Biopharmaceutical Scheduling Using a Flexible Genetic Algorithm Approach

A thesis submitted to University College London (UCL)

for the degree of

Doctor of Philosophy (PhD)

by

Karolis Jankauskas, MEng (Hons)

The Advanced Centre for Biochemical Engineering

Department of Biochemical Engineering

UCL

London, United Kingdom

September, 2018

Declaration

I, Karolis Jankauskas, confirm that this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

The goal of biopharmaceutical capacity planning and scheduling is to identify an optimal production schedule (solution) that would satisfy multiple financial and operational objectives. It is a complex combinatorial optimisation problem characterised by features such as multi-product portfolios, variable process durations and yields, long product lead and approval times, and uncertain market forecasts. The bulk of research in the area of biopharmaceutical capacity planning and scheduling has focused on Mixed Integer Linear Programming (MILP) formulations. Heuristic optimisation methods such as Genetic Algorithms (GAs) have received very little attention even though they are reportedly more flexible, easier to implement and, in certain cases, have the potential of outperforming mathematical programming models. Therefore, this thesis addresses this knowledge gap by describing the development of a flexible GA-based Decision Support Tool (DST) for single- and multi-objective biopharmaceutical capacity planning and scheduling under deterministic and uncertain product demand.

This thesis makes four broad contributions. Firstly, a GA is designed for solving biopharmaceutical capacity planning and scheduling problems using a discrete-time representation. The effectiveness of the algorithm is demonstrated on two industrial case studies and compared with discrete-time MILP models from the literature. A rolling time horizon strategy is applied to improve solution quality and the performance of the GA. A Particle Swam Optimisation (PSO) algorithm is utilised as a meta-optimiser to automatically tune the parameters of the GA. Secondly, a novel variable-length chromosome structure and an entirely new continuous-time scheduling heuristic are developed for more realistic and efficient medium-term scheduling of biopharmaceutical manufacture. The variable-length chromosome enables the GA to generate production schedules from a single gene. The novel variable-length GA with

an embedded continuous-time scheduling heuristic is shown to outperform related discrete- and continuous-time MILP models on two literature-based examples. Thirdly, a multi-objective component is added to the variable-length GA and the continuous-time scheduling heuristic is extended with additional constraints and features, including rolling product sequence-dependent changeovers and lengthy product quality control and assurance (QC/QA) checks. A real-life industrial case study is used to demonstrate the functionality and benefits of the multi-objective optimisation. The multi-objective variable-length GA is used to optimise both the total production throughput and monthly product inventory levels of a multi-product biopharmaceutical facility. Finally, the multi-objective variable-length GA is combined with a Graphics Processing Unit (GPU)-accelerated Monte Carlo simulation for biopharmaceutical capacity planning and scheduling under uncertain product demand. The merits of the approach are highlighted by comparing the production schedules generated when the uncertainty in demand is ignored and when it is accounted for by characterising it with a probability distribution. In the final sections of this thesis an example of a commercial application of this work is presented.

Impact Statement

Capacity planning and scheduling plays a very important role in the biopharmaceutical industry. Improper planning decisions can lead to high costs and loss of profit. Ransohoff (2004) estimated that for a typical 500 kg/year mAb facility 50% underutilisation could cost $2-3 M/month whereas 50% under capacity would likely result in a monthly profit loss of $40-50M. When Amgen launched Enbrel, an arthritis drug, in 1998, the demand for it was higher than anticipated. Malik et al. (2002) estimated that the lack of manufacturing capacity for Enbrel cost the company more than $200M in lost revenue in 2001. Hence, this work describes a flexible, GA-based DST developed in collaboration with industry experts for multi-objective capacity planning and scheduling of biopharmaceutical manufacture bringing several benefits to both academia and industry. A special focus is placed on deployability of the tool which is something that is very rarely discussed in production planning and scheduling literature.

