling horizon sim–opt approach for an age-differentiated demand," *Journal of Simulation*, vol. 13, no. 3, pp. 209–225, 2019. https://doi.org/10.1080/17477778.2018.1497461.

[49] R. Haijema *et al.*, "Blood platelet production: Optimization by dynamic programming and simulation," *Computers & Operations Research*, vol. 34, no. 3, pp. 760–779, 2007. https://doi.org/10.1016/j.cor.2005.03.023.

[50] N. V. van Dijk *et al.*, "Blood platelet production: A novel approach for practical optimization," *Transfusion*, vol. 49, no. 3, pp. 411–420, 2009. https://doi.org/10.1111/j.1537-2995.2008.01996.x.

[51] R. Haijema *et al.*, "Blood platelet production with breaks: Optimization by SDP and simulation," *International Journal of Production Economics*, vol. 121, no. 2, pp. 464–473, 2009. https://doi.org/10.1016/j.ijpe.2006.11.026.

[52] W. de Kort *et al.*, "Platelet pool inventory management: Theory meets practice," *Transfusion*, vol. 51, no. 11, pp. 2295–2303, 2011. https://doi.org/10.1111/j.1537-2995.2011.03190.x.

[53] S. Gunpinar and G. Centeno, "Stochastic integer programming models for reducing wastages and shortages of blood products at hospitals," *Computers and Operations Research*, vol. 54, pp. 129–141, 2015. https://doi.org/10.1016/j.cor.2014.08.017.

[54] S. Rajendran and A. R. Ravindran, "Platelet ordering policies at hospitals using stochastic integer programming model and heuristic approaches

to reduce wastage," *Computers & Industrial Engineering*, vol. 110, pp. 151–164, 2017. https://doi.org/10.1016/j.cie.2017.05.021.

[55] M. Meneses *et al.*, "Blood inventory management: Ordering policies for hospital blood banks under uncertainty," *International Transactions in Operational Research*, 2021. https://doi.org/10.1111/itor.12981.

[56] M. Meneses *et al.*, "Modelling the blood supply chain," *European Journal of Operational Research*, vol. 307, no. 2, pp. 499–518, 2023. https://doi.org/10.1016/j.ejor.2022.06.005.

[57] N. Yates *et al.*, "Approaches to assessing and minimizing blood wastage in the hospital and blood supply chain," *ISBT Science Series*, vol. 12, no. 1, pp. 91–98, 2017. https://doi.org/10.1111/voxs.12330.

[58] S. H. Stanger *et al.*, "Blood inventory management: Hospital best practice," *Transfusion Medicine Reviews*, vol. 26, no. 2, pp. 153–163, 2012. https://doi.org/10.1016/j.tmrv.2011.09.001.

[59] J.-D. Tissot and O. Garraud, "Ethics and blood donation: A marriage of convenience," *La Presse Médicale*, vol. 45, no. 7, e247–e252, 2016. https://doi.org/10.1016/j.lpm.2016.06.016.

[60] H. Shih *et al.*, "A multiple criteria decision-making model for minimizing platelet shortage and outdating in blood supply chains under demand uncertainty," *Healthcare Analytics*, vol. 3, p. 100 180, 2023. https://doi.org/10.1016/j.health.2023.100180.

[61] M. Motamedi *et al.*, "Demand forecasting for platelet usage: From univariate time series to multivariate models," *arXiv*, 2021. http://arxiv.org/abs/2101.02305.

[62] N. Li *et al.*, "A decision integration strategy for short-term demand forecasting and ordering for red blood cell components," *Operations Research for Health Care*, vol. 29, p. 100 290, 2021. https://doi.org/10.1016/j.orhc.2021.100290.

[63] G. Dumkreiger, "Data driven personalized management of hospital inventory of perishable and substitutable blood products," Ph.D. dissertation, Arizona State University, Phoenix, AZ, USA, 2020. https://keep.lib.asu.edu/items/158661.

[64] H. Ensafian *et al.*, "Raising quality and safety of platelet transfusion services in a patient-based integrated supply chain under uncertainty," *Computers & Chemical Engineering*, vol. 106, pp. 355–372, 2017. https://doi.org/10.1016/j.compchemeng.2017.06.015.

[65] N. Bakmohammadi *et al.*, "Optimal policy of ordering blood units in the hospital according to compatibility and priority transfers between blood groups under uncertainty conditions: A case study," *Transfusion and Apheresis Science*, vol. 62, no. 2, p. 103 529, 2023. https://doi.org/10.1016/j.transci.2022.103529.

[66] M. Najafi *et al.*, "Blood inventory management in hospitals: Considering supply and demand uncertainty and blood transshipment possibility," *Operations Research for Health Care*, vol. 15, pp. 43–56, 2017. https://doi.org/10.1016/j.orhc.2017.08.006.

[67] D. Dalalah and K. A. Alkhaledi, "Optimization of red blood cell inventory: A blood-type compatibility-preference and emergency model," *International Transactions in Operational Research*, vol. 30, no. 1, pp. 239–272, 2023. https://doi.org/10.1111/itor.12932.

[68] R. Chithraponnu and S. Umamaheswari, "Amelioration in cross-matching policy with subtypes of A for priority-based demand," *International Journal of Advanced and Applied Sciences*, vol. 10, no. 2, pp. 210–218, 2023. https://doi.org/10.21833/ijaas.2023.02.025.

[69] B. Hamdan and A. Diabat, "A two-stage multi-echelon stochastic blood supply chain problem," *Computers & Operations Research*, vol. 101, pp. 130–143, 2019. https://doi.org/10.1016/j.cor.2018.09.001.

[70] U. Abdulwahab and M. I. Wahab, "Approximate dynamic programming modeling for a typical blood platelet bank," *Computers and Industrial Engineering*, vol. 78, pp. 259–270, 2014. https://doi.org/10.1016/j.cie.2014.07.017.

[71] E. M. Hendrix *et al.*, "On computing optimal policies in perishable inventory control using value iteration," *Computational and Mathematical Methods*, vol. 1, no. 4, e1027, 2019. https://doi.org/10.1002/cmm4.1027.

[72] J. S. Rytilä and K. M. Spens, "Using simulation to increase efficiency in blood supply chains," *Management Research News*, vol. 29, no. 12, pp. 801–819, 2006. https://doi.org/10.1108/01409170610717826.

[73] R. Kopach *et al.*, "Tutorial on constructing a red blood cell inventory management system with two demand rates," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1051–1059, 2008. https://doi.org/10.1016/j.ejor.2006.01.051.

[74] K. Katsaliaki and S. C. Brailsford, "Using simulation to improve the blood supply chain," *Journal of the Operational Research Society*, vol. 58, no. 2, pp. 219–227, 2007. https://doi.org/10.1057/palgrave.jors.2602195.

[75] E. Yuzgec *et al.*, "A simulation model for blood supply chain systems," in *Proceedings of the 63rd Annual Conference and Expo of the Institute of Industrial Engineers*, San Juan, Puerto Rico, 18–22 May, 2013, pp. 1919–1927. https://search.proquest.com/docview/1471958846/abstract/86773457EE3E4E90PQ/1.

[76] J. T. Blake *et al.*, "The operational impact of introducing cold stored platelets," *Transfusion*, vol. 63, no. 12, pp. 2248–2255, 2023. https://doi.org/10.1111/trf.17565.

[77] J. T. Blake, "Determining the inventory impact of extended-shelf-life platelets with a network simulation model," *Transfusion*, vol. 57, no. 12, pp. 3001–3008, 2017. https://doi.org/10.1111/trf.14305.

[78] J. T. Blake *et al.*, "Déjà-vu all over again: Using simulation to evaluate the impact of shorter shelf life for red blood cells at Héma-Québec," *Transfusion*, vol. 53, no. 7, pp. 1544–1558, 2013. https://doi.org/10.1111/j.1537-2995.2012.03947.x.