This thesis addresses the research gap in heuristic-based biopharmaceutical capacity planning and scheduling optimisation by describing a framework based on a novel variable-length GA embedded with a continuous-time scheduling heuristic. The framework has been applied to a variety of literature-based and real life industrial case studies The results were presented during the 28th European Conference on Operational Research (EURO), the 253rd American Chemical Society (ACS) National Meeting (Jankauskas, Long, et al., 2017), and a keynote lecture at the 27th European Symposium on Computer Aided Process Engineering (ESCAPE) (Jankauskas, Papageorgiou, et al., 2017).

Even though there have been multiple biopharmaceutical capacity planning and scheduling optimisation models reported in the literature, companies still rely mostly

on manual, spreadsheet-based scheduling methods mostly due to a steep learning curve and a high level of expertise associated with mathematical programming models (Mustafa et al., 2006; Widmer et al., 2008). Using more accessible research principles, the GA-based DST developed during this PhD helps the biopharmaceutical companies understand the impact of constraints and uncertainties on key operational and risk metrics and allows to make better scheduling decisions faster. A commercial application of the framework was demonstrated to the industrial sponsor in Indianapolis, USA, during August 1-3, 2018.

Acknowledgements

I would like to express my gratitude to Prof. Suzanne Farid, my academic supervisor, for her guidance and support. I would also like to thank my industrial supervisor Dr. Graham McCartney for his consistent positive attitude and feedback.

I am eternally grateful to my family and friends. I would not be where I am today without them.

Financial support provided by the Engineering and Physical Sciences Research Council (EPSRC) and Eli Lilly & Company is gratefully acknowledged.

Contents

1. Background

The environment of biopharmaceutical manufacture is vastly dynamic and complex. Its business landscape is defined by expensive, long-term research and development (R&D) process and high risks of clinical failure. Biopharmaceutical products are also immensely sophisticated requiring substantial investment of capital, human resources, and technological expertise to produce them. Depending on the scale, biopharmaceutical facilities cost approximately $40-500M and can take several years to build. Moreover, they are costly to operate, with long process durations, relatively low yields, and the need for highly skilled experts to run them (Otto et al., 2014).

Nevertheless, due to success and general efficiency of biopharmaceutical products in treating complex health diseases, the industry has experienced constant and enormous growth since its inception in 1982 (Siganporia, 2016). For example, the number of biotech patent applications every year has been growing at 25% annually since 1995, the global revenues of biopharmaceuticals were reported to be over $100B in 2010 (Walsh, 2010) and over $150B in 2014 (Otto et al., 2014). The overall annual industry growth has been estimated at 14-15% (Langer & Rader, 2017).

Managing manufacturing facility assets for these growing and dynamic biopharmaceutical portfolios requires careful capacity planning. Essential to achieving this are agile capacity planning algorithms that can reconcile multiple conflicting objectives and deal with inherent uncertainty. Hence, this thesis presents the development of a flexible planning and scheduling tool for optimising the manufacturing capacity in an existing multi-product facility using a stochastic multi-objective GA.

This chapter will discuss the risks and costs of biopharmaceutical drug development (**Section 1.1**), describe what a typical biopharmaceutical manufacturing process looks like (**Section 1.2**), overview the concepts of planning and scheduling (**Section 1.3**) and the most common optimisation approaches (**Sections 1.4** and **1.5**), and review related work carried out on capacity planning and scheduling in pharmaceutical and biopharmaceutical industry (**Section 1.6)**. Finally, the aims and the overall structure of this thesis are discussed in **Section 1.7**.

1.1. Biopharmaceutical Industry Overview

For a biopharmaceutical product to make it into the market, it must first pass a series pre-clinical tests and clinical trials (Figure 1.1). An investigational new drug application can be filed with the Food and Drug Administration (FDA) for drugs that pass the pre-clinical testing. If the application is successful, phase I of clinical trials can begin. During this phase the drug product is administered to a small number of healthy volunteers to study its safety and pharmacology, i.e. absorption, metabolic effect, excretion, and toxicity. The next phase, i.e. phase II, of clinical trials examines the effectiveness of the compound on subjects with the target disease. Phase III is the final stage before a new drug application (NDA) can be submitted to the FDA and the drug can be sold to the market (Friedman et al., 2015).