[79] A. J. Katz *et al.*, "Simulation analysis of platelet production and inventory management," *Vox Sanguinis*, vol. 44, no. 1, pp. 31–36, 1983. https://doi.org/10.1111/j.1423-0410.1983.tb04100.x.

[80] A. Asllani *et al.*, "A simulation-based apheresis platelet inventory management model," *Transfusion*, vol. 54, no. 10, pp. 2730–2735, 2014. https://doi.org/10.1111/trf.12570.

[81] F. Baesler *et al.*, "Analysis of inventory strategies for blood components in a regional blood center using process simulation," *Transfusion*, vol. 54, no. 2, pp. 323–330, 2014. https://doi.org/10.1111/trf.12287.

[82] R. Kopach *et al.*, "Models for predicting critical blood product shortages," in *Proceedings of the 29th meeting of the EURO Working Group on Operational Research Applied to Health Services (ORAHS 2003)*, Prague, Czech Republic, 27 July – 1 August, 2003, pp. 77–90. http://orahs.di.unito.it/docs/2003-ORAHS-proceedings.pdf.

[83] A. Pereira, "Performance of time-series methods in forecasting the demand for red blood cell transfusion," *Transfusion*, vol. 44, no. 5, pp. 739–746, 2004. https://doi.org/10.1111/j.1537-2995.2004.03363.x.

[84] O. S. Filho *et al.*, "A decision-making tool for demand forecasting of blood components," in *Proceedings of the 14th IFAC Symposium on Information Control Problems in Manufacturing*, Bucharest, Romania, 23–25 May, 2012. http://dx.doi.org/10.3182/20120523-3-RO-2023.00201.

[85] O. S. S. Filho *et al.*, "Demand forecasting for blood components distribution of a blood supply chain," in *Proceedings of the 6th IFAC Conference on Management and Control of Production and Logistics*, Fortaleza, Brazil,

11–13 September, 2013, pp. 565–571. https://doi.org/10.3182/20130911-3-BR-3021.00092.

[86] S. M. Fortsch and E. A. Khapalova, "Reducing uncertainty in demand for blood," *Operations Research for Health Care*, vol. 9, pp. 16–28, 2016. https://doi.org/10.1016/j.orhc.2016.02.002.

[87] Y.-N. Liu *et al.*, "Time series analysis and prediction on clinical usage demand of red blood cells," *Journal of Shanghai Jiaotong University (Medical Science)*, vol. 40, no. 8, pp. 1113–1119, 2020. https://doi.org/10.3969/j.issn.1674-8115.2020.08.019.

[88] A. Wijayanayake and M. Dandunna, "An efficient model to improve the performance of platelet inventory of the blood banks," *Advances in Science, Technology and Engineering Systems Journal*, vol. 2, no. 3, pp. 839–844, 2017. https://www.astesj.com/v02/i03/p104/.

[89] B. Fanoodi *et al.*, "Reducing demand uncertainty in the platelet supply chain through artificial neural networks and ARIMA models," *Computers in Biology and Medicine*, vol. 113, p. 103 415, 2019. https://doi.org/10.1016/j.compbiomed.2019.103415.

[90] J. Quinn *et al.*, "The successful implementation of an automated institution-wide assessment of hemoglobin and ABO typing to dynamically estimate red blood cell inventory requirements," *Transfusion*, vol. 59, no. 7, pp. 2203–2206, 2019. https://doi.org/10.1111/trf.15272.

[91] N. Li *et al.*, "From demand forecasting to inventory ordering decisions for red blood cells through integrating machine learning, statistical modeling, and inventory optimization," *Transfusion*, vol. 62, no. 1, pp. 87–99, 2022. https://doi.org/10.1111/trf.16739.

[92] H. Abouee-Mehrizi *et al.*, "Data-driven platelet inventory management under uncertainty in the remaining shelf life of units," *Production and Operations Management*, vol. 31, no. 10, pp. 3914–3932, 2022. https://doi.org/10.1111/poms.13795.

[93] M. Mirjalili *et al.*, "A data-driven approach to determine daily platelet order quantities at hospitals," *Transfusion*, vol. 62, no. 10, pp. 2048–2056, 2022. https://doi.org/10.1111/trf.17080.

[94] M. Schilling *et al.*, "Reduction of platelet outdating and shortage by forecasting demand with statistical learning and deep neural networks: Modeling study," *JMIR Medical Informatics*, vol. 10, no. 2, e29978, 2022. https://doi.org/10.2196/29978.

[95] R. Khaldi *et al.*, "Artificial neural network based approach for blood demand forecasting: Fez transfusion blood center case study," in *Proceedings of the 2nd International Conference on Big Data, Cloud and Applications (BDCA '17)*, Tetouan, Morocco, 29–30 March, 2017, pp. 1–6. https://doi.org/10.1145/3090354.3090415.

[96] S. H. Stanger *et al.*, "What drives perishable inventory management performance? Lessons learnt from the UK blood supply chain," *Supply Chain Management: An International Journal*, vol. 17, no. 2, pp. 107–123, 2012. https://doi.org/10.1108/13598541211212861.

[97] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: The MIT Press, 2018, ISBN: 978-0-262-03924-6.

[98] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, 1959. http://people.csail.mit.edu/brooks/idocs/Samuel.pdf.

[99] G. Tesauro, "Temporal difference learning and TD-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995. https://dl.acm.org/doi/10.1145/203330.203343.

[100] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. https://doi.org/10.1038/nature16961.

[101] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. https://doi.org/10.1038/nature24270.

[102] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019. https://doi.org/10.1038/s41586-019-1724-z.

[103] A. Mirhoseini *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021. https://doi.org/10.1038/s41586-021-03544-w.

[104] N. Lazic *et al.*, "Data center cooling using model-predictive control," in *Advances in Neural Information Processing Systems: Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, Montréal, Canada, 3–8 December, 2018. https://proceedings.neurips.cc/paper/2018/hash/059fdcd96baeb75112f09fa1dcc740cc-Abstract.html.

[105] K. Arulkumaran *et al.*, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. https://doi.org/10.1109/MSP.2017.2743240.

[106] R. N. Boute *et al.*, "Deep reinforcement learning for inventory control: A roadmap," *European Journal of Operational Research*, vol. 298, no. 2, pp. 401–412, 2022. https://doi.org/10.1016/j.ejor.2021.07.016.

[107] M. J. Kochenderfer *et al.*, *Algorithms for Decision Making*. Cambridge, MA, USA: The MIT Press, 2022, ISBN: 978-0-262-04701-2.

[108] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods to Practical Problems of Chatbots, Robotics, Discrete Optimization, Web Automation, and More*, 2nd ed. Birmingham, UK: Packt, 2020, ISBN: 978-1-83882-699-4.

[109] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957.

[110] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, University of Oxford, Oxford, UK, 1989. https://www.cs.rhul.ac.uk/%5C~chrisw/new%5C_thesis.pdf.

[111] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed. Hoboken, NJ, USA: Wiley, 2011, ISBN: 978-0-470-60445-8.

[112] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," *arXiv*, 2016. http://arxiv.org/abs/1602.01783.

[113] J. Schulman *et al.*, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 6–11 July, 2015, pp. 1889–1897. http://proceedings.mlr.press/v37/schulman15.html.

[114] J. Schulman *et al.*, "Proximal policy optimization algorithms," *arXiv*, 2017. http://arxiv.org/abs/1707.06347.

[115] H. Bai *et al.*, "Evolutionary reinforcement learning: A survey," *arXiv*, 2023. http://arxiv.org/abs/2303.04150.

[116] E. Galván and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 6, pp. 476–493, 2021. https://doi.org/10.1109/TAI.2021.3067574.

[117] T. Salimans *et al.*, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv*, 2017. http://arxiv.org/abs/1703.03864.

[118] F. P. Such *et al.*, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv*, 2018. http://arxiv.org/abs/1712.06567.

[119] R. T. Lange, *Gymnax: A JAX-based reinforcement learning environment library*, 2022. http://github.com/RobertTLange/gymnax.

[120] C. D. Freeman *et al.*, "Brax – a differentiable physics engine for large scale rigid body simulation," *arXiv*, 2021. http://arxiv.org/abs/2106.13281.

[121] R. T. Lange, "Evosax: JAX-based evolution strategies," *arXiv*, 2022. http://arxiv.org/abs/2212.04180.

[122] Y. Tang *et al.*, "EvoJAX: Hardware-accelerated neuroevolution," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22)*, Boston, MA, USA, 9–13 July, 2022, pp. 308–311. https://doi.org/10.1145/3520304.3528770.

[123] A. Y. Majid *et al.*, "Deep reinforcement learning versus evolution strategies: A comparative survey," *arXiv*, 2021. http://arxiv.org/abs/2110.01411.

[124] P. Li *et al.*, "Bridging evolutionary algorithms and reinforcement learning: A comprehensive survey," *arXiv*, 2024. http://arxiv.org/abs/2401.11963.

[125] Q. Zhu *et al.*, "A survey on evolutionary reinforcement learning algorithms," *Neurocomputing*, vol. 556, p. 126 628, 2023. https://doi.org/10.1016/j.neucom.2023.126628.

[126] S. Levine *et al.*, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv*, 2020. https://arxiv.org/abs/2005.01643v1.

[127] S. V. Albrecht *et al.*, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. Cambridge, MA, USA: The MIT Press, 2024, ISBN: 978-0-262-04937-5. https://www.marl-book.com.

[128] T. Chu *et al.*, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2020. https://doi.org/10.1109/TITS.2019.2901791.

[129] J. Wang *et al.*, "Multi-agent reinforcement learning for active voltage control on power distribution networks," in *Advances in Neural Information Processing Systems: Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, Virtual Conference, 6–14 December, 2021. https://proceedings.neurips.cc/paper/2021/file/1a6727711b84fd1efbb87fc565199d13-Paper.pdf.

[130] J. K. Terry *et al.*, "PettingZoo: Gym for multi-agent reinforcement learning," *arXiv*, 2021. http://arxiv.org/abs/2009.14471.

[131] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the Tenth International Conference on International Conference on Machine Learning (ICML '93)*, Amherst, MA, USA, 27–29 July, 1993, pp. 330–337. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b3fc56876ad1cdf35ad4af13b991bbb24d219bd9.

[132] J. Foerster *et al.*, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems: Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS 2016)*, Barcelona, Spain, 5–11 December, 2016. https://proceedings.neurips.cc/paper_files/paper/2016/hash/c7635bfd99248a2cdef8249ef7bfbef4-Abstract.html.

[133] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *arXiv*, 2021. http://arxiv.org/abs/1908.03963.

[134] R. Lowe *et al.*, "Multi-agent actor-critic for mixed cooperative-competitive environments," *arXiv*, 2020. http://arxiv.org/abs/1706.02275.

[135] C. Yu *et al.*, "The surprising effectiveness of PPO in cooperative, multi-agent games," *arXiv*, 2022. http://arxiv.org/abs/2103.01955.

[136] P. Sunehag *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv*, 2017. http://arxiv.org/abs/1706.05296.

[137] T. Rashid *et al.*, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," *arXiv*, 2018. http://arxiv.org/abs/1803.11485.

[138] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: A survey," *Artificial Intelligence Review*, vol. 55, no. 2, pp. 895–943, 2022. https://doi.org/10.1007/s10462-021-09996-w.

[139] G. Brockman *et al.*, "OpenAI gym," *arXiv*, 2016. http://arxiv.org/abs/1606.01540.

[140] M. Towers *et al.*, *Gymnasium*, 2023. https://doi.org/10.5281/zenodo.8127026.

[141] J. Bradbury *et al.*, *JAX: Composable transformations of Python+NumPy programs*, 2018. http://github.com/google/jax.

[142] C. Yu *et al.*, "Reinforcement learning in healthcare: A survey," *arXiv*, 2020. http://arxiv.org/abs/1908.08796.

[143] M. Komorowski *et al.*, "The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care," *Nature Medicine*, vol. 24, no. 11, pp. 1716–1720, 2018. https://doi.org/10.1038/s41591-018-0213-5.

[144] Y. Wang *et al.*, "Predicting the need for blood transfusion in intensive care units with reinforcement learning," in *BCB '22: Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, Northbrook, IL, USA, 7–10 August, 2022, pp. 1–10. https://doi.org/10.1145/3535508.3545523.

[145] M. Allen and T. Monks, "Integrating deep reinforcement learning networks with health system simulations," *arXiv*, 2020. https://doi.org/10.5281/zenodo.3936515.

[146] B. Balaji *et al.*, "ORL: Reinforcement learning benchmarks for online stochastic optimization problems," *arXiv*, 2019. http://arxiv.org/abs/1911.10641.

[147] C. D. Hubbs *et al.*, "OR-gym: A reinforcement learning library for operations research problems," *arXiv*, 2020. http://arxiv.org/abs/2008.06319.

[148] B. Rolf *et al.*, "A review on reinforcement learning algorithms and applications in supply chain management," *International Journal of Production Research*, pp. 1–29, 2022. https://doi.org/10.1080/00207543.2022.2140221.

[149] Y. Yan *et al.*, "Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities," *Transportation Research Part E: Logistics and Transportation Review*, vol. 162, p. 102 712, 2022. https://doi.org/10.1016/j.tre.2022.102712.

[150] A. Kara and I. Dogan, "Reinforcement learning approaches for specifying ordering policies of perishable inventory systems," *Expert Systems with Applications*, vol. 91, pp. 150–158, 2018. https://doi.org/10.1016/j.eswa.2017.08.046.

[151] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," University of Cambridge, Cambridge, UK, 1994. http://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/rummery_tr166.pdf.

[152] R. Sun *et al.*, "Inventory cost control model for fresh product retailers based on DQN," in *Proceedings of the 2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, 9–12 December, 2019, pp. 5321–5325. https://doi.org/10.1109/BigData47090.2019.9006424.

[153] N. N. Sultana *et al.*, "Reinforcement learning for multi-product multi-node inventory management in supply chains," *arXiv*, 2020. http://arxiv.org/abs/2006.04037.

[154] H. Meisheri *et al.*, "Using reinforcement learning for a large variable-dimensional inventory management problem," in *Adaptive Learning Agents Workshop at the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)*, Virtual Conference, 9–13 May, 2020. https://ala2020.vub.ac.be/papers/ALA2020_paper_5.pdf.

[155] H. Meisheri *et al.*, "Scalable multi-product inventory control with lead time constraints using reinforcement learning," *Neural Computing and Applications*, 2021. https://doi.org/10.1007/s00521-021-06129-w.

[156] R. Wang *et al.*, "Solving a joint pricing and inventory control problem for perishables via deep reinforcement learning," *Complexity*, vol. 2021, 2021. https://doi.org/10.1155/2021/6643131.

[157] T. Abu Zwaida *et al.*, "Optimization of inventory management to prevent drug shortages in the hospital supply chain," *Applied Sciences*, vol. 11, no. 6, p. 2726, 2021. https://doi.org/10.3390/app11062726.

[158] A. Y. Ng *et al.*, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999)*, Bled, Slovenia, 27–30 June, 1999, pp. 278–287. http://people.eecs.berkeley.edu/~russell/papers/icml99-shaping.pdf.

[159] S. Jullien *et al.*, "A simulation environment and reinforcement learning method for waste reduction," *arXiv*, 2023. http://arxiv.org/abs/2205.15455.

[160] M. Selukar *et al.*, "Inventory control of multiple perishable goods using deep reinforcement learning for sustainable environment," *Sustainable Energy Technologies and Assessments*, vol. 52, p. 102 038, 2022. https://doi.org/10.1016/j.seta.2022.102038.

[161] D. G. Gioia *et al.*, "Inventory management of vertically differentiated perishable products with stock-out based substitution," in *Proceedings of the 10th IFAC Conference on Manufacturing Modelling, Management and*

Control (MIM 2022), Nantes, France, 22–24 June, 2022, pp. 2683–2688. https://doi.org/10.1016/j.ifacol.2022.10.115.