The characteristics of biopharmaceutical drug development are vastly different compared to most of the chemical engineering industry: high costs of development, high risks of failure during drug discovery, clinical trials that take years to complete, time sensitive compounds, limited product shelf-life, stringent current good manufacturing practices (cGMP), unique process validation requirements, and intense competition from generics after the end of a 20-year patent (Laínez et al., 2012; Majozi et al., 2015).

Figure 1.1. Traditional drug development value chain (Source: Sabatier et al., 2010).

The likelihood of a new biopharmaceutical drug product gaining approval for marketing and the rate of approval for new products have been getting lower over the years. According to Kaitin and DiMasi (2010), only one in six new drugs that entered clinical trials in the United States during 1993-1998 and the 1999-2004 sub-periods were successfully approved for marketing. Shanley (2014) reported that only 12% of the candidate drugs get approved for use. According to a more recent study by DiMasi et al. (2016), the likelihood that a drug that enters clinical testing is also about 12%. Figure 1.2 highlights the risks and costs associated with the development process.

	Pre-clinical	Phase I	Phase II	Phase III	FDA	Market	
Average number of projects required to achieve 1 market success	12.4	8.6	4.6	1.6	1.1	1	
			Cost of portfolio development				**Total**
Total costs ($ million)	87	281	185	162	33		748
Cost as % total expenditure	11.6	37.5	24.8	21.7	4.4		
Cost of pre-clinical and clinical trials ($ million)	55	265	162	121	3		605
Cost of manufacturing ($ million)	14	16	11	14	0		54
Cost of process development ($ million)	19	0	13	27	30		88
Non-clinical cost as % total stage cost	37.2	5.6	12.8	25.4	90		19.1

Figure 1.2. The costs of biopharmaceutical drug development pathway (Source: Nie, 2015).

The cost of development of a single drug entering human trials between 1989 and 2002 was estimated to be in excess of $800M (DiMasi et al., 2003). Based on the data collected at University College London (UCL) (Farid, 2007), the costs of investment for antibody manufacturing facilities with total site capabilities in the range of 20,000–200,000L were reported to be $7,130–$17,000/m^2 and $1,765–$4,220/L.

The development costs have been rising continuously for years. For example, in an analysis by Paul et al. (2010), the cost of a new molecular entity was reported to be in excess of $1B whereas DiMasi et al. (2016) revised this figure to $2.8B.

1.2. Biopharmaceutical Manufacture

A biopharmaceutical refers to any pharmaceutical drug product for therapeutic or *in vivo* diagnostic purposes produced from biological sources such as microbial, e.g. *E. coli* or *P. pastoris*, mammalian, e. g. Chinese hamster ovary (CHO) cells, and plant cell cultures. The unique and complex macromolecular structure of biopharmaceuticals distinguishes them from conventional chemical products. Biopharmaceutical production platforms can be operated in a batch, fed-batch, or perfusion mode. Fed-batch mode, where nutrients are periodically added to the bioreactor over the course of cell fermentation, is preferable to a regular batch mode mostly due to higher yields. In perfusion mode, the product is harvested throughout the culture rather than at the end of it. Perfusion mode is favoured when the product is unstable and the purification of it is time sensitive. Most biopharmaceutical production platforms are fed-batch-based (Fike, 2009; Jiang et al., 2012).