[162] E. Ahmadi *et al.*, "Intelligent inventory management approaches for perishable pharmaceutical products in a healthcare supply chain," *Computers & Operations Research*, vol. 147, p. 105 968, 2022. https://doi.org/10.1016/j.cor.2022.105968.

[163] K. Wang *et al.*, "Single-site perishable inventory management under uncertainties: A deep reinforcement learning approach," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–6, 2023. https://doi.org/10.1109/TKDE.2023.3250249.

[164] H. Abouee-Mehrizi *et al.*, "Platelet inventory management with approximate dynamic programming," *arXiv*, 2023. https://arxiv.org/abs/2307.09395.

[165] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, San Francisco, CA, USA, 13–17 August, 2016, pp. 785–794. https://doi.org/10.1145/2939672.2939785.

[166] M. Mirjalili, "Data-driven modelling and control of hospital blood inventory," Ph.D. dissertation, University of Toronto, Toronto, Canada, 2022. https://tspace.library.utoronto.ca/bitstream/1807/124976/1/Mirjalili_Mahdi_202211_PhD_thesis.pdf.

[167] N. Mohamadi *et al.*, "An application of deep reinforcement learning and vendor-managed inventory in perishable supply chain management," *Engineering Applications of Artificial Intelligence*, vol. 127, p. 107 403, 2024. https://doi.org/10.1016/j.engappai.2023.107403.

[168] Q. Li *et al.*, "Blood component preparation-inventory problem with stochastic demand and supply," *International Transactions in Operational Research*, 2021. https://doi.org/10.1111/itor.13073.

[169] M. Altaf *et al.*, "Deep reinforcement learning model for blood bank vehicle routing multi-objective optimization," *Computers, Materials & Continua*, vol. 70, no. 2, pp. 3955–3967, 2022. https://doi.org/10.32604/cmc.2022.019448.

[170] J. Gijsbrechts *et al.*, "Can deep reinforcement learning improve inventory management? Performance on dual sourcing, lost sales and multi-echelon problems," *Manufacturing & Service Operations Management*, vol. 24, no. 3, pp. 1349–1368, 2022. https://doi.org/10.1287/msom.2021.1064.

[171] H. D. Perez *et al.*, "Algorithmic approaches to inventory management optimization," *Processes*, vol. 9, no. 1, p. 102, 2021. https://doi.org/10.3390/pr9010102.

[172] J. M. Farrington *et al.*, "Deep reinforcement learning for managing platelets in a hospital blood bank," *Blood*, 65th ASH Annual Meeting Abstracts, vol. 142, p. 2311, 2023. https://doi.org/10.1182/blood-2023-178306.

[173] K. G. Pauls-Worm *et al.*, "Inventory control for a perishable product with non-stationary demand and service level constraints," Wageningen University, 2013. http://www.optimization-online.org/DB_FILE/2013/08/4010.pdf.

[174] M. L. Bynum *et al.*, *Pyomo – Optimization Modeling in Python*, 3rd ed. Cham, Switzerland: Springer International, 2021, ISBN: 978-3-030-68927-8.

[175] W. E. Hart *et al.*, "Pyomo: Modeling and solving mathematical programs in Python," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011. https://doi.org/10.1007/s12532-011-0026-8.

[176] B. Kneuven *et al.*, "A parallel hub-and-spoke system for large-scale scenario-based optimization under uncertainty," *Optimization Online*, 2020. http://www.optimization-online.org/DB%5C_HTML/2020/11/8088.html.

[177] Gurobi Optimization, LLC, *Gurobi optimizer reference manual*, 2022. https://www.gurobi.com.

[178] H. Van Hasselt *et al.*, "Deep reinforcement learning with double Q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, Phoenix, AZ, USA, 12–17 February, 2016, pp. 2094–2100. https://ojs.aaai.org/index.php/AAAI/article/view/10295.

[179] J. Weng *et al.*, "Tianshou: A highly modularized deep reinforcement learning library," *arXiv*, 2021. http://arxiv.org/abs/2107.14171.

[180] J. Schulman *et al.*, "High-dimensional continuous control using generalized advantage estimation," *arXiv*, 2018. http://arxiv.org/abs/1506.02438.

[181] F. Pardo *et al.*, "Time limits in reinforcement learning," *arXiv*, 2017. https://arxiv.org/abs/1712.00378v3.

[182] S. Y. Su and R. A. Deininger, "Generalization of White's method of successive approximations to periodic Markovian decision processes," *Operations Research*, vol. 20, no. 2, pp. 318–326, 1972. https://doi.org/10.1287/opre.20.2.318.

[183] Joblib Development Team, *Joblib: Running Python functions as pipeline jobs*, 2020. https://joblib.readthedocs.io/.

[184] J. Farrington *et al.*, "Going faster to see further: GPU-accelerated value iteration and simulation for perishable inventory control using JAX," *arXiv*, 2023. http://arxiv.org/abs/2303.10672.

[185] J. Nickolls *et al.*, "Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for?" *Queue*, vol. 6, no. 2, pp. 40–53, 2008. https://doi.org/10.1145/1365490.1365500.

[186] A. P. Jóhannsson, "GPU-based Markov decision process solver," M.Sc. disseratation, Reykjavík University, Reykjavík, Iceland, 2009. https://en.ru.is/media/skjol-td/MSThesis_ArsaellThorJohannsson.pdf.

[187] E. M. Aldrich *et al.*, "Tapping the supercomputer under your desk: Solving dynamic equilibrium models with graphics processors," *Journal of Economic Dynamics and Control*, vol. 35, no. 3, pp. 386–393, 2011. https://doi.org/10.1016/j.jedc.2010.10.001.

[188] A. Aamer *et al.*, "Data analytics in the supply chain management: Review of machine learning applications in demand forecasting," *Operations and Supply Chain Management: An International Journal*, vol. 14, no. 1, pp. 1–13, 2020. https://doi.org/10.31387/oscm0440281.

[189] V. Duarte *et al.*, "Benchmarking machine-learning software and hardware for quantitative economics," *Journal of Economic Dynamics and Control*, vol. 111, p. 103 796, 2020. https://doi.org/10.1016/j.jedc.2019.103796.

[190] R. Kirkby, "A toolkit for value function iteration," *Computational Economics*, vol. 49, no. 1, pp. 1–15, 2017. https://doi.org/10.1007/s10614-015-9544-1.

[191] R. Kirkby, "Quantitative macroeconomics: Lessons learned from fourteen replications," *Computational Economics*, 2022. https://doi.org/10.1007/s10614-022-10234-w.

[192] P. Chen and L. Lu, "Markov decision process parallel value iteration algorithm on GPU," in *Proceedings of 2013 International Conference on Information Science and Computer Applications*, Suwon, South Korea, 24–26 June, 2013, pp. 299–304. https://doi.org/10.2991/isca-13.2013.51.

[193] T. Inamoto *et al.*, "An implementation of dynamic programming for many-core computers," in *Proceedings of the SICE Annual Conference 2011*, Tokyo, Japan, 13–18 September, 2011, pp. 961–966. https://ieeexplore.ieee.org/abstract/document/6060648.

[194] S. Ruiz and B. Hernández, "A parallel solver for Markov decision process in crowd simulations," in *Proceedings of the Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*, Cuernavaca, Mexico, 25–31 October, 2015, pp. 107–116. https://doi.org/10.1109/MICAI.2015.23.

[195] D.-A. Constantinescu *et al.*, "Performance evaluation of decision making under uncertainty for low power heterogeneous platforms," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 119–133, 2020. https://doi.org/10.1016/j.jpdc.2019.11.009.

[196] G. Ortega *et al.*, "A CUDA approach to compute perishable inventory control policies using value iteration," *The Journal of Supercomputing*, vol. 75, no. 3, pp. 1580–1593, 2019. https://doi.org/10.1007/s11227-018-2692-z.

[197] P. C. Yianni *et al.*, "Accelerating petri-net simulations using NVIDIA graphics processing units," *European Journal of Operational Research*, vol. 265, no. 1, pp. 361–371, 2018. https://doi.org/10.1016/j.ejor.2017.06.068.

[198] Z. Bäumelt *et al.*, "A novel approach for nurse rerostering based on a parallel algorithm," *European Journal of Operational Research*, vol. 251, no. 2, pp. 624–639, 2016. https://doi.org/10.1016/j.ejor.2015.11.022.

[199] M. A. Boschetti *et al.*, "Route relaxations on GPU for vehicle routing problems," *European Journal of Operational Research*, vol. 258, no. 2, pp. 456–466, 2017. https://doi.org/10.1016/j.ejor.2016.09.050.

[200] P. Hijma *et al.*, "Optimization techniques for GPU programming," *ACM Computing Surveys*, 2022. https://doi.org/10.1145/3570638.

[201] W. Jeon *et al.*, "Chapter six - deep learning with GPUs," in *Advances in Computers*, ser. Hardware Accelerator Systems for Artificial Intelligence and Machine Learning, S. Kim and G. C. Deka, Eds., vol. 122, Cambridge,

MA, USA: Elsevier, 2021, pp. 167–215, ISBN: 978-0-12-823123-4. https://doi.org/10.1016/bs.adcom.2020.11.003.

[202] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI '16)*, Savannah, GA, USA, 2–4 November, 2016, pp. 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[203] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems: Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, Canada, 8–14 December, 2019. https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[204] T. J. Sargent and J. Stachurski. "Dynamic programming on the GPU via JAX - QuantEcon notes." (2022), https://notes.quantecon.org/submission/622ed4daf57192000f918c61/comments (accessed 2023-01-24).

[205] K. Perumalla and M. Alam, "Design considerations for GPU-based mixed integer programming on parallel computing platforms," in *Proceedings of the 50th International Conference on Parallel Processing Workshop (ICPP Workshops '21)*, Lemont, IL, USA, 9–12 August, 2021. https://doi.org/10.1145/3458744.3473366.

[206] S. Amaran *et al.*, "Simulation optimization: A review of algorithms and applications," *Annals of Operations Research*, vol. 240, no. 1, pp. 351–380, 2016. https://doi.org/10.1007/s10479-015-2019-x.

[207] M. C. Fu *et al.*, "Simulation optimization: A panel on the state of the art in research and practice," in *Proceedings of the Winter Simulation Conference 2014*, Savannah, GA, USA, 7–10 December, 2014, pp. 3696–3706. https://doi.org/10.1109/WSC.2014.7020198.

[208] G. Srimool *et al.*, "Speeding up a large logistics optimization problems using GPU technology," in *Proceedings of the 8th Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI) Association of Thailand - Conference 2011*, Khon Kaen, Thailand, 17–19 May, 2011, pp. 450–454. https://doi.org/10.1109/ECTICON.2011.5947872.

[209] M. C. Lau and R. Srinivasan, "A hybrid CPU-graphics processing unit (GPU) approach for computationally efficient simulation-optimization," *Computers & Chemical Engineering*, vol. 87, pp. 49–62, 2016. https://doi.org/10.1016/j.compchemeng.2016.01.001.

[210] W.-m. W. Hwu *et al.*, *Programming massively parallel processors: A hands-on approach*, 4th ed. Cambridge, MA, USA: Morgan Kaufmann, 2023, ISBN: 978-0-323-91231-0.

[211] "How to think in JAX," JAX documentation. (2024), https://jax.readthedocs.io/en/latest/notebooks/thinking_in_jax.html (accessed 2024-12-14).

[212] T. Akiba *et al.*, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*, Anchorage, AK, USA, 4–9 August, 2019, pp. 2623–2631. https://doi.org/10.1145/3292500.3330701.

[213] K. Deb *et al.*, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002. https://doi.org/10.1109/4235.996017.

[214] "NVIDIA GeForce RTX 3060 graphics card," Ebuyer. (2024), https://www.ebuyer.com/store/Components/cat/Graphics-Cards-Nvidia/subcat/GeForce-RTX-3060?q=nvidia+3060 (accessed 2024-06-10).

[215] "A100 PCIe 40 GB vs GeForce RTX 3060," TechnicalCity. (2024), https://technical.city/en/video/GeForce-RTX-3060-vs-A100-PCIe-40-GB (accessed 2024-06-10).

[216] C. Li and I. E. Grossmann, "A review of stochastic programming methods for optimization of process systems under uncertainty," *Frontiers in Chemical Engineering*, vol. 2, p. 34, 2021. https://doi.org/10.3389/fceng.2020.622241.

[217] S. Karayev and C. Frye. "Cloud GPUs," Full Stack Deep Learning. (2023), https://fullstackdeeplearning.com/cloud-gpus/ (accessed 2024-06-28).

[218] A. Peri, "A hardware approach to value function iteration," *Journal of Economic Dynamics and Control*, vol. 114, p. 103 894, 2020. https://doi.org/10.1016/j.jedc.2020.103894.

[219] B. Liu and F. Papier, "Remanufacturing of multi-component systems with product substitution," *European Journal of Operational Research*, vol. 301, no. 3, pp. 896–911, 2022. https://doi.org/10.1016/j.ejor.2021.11.029.

[220] X. Liu *et al.*, "Replacement and inventory control for a multi-customer product service system with decreasing replacement costs," *European Journal of Operational Research*, vol. 273, no. 2, pp. 561–574, 2019. https://doi.org/10.1016/j.ejor.2018.08.029.

[221] M. A. Voelkel *et al.*, "An aggregation-based approximate dynamic programming approach for the periodic review model with random yield," *European Journal of Operational Research*, vol. 281, no. 2, pp. 286–298, 2020. https://doi.org/10.1016/j.ejor.2019.08.035.

[222] M. Heydar *et al.*, "A stochastic model for the patient-bed assignment problem with random arrivals and departures," *Annals of Operations Research*, vol. 315, no. 2, pp. 813–845, 2022. https://doi.org/10.1007/s10479-021-03982-9.

[223] E. J. Lodree *et al.*, "Staff assignment policies for a mass casualty event queuing network," *Annals of Operations Research*, vol. 283, no. 1, pp. 411–442, 2019. https://doi.org/10.1007/s10479-017-2635-8.

[224] A. Torrado and A. Barbosa-Póvoa, "Towards an optimized and sustainable blood supply chain network under uncertainty: A literature review," *Cleaner Logistics and Supply Chain*, vol. 3, p. 100 028, 2022. https://doi.org/10.1016/j.clscn.2022.100028.

[225] R. Haijema, "Optimal ordering, issuance and disposal policies for inventory management of perishable products," *International Journal of Production Economics*, vol. 157, pp. 158–169, 2014. https://doi.org/10.1016/j.ijpe.2014.06.014.

[226] Blood Stocks Management Scheme, "Inventory management best practice guide," 2022. https://nhsbtdbe.blob.core.windows.net/umbraco-assets-corp/25788/bsms-inventory-management-best-practice-guide-jan-2022.pdf.

[227] M. Bakker *et al.*, "Review of inventory systems with deterioration since 2001," *European Journal of Operational Research*, vol. 221, no. 2, pp. 275–284, 2012. https://doi.org/10.1016/j.ejor.2012.03.004.

[228] L. Janssen *et al.*, "Literature review of deteriorating inventory models by key topics from 2012 to 2015," *International Journal of Production Economics*, vol. 182, pp. 86–112, 2016. https://doi.org/10.1016/j.ijpe.2016.08.019.

[229] I. Z. Karaesmen *et al.*, "Managing perishable and aging inventories: Review and future research directions," in *Planning Production and Inventories in the Extended Enterprise: A State of the Art Handbook, Volume 1*, ser. International Series in Operations Research & Management Science, K. G. Kempf *et al.*, Eds., New York, NY, USA: Springer, 2011, pp. 393–436, ISBN: 978-1-4419-6485-4. https://doi.org/10.1007/978-1-4419-6485-4_15.