Regardless of the mode of operation, a biopharmaceutical production process is typically divided into two broad manufacturing stages: upstream processing (*USP*) and downstream processing (*DSP*) (Figure 1.3). In *USP*, cells are derived from culture banks and nurtured in progressively larger bioreactors to express the biopharmaceutical product. In *DSP*, the raw product is extracted from the cells and purified using a series of processing steps such as centrifugation, microfiltration, chromatography, ultrafiltration, and viral clearance. Additionally, every step in both *USP* and *DSP* comprises several ancillary unit operations for cleaning, sterilisation,

preparation of intermediate materials such as culture media and buffer solutions, and product quality control and assurance (QC/QA).

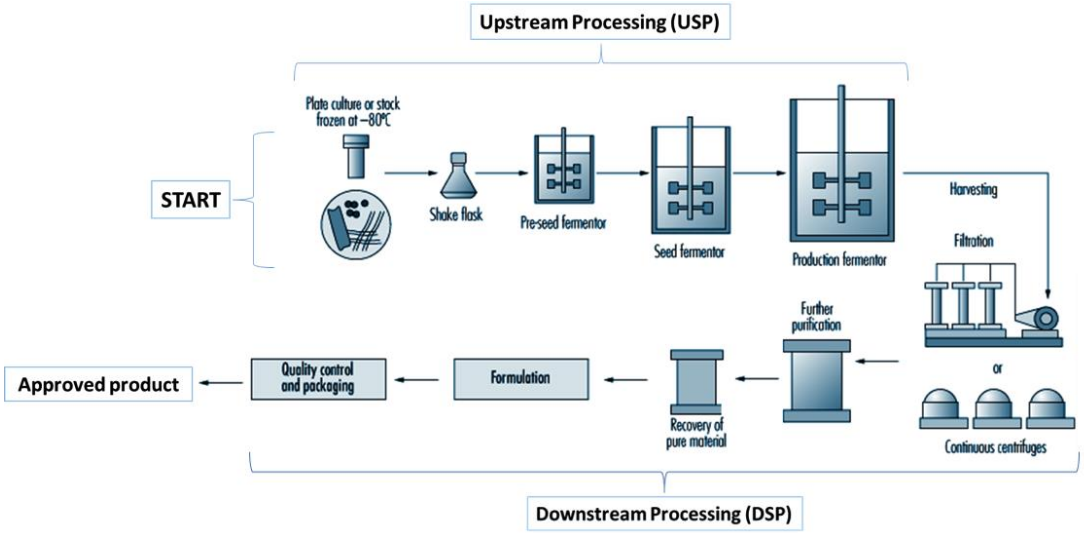


Figure 1.3. Typical biopharmaceutical production process flowsheet. Adapted from Tait (1998).

Biopharmaceutical production is performed in a series of manufacturing campaigns. Determining the duration of each campaign is a difficult challenge that requires careful consideration of the tradeoff between two different kinds of risks and costs. For example, due to costs, risks of cross-contamination, and considerable amount of time associated with setup and cleaning during a campaign changeover, some companies prefer long campaigns with uninterrupted series of batches (Lakhdar et al., 2005). However, in order to meet uncertain demand, it can be safer for the biopharmaceutical facilities to operate multiple smaller scale bioreactors that are scheduled appropriately (Simaria et al., 2012).

1.3. Planning and Scheduling Overview

Planning and scheduling can have a substantial impact on production performance and cost-effectiveness of manufacturing operations. A good production schedule can result in significant savings through better capacity utilisation. For example, Ransohoff

(2004) reported that a typical mammalian cell-culture facility could increase its annual revenue by $380M with a 25% increase in plant utilisation. Planning and scheduling appear in a wide range of industries, including Pulp and Paper, Metals, Oil and Gas, Chemicals, Food and Beverages, Pharmaceuticals, Transportations, Service, and Military, because of a substantial impact on production performance and the cost-effectiveness of manufacturing operations.

Production planning refers to the preparation of manufacture: specifying what components are needed to manufacture a product, determining optimal sourcing of raw materials, clarifying what processes and unit operations are necessary to transform those raw materials into a final product, and defining the distribution network. Production scheduling, on the other hand, involves decisions regarding optimal allocation, sequencing, and timing of resources or capacity across a broad number of competing tasks to satisfy one or more objectives and constraints. Figure 1.4 illustrates three major decisions in scheduling of batch processes: batching (lot-sizing), assignment, and sequencing.