[230] G. Mirabelli and V. Solina, "Optimization strategies for the integrated management of perishable supply chains: A literature review," *Journal of Industrial Engineering and Management*, vol. 15, no. 1, pp. 58–91, 2022. https://doi.org/10.3926/jiem.3603.

[231] W. P. Pierskalla and C. D. Roach, "Optimal issuing policies for perishable inventory," *Management Science*, vol. 18, no. 11, pp. 603–614, 1972. https://www.jstor.org/stable/2629154.

[232] I. Civelek *et al.*, "Blood platelet inventory management with protection levels," *European Journal of Operational Research*, vol. 243, no. 3, pp. 826–838, 2015. https://doi.org/10.1016/j.ejor.2015.01.023.

[233] B. Abbasi and S. Z. Hosseinifard, "On the issuing policies for perishable items such as red blood cells and platelets in blood service," *Decision Sciences*, vol. 45, no. 5, pp. 995–1020, 2014. https://doi.org/https://doi.org/10.1111/deci.12092.

[234] M. P. Atkinson *et al.*, "A novel allocation strategy for blood transfusions: Investigating the tradeoff between the age and availability of transfused blood," *Transfusion*, vol. 52, no. 1, pp. 108–117, 2012. https://doi.org/10.1111/j.1537-2995.2011.03239.x.

[235] J. B. Jennings, "An analysis of hospital blood bank whole blood inventory control policies," *Transfusion*, vol. 8, no. 6, pp. 335–342, 1968. https://doi.org/10.1111/j.1537-2995.1968.tb02433.x.

[236] A. Pereira, "Blood inventory management in the type and screen era," *Vox Sanguinis*, vol. 89, no. 4, pp. 245–250, 2005. https://doi.org/10.1111/j.1423-0410.2005.00700.x.

[237] J. Staves *et al.*, "Electronic remote blood issue: A combination of remote blood issue with a system for end-to-end electronic control of transfusion to provide a "total solution" for a safe and timely hospital blood transfusion service," *Transfusion*, vol. 48, no. 3, pp. 415–424, 2008. https://doi.org/10.1111/j.1537-2995.2007.01545.x.

[238] D. P. Yahnke *et al.*, "Analysis and optimization of a regional blood bank distribution process," *Transfusion*, vol. 12, no. 2, pp. 111–118, 1972. https://doi.org/10.1111/j.1537-2995.1972.tb05896.x.

[239] D. P. Yahnke *et al.*, "Analysis and optimization of a regional blood bank distribution process: II. Derivation and use of a method for evaluating hospital management procedures," *Transfusion*, vol. 13, no. 3, pp. 156–169, 1973. https://doi.org/10.1111/j.1537-2995.1973.tb05466.x.

[240] A. F. H. Britten and D. G. Geurtze, "Weekly rotation of blood inventory—a system for supplying small hospitals," *Transfusion*, vol. 19, no. 6, pp. 738–741, 1979. https://doi.org/10.1046/j.1537-2995.1979.19680104100.x.

[241] K. Chen *et al.*, "Managing hospital platelet inventory with mid-cycle expedited replenishments and returns," *Production and Operations Management*, vol. 31, no. 5, pp. 2015–2037, 2022. https://doi.org/10.1111/poms.13662.

[242] M. Ahmadimanesh *et al.*, "Designing an optimal model of blood logistics management with the possibility of return in the three-level blood transfusion network," *Research Square*, 2022. https://www.researchsquare.com/article/rs-1187827/v1.

[243] P. Ambilkar *et al.*, "Product returns management: A comprehensive review and future research agenda," *International Journal of Production Research*, pp. 1–25, 2021. https://doi.org/10.1080/00207543.2021.1933645.

[244] H. Abdulla *et al.*, "Taking stock of consumer returns: A review and classification of the literature," *Journal of Operations Management*, vol. 65, no. 6, pp. 560–605, 2019. https://doi.org/10.1002/joom.1047.

[245] A. Canda *et al.*, "Modeling and forecasting product returns: An industry case study," in *Proceedings of the 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore,

6–9 December, 2015, pp. 871–875. https://doi.org/10.1109/IEEM.2015.7385772.

[246] C. A. Tsiliyannis, "Markov chain modeling and forecasting of product returns in remanufacturing based on stock mean-age," *European Journal of Operational Research*, vol. 271, no. 2, pp. 474–489, 2018. https://doi.org/10.1016/j.ejor.2018.05.026.

[247] M. C. Chou *et al.*, "Policies for inventory models with product returns forecast from past demands and past sales," *Annals of Operations Research*, vol. 288, no. 1, pp. 137–180, 2020. https://doi.org/10.1007/s10479-020-03545-4.

[248] H. Cui *et al.*, "Predicting product return volume using machine learning methods," *European Journal of Operational Research*, vol. 281, no. 3, pp. 612–627, 2020. https://doi.org/10.1016/j.ejor.2019.05.046.

[249] I. Joshi and J. Morley, "Artificial intelligence: How to get it right," *NHS Transformation Directorate*, 2019. https://transform.england.nhs.uk/media/documents/NHSX_AI_report.pdf.

[250] Y. Zhu *et al.*, "A local algorithm for product return prediction in e-commerce," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 13–19 July, 2018, pp. 3718–3724. https://doi.org/10.24963/ijcai.2018/517.

[251] X. Li *et al.*, "A trust-aware random walk model for return propensity estimation and consumer anomaly scoring in online shopping," *Science China Information Sciences*, vol. 62, no. 5, p. 52 101, 2019. https://doi.org/10.1007/s11432-018-9511-1.

[252] Y. Fu *et al.*, "Fused latent models for assessing product return propensity in online commerce," *Decision Support Systems*, vol. 91, pp. 77–88, 2016. https://doi.org/10.1016/j.dss.2016.08.002.

[253] E. Brodheim and G. Prastacos, "Demand, usage and issuing of blood at hospital blood banks," Operations Research Laboratory, The New York Blood Center, 1980.

[254] N. H. Shah *et al.*, "Making machine learning models clinically useful," *JAMA*, vol. 322, no. 14, pp. 1351–1352, 2019. https://doi.org/10.1001/jama.2019.10306.

[255] Q. Zhou *et al.*, "Clinical impact and quality of randomized controlled trials involving interventions evaluating artificial intelligence prediction tools: A systematic review," *npj Digital Medicine*, vol. 4, no. 1, pp. 1–12, 2021. https://doi.org/10.1038/s41746-021-00524-2.

[256] R. C. Li *et al.*, "Developing a delivery science for artificial intelligence in healthcare," *npj Digital Medicine*, vol. 3, no. 1, pp. 1–3, 2020. https://doi.org/10.1038/s41746-020-00318-y.

[257] V. V. Mišić *et al.*, "A simulation-based evaluation of machine learning models for clinical decision support: Application and analysis using hospital readmission," *npj Digital Medicine*, vol. 4, no. 1, pp. 1–11, 2021. https://doi.org/10.1038/s41746-021-00468-7.

[258] M. Wornow *et al.*, "APLUS: A Python library for usefulness simulations of machine learning models in healthcare," *Journal of Biomedical Informatics*, vol. 139, p. 104 319, 2023. https://doi.org/10.1016/j.jbi.2023.104319.

[259] J. A. Taylor *et al.*, "The road to hell is paved with good intentions: The experience of applying for national data for linkage and suggestions for improvement," *BMJ Open*, vol. 11, no. 8, e047575, 2021. https://doi.org/10.1136/bmjopen-2020-047575.

[260] N. Q. Viet *et al.*, "The value of information in supply chain decisions: A review of the literature and research agenda," *Computers & Industrial Engineering*, vol. 120, pp. 68–82, 2018. https://doi.org/10.1016/j.cie.2018.04.034.

[261] R. Fildes and B. Kingsman, "Incorporating demand uncertainty and forecast error in supply chain planning models," *Journal of the Operational Research Society*, vol. 62, no. 3, pp. 483–500, 2011. https://doi.org/10.1057/jors.2010.40.

[262] K. Altendorfer *et al.*, "Effects of forecast errors on optimal utilisation in aggregate production planning with stochastic customer demand," *International Journal of Production Research*, vol. 54, no. 12, pp. 3718–3735, 2016. https://doi.org/10.1080/00207543.2016.1162918.

[263] N. Sanders and G. Graman, "Quantifying costs of forecast errors: A case study of the warehouse environment," *Omega*, vol. 37, no. 1, pp. 116–125, 2009. https://doi.org/10.1016/j.omega.2006.10.004.

[264] A. Kron *et al.*, "Multicenter observational study evaluating the impact of platelet transport bags on product wastage," *Transfusion*, vol. 61, no. 5, pp. 1383–1388, 2021. https://doi.org/10.1111/trf.16303.

[265] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, pp. 84–90, 2022. https://doi.org/10.1016/j.inffus.2021.11.011.

[266] H. Ma *et al.*, "On use of partial area under the ROC curve for evaluation of diagnostic performance," *Statistics in Medicine*, vol. 32, no. 20, pp. 3449–3458, 2013. https://doi.org/10.1002/sim.5777.

[267] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems: Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS '17)*, Long Beach, CA, USA, 4–9 December, 2017, pp. 4765–4774. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.

[268] E. Fenwick *et al.*, "Value of information analysis for research decisions—an introduction: Report 1 of the ISPOR Value of Information Analysis

Emerging Good Practices Task Force," *Value in Health*, vol. 23, no. 2, pp. 139–150, 2020. https://doi.org/10.1016/j.jval.2020.01.001.

[269] A. Feizi *et al.*, "Vertical patient streaming in emergency departments," *SSRN*, 2023. https://papers.ssrn.com/abstract=4465161.

[270] M. G. Seneviratne *et al.*, "Bridging the implementation gap of machine learning in healthcare," *BMJ Innovations*, vol. 6, no. 2, 2020. https://doi.org/10.1136/bmjinnov-2019-000359.

[271] C. Aubron *et al.*, "Platelet storage duration and its clinical and transfusion outcomes: A systematic review," *Critical Care*, vol. 22, no. 1, p. 185, 2018. https://doi.org/10.1186/s13054-018-2114-x.

[272] L. J. Estcourt *et al.*, "Guidelines for the use of platelet transfusions," *British Journal of Haematology*, vol. 176, no. 3, pp. 365–394, 2017. https://doi.org/10.1111/bjh.14423.

[273] National Institute for Health and Care Excellence, "Blood transfusion (NICE guideline 24)," 2015. https://www.nice.org.uk/guidance/ng24/chapter/Recommendations.

[274] M. Sekhar *et al.*, "Effective implementation of a patient blood management programme for platelets," *Transfusion Medicine*, vol. 26, no. 6, pp. 422–431, 2016. https://doi.org/10.1111/tme.12331.

[275] A. Abbaspour *et al.*, "A simple empirical model for blood platelet production and inventory management under uncertainty," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 2, pp. 1783–1799, 2021. https://doi.org/10.1007/s12652-020-02254-x.

[276] D. J. Triulzi *et al.*, "The impact of platelet transfusion characteristics on posttransfusion platelet increments and clinical bleeding in patients with hypoproliferative thrombocytopenia," *Blood*, vol. 119, no. 23, pp. 5553–5562, 2012. https://doi.org/10.1182/blood-2011-11-393165.

[277] Z.-J. Ma *et al.*, "An emergency blood allocation approach considering blood group compatibility in disaster relief operations," *International Journal of Disaster Risk Science*, vol. 10, no. 1, pp. 74–88, 2019. https://doi.org/10.1007/s13753-018-0212-7.

[278] V. Silva Magalhães *et al.*, "Simulation- optimisation approach to support management of blood components inventory," *Journal of Simulation*, pp. 1–16, 2023. https://doi.org/10.1080/17477778.2023.2293861.

[279] P. Govender and A. E. Ezugwu, "Boosting symbiotic organism search algorithm with ecosystem service for dynamic blood allocation in blood banking system," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 34, no. 2, pp. 261–293, 2022. https://doi.org/10.1080/0952813X.2021.1871665.

[280] P. Govender and A. E. Ezugwu, "A symbiotic organisms search algorithm for optimal allocation of blood products," *IEEE Access*, vol. 7, pp. 2567–2588, 2019. https://doi.org/10.1109/ACCESS.2018.2886408.

[281] B. Deniz *et al.*, "Managing perishables with substitution: Inventory issuance and replenishment heuristics," *Manufacturing & Service Operations Management*, vol. 12, no. 2, pp. 319–329, 2010. https://doi.org/10.1287/msom.1090.0276.

[282] W. Qiao *et al.*, "Distributed dynamic pricing of multiple perishable products using multi-agent reinforcement learning," *Expert Systems with Applications*, vol. 237, p. 121 252, 2024. https://doi.org/10.1016/j.eswa.2023.121252.

[283] R. Leluc *et al.*, "MARLIM: Multi-agent reinforcement learning for inventory management," *arXiv*, 2023. http://arxiv.org/abs/2308.01649.

[284] Y. Ding *et al.*, "Multi-agent reinforcement learning with shared resources for inventory management," *arXiv*, 2022. http://arxiv.org/abs/2212.07684.

[285] X. Yang *et al.*, "A versatile multi-agent reinforcement learning benchmark for inventory management," *arXiv*, 2023. http://arxiv.org/abs/2306.07542.

[286] X. Liu *et al.*, "Multi-agent deep reinforcement learning for multi-echelon inventory management," *SSRN*, 2022. https://papers.ssrn.com/abstract=4262186.

[287] M. Khirwar *et al.*, "Cooperative multi-agent reinforcement learning for inventory management," *arXiv*, 2023. http://arxiv.org/abs/2304.08769.

[288] S. Koyamada *et al.*, "Pgx: Hardware-accelerated parallel game simulators for reinforcement learning," in *Advances in Neural Information Processing Systems: Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS Datasets and Benchmarks Track 2023)*, New Orleans, LA, USA, 10–16 December, 2023, pp. 45 716–45 743. https://proceedings.neurips.cc/paper_files/paper/2023/hash/8f153093758af93861a74a1305dfdc18-Abstract-Datasets_and_Benchmarks.html.

[289] S. Lange *et al.*, "Batch reinforcement learning," in *Reinforcement Learning: State-of-the-Art*, ser. Adaptation, Learning, and Optimization, M. Wiering and M. van Otterlo, Eds., Berlin, Germany: Springer, 2012, pp. 45–73, ISBN: 978-3-642-27645-3. https://doi.org/10.1007/978-3-642-27645-3_2.

[290] H. van Hasselt and M. A. Wiering, "Using continuous action spaces to solve discrete problems," in *Proceedings of the 2009 International Joint Conference on Neural Networks*, Atlanta, GA, USA, 14–19 June, 2009, pp. 1149–1156. https://doi.org/10.1109/IJCNN.2009.5178745.

[291] Y. Chandak *et al.*, "Learning action representations for reinforcement learning," in *Proceedings of the 36th International Conference on Machine*

Learning, Long Beach, CA, USA, 10–15 June, 2019, pp. 941–950. http://proceedings.mlr.press/v97/chandak19a.html.

[292] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," *arXiv*, 2015. http://arxiv.org/abs/1512.07679.

[293] N. Vanvuchelen and R. N. Boute, "The use of continuous action representations to scale deep reinforcement learning: An application to inventory control," *SSRN*, 2022. https://www.ssrn.com/abstract=4253600.

[294] T. Yu Suen *et al.*, "A two-stage stochastic model for a multi-objective blood platelet supply chain network design problem incorporating frozen platelets," *Computers & Industrial Engineering*, vol. 185, p. 109 651, 2023. https://doi.org/10.1016/j.cie.2023.109651.

[295] D. M. Roijers *et al.*, "A survey of multi-objective sequential decision-making," *Journal of Artificial Intelligence Research*, vol. 48, pp. 67–113, 2013. https://doi.org/10.1613/jair.3987.

[296] E. Chołodowicz and P. Orłowski, "Control of perishable inventory system with uncertain perishability process using neural networks and robust multicriteria optimization," *Bulletin of the Polish Academy of Sciences Technical Sciences*, pp. 141 182–141 182, 2022. https://doi.org/10.24425/bpasts.2022.141182.

[297] M. Khouja and S. Goyal, "A review of the joint replenishment problem literature: 1989–2005," *European Journal of Operational Research*, vol. 186, no. 1, pp. 1–16, 2008. https://doi.org/10.1016/j.ejor.2007.03.007.

[298] L. Peng *et al.*, "A review of the joint replenishment problem from 2006 to 2022," *Management System Engineering*, vol. 1, no. 1, p. 9, 2022. https://doi.org/10.1007/s44176-022-00010-3.

[299] G. Nagpal *et al.*, "The first half-century of decision modelling for substitutable products: A literature review and bibliographic analysis," *Operations and Supply Chain Management: An International Journal*, vol. 14, no. 3, pp. 261–276, 2021. https://doi.org/10.31387/oscm0460301.

[300] H. Shin *et al.*, "A classification of the literature on the planning of substitutable products," *European Journal of Operational Research*, vol. 246, no. 3, pp. 686–699, 2015. https://doi.org/10.1016/j.ejor.2015.04.013.

[301] N. Vanvuchelen *et al.*, "Use of proximal policy optimization for the joint replenishment problem," *Computers in Industry*, vol. 119, p. 103 239, 2020. https://doi.org/10.1016/j.compind.2020.103239.

[302] S. Prestwich *et al.*, "A neuroevolutionary approach to stochastic inventory control in multi-echelon systems," *International Journal of Production Research*, vol. 50, no. 8, pp. 2150–2160, 2012. https://doi.org/10.1080/00207543.2011.574503.

[303] I. Jackson, "Neuroevolutionary approach to metamodeling of production-inventory systems with lost-sales and Markovian demand," in *Reliability and Statistics in Transportation and Communication: Selected Papers from the 19th International Conference on Reliability and Statistics in Transportation and Communication (RelStat '19)*, Riga, Latvia, 16–19 October, 2020, pp. 90–99. https://doi.org/10.1007/978-3-030-44610-9_10.

[304] J. L. Balintfy, "On a basic class of multi-item inventory problems," *Management Science*, vol. 10, no. 2, pp. 287–297, 1964. https://doi.org/10.1287/mnsc.10.2.287.

[305] J. Heek *et al.*, *Flax: A neural network library and ecosystem for JAX*, 2023. http://github.com/google/flax.

[306] C. Lu *et al.*, "Discovered policy optimisation," in *Advances in Neural Information Processing Systems: Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, New Orleans, LA, USA, November 28–December 9, 2022, pp. 16 455–16 468. https://proceedings.neurips.cc/paper_files/paper/2022/hash/688c7a82e31653e7c256c6c29fd3b438-Abstract-Conference.html.

[307] L. Biewald, *Experiment tracking with weights and biases*, 2020. https://www.wandb.com/.

[308] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature - PPSN VIII: 8th International Conference*, Birmingham, UK, 18–22 September, 2004, pp. 832–842. https://doi.org/10.1007/978-3-540-30217-9_84.

[309] C. F. Hayes *et al.*, "A practical guide to multi-objective reinforcement learning and planning," *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 1, p. 26, 2022. https://doi.org/10.1007/s10458-022-09552-y.

[310] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in Python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020. https://doi.org/10.1109/ACCESS.2020.2990567.

[311] Health Service Executive, "Platelet transfusion in clinical practice: Professional guidance document," 2014. https://www.lenus.ie/bitstream/handle/10147/313664/plateletguidance.pdf?sequence=1%26isAllowed=y.

[312] R. de Kock *et al.*, "Mava: A research library for distributed multi-agent reinforcement learning in JAX," *arXiv*, 2023. http://arxiv.org/abs/2107.01460.

[313] C. Bonnet *et al.*, "Jumanji: A diverse suite of scalable reinforcement learning environments in JAX," *arXiv*, 2024. http://arxiv.org/abs/2306.09884.

[314] M. Lechner *et al.*, "Gigastep - one billion steps per second multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems: Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS Datasets and Benchmarks Track 2023)*, New Orleans, LA, USA, 10–16 December, 2023. https://proceedings.neurips.cc/paper_files/paper/2023/hash/00ba06ba5c324efdfb068865ca44cf0b-Abstract-Datasets_and_Benchmarks.html.

[315] F. Felten *et al.*, "A toolkit for reliable benchmarking and research in multi-objective reinforcement learning," in *Advances in Neural Information Processing Systems: Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS Datasets and Benchmarks Track 2023)*, New Orleans, LA, USA, 10–16 December, 2023. https://proceedings.neurips.cc/paper_files/paper/2023/hash/4aa8891583f07ae200ba07843954caeb-Abstract-Datasets_and_Benchmarks.html.

[316] X. Sun *et al.*, "RBC inventory-management system based on XGBoost model," *Indian Journal of Hematology & Blood Transfusion: An Official Journal of Indian Society of Hematology and Blood Transfusion*, vol. 37, no. 1, pp. 126–133, 2021. https://doi.org/10.1007/s12288-020-01333-5.

[317] M. Engelke *et al.*, "Predicting individual patient platelet demand in a large tertiary care hospital using machine learning," *Transfusion Medicine and Hemotherapy*, pp. 1–9, 2023. https://doi.org/10.1159/000528428.

[318] R. Bhandawat *et al.*, "Cooperative blood inventory ledger (CoBIL): A decentralized decision-making framework for improving blood product management," *Computers & Industrial Engineering*, vol. 172, p. 108 571, 2022. https://doi.org/10.1016/j.cie.2022.108571.

[319] A. Rutherford *et al.*, "JaxMARL: Multi-agent RL environments in JAX," *arXiv*, 2023. http://arxiv.org/abs/2311.10090.

[320] M. S. Karafin *et al.*, "Demographic and epidemiologic characterization of transfusion recipients from four US regions: Evidence from the REDS-III recipient database," *Transfusion*, vol. 57, no. 12, pp. 2903–2913, 2017. https://doi.org/10.1111/trf.14370.

[321] A. Pape *et al.*, "Clinical evidence of blood transfusion effectiveness," *Blood Transfusion*, vol. 7, no. 4, pp. 250–258, 2009. https://doi.org/10.2450/2008.0072-08.

[322] Joint United Kingdom (UK) Blood Transfusion and Tissue Transplantation Services Professional Advisory Committee. "JPAC - transfusion guidelines 25.10: Protocol 000005 – blood component fate information." (2022), https://www.transfusionguidelines.org/red-book/chapter-25-standards-for-electronic-data-interchange-within-the-uk-blood-transfusion-services/25-10-protocol-000005-blood-component-fate-information (accessed 2022-03-23).

[323] R. Sun and G. Zhao, "Analyses about efficiency of reinforcement learning to supply chain ordering management," in *Proceedings of the IEEE 10th International Conference on Industrial Informatics*, Beijing, China, 13 September, 2012, pp. 124–127. https://doi.org/10.1109/INDIN.2012.6301163.

[324] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. https://www.jmlr.org/papers/v12/pedregosa11a.html.

[325] O. Yadan, *Hydra - a framework for elegantly configuring complex applications*, 2019. https://github.com/facebookresearch/hydra.

[326] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arXiv*, 2015. https://arxiv.org/abs/1412.6980.

[327] D. Harar. "Machine learning with ordered labels," Stack Exchange. (2023), https://stats.stackexchange.com/q/611604 (accessed 2024-03-13).