Despite the variety of business environments, the type and goals of scheduling problems are usually defined by four major factors: market environment, interaction with other planning functions, production environment, and specific processing characteristics (Harjunkoski et al., 2014).

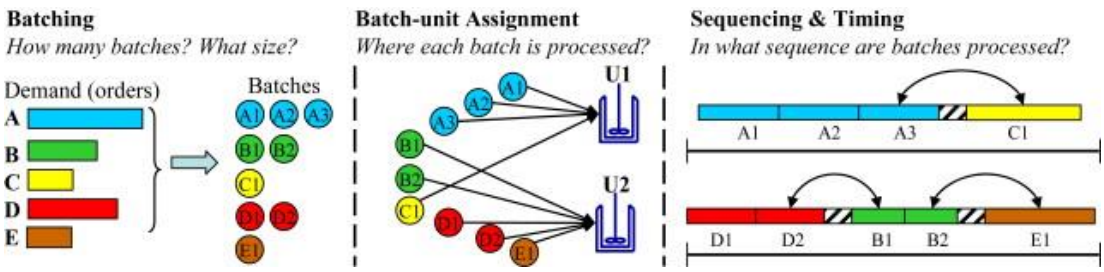


Figure 1.4. Major decisions in batch process scheduling (Source: Harjunkoski et al., 2014).

The variability and volume of product demand dictates the regularity and frequency of production campaigns. For example, the typical strategy for scheduling the production of high-volume products is to generate a cyclic schedule in order to maintain the product inventory at certain strategic levels, i.e. make-to-stock. In contrast, products with infrequent demand orders are usually scheduled as needed, i.e. make-to-order (Pochet & Wolsey, 2006). Manufacturing capacity also plays an important role in determining planning and scheduling objectives. If a production facility has manufacturing capacity to spare, then the goal of planning and scheduling is usually to minimise the total cost or earliness. On the other hand, when the product demand is higher than the manufacturing capacity, the goal is to maximise the total profit or throughput and/or minimise backlogs. It is common for companies to have multiple make-to-stock and make-to-order products manufactured in the same facility.

Production scheduling also depends on the outcomes from other supply chain management functions such as procurement, distribution, and demand planning. For example, the availability of raw materials and the estimated quantities and due dates of product demand orders are key inputs to scheduling. Other factors influencing production scheduling decisions include the type of manufacturing process, e.g. batch or continuous, and the type of facility, e.g. single-stage or multi-stage. The more intricate the facility design is, the more complex a scheduling problem will be. The scheduling of a facility is also affected by the specific processing characteristics such as utilities, setup and changeover requirements, and storage and resource constraints.

Traditionally, planning and scheduling has been carried out manually by specialists using spreadsheets, industry experience, and rule-based scheduling, e.g. first come first serve (FCFS), schedule the job with the shortest processing time (SPT), earliest due date first (EDD) (Panwalkar & Iskander, 1977; Haupt, 1989). However, due to

increasing production volumes, greater number of products, different manufacturing scenarios, and uncertain markets, it is difficult to ensure a cost-effective production plan without any optimisation support. Scheduling problems are NP-hard (Bitran & Yanasse, 1982) which is to say that finding the optimal solution to scheduling optimisation problems, especially the large-scale ones, is very difficult. Therefore, the general problem of planning and scheduling has received a considerable amount of attention in the literature. Ever since the introduction of the first basic lot sizing problem in 1958 (Wagner & Whitin, 1958), a number of papers have been written across different scientific communities. Due to the variety of problems, a number of approaches have been developed, including expert systems, decomposition-based methods, and optimisation algorithms based on mathematical programming or heuristics. Useful reviews of the development of planning and scheduling optimisation approaches over the last 10-20 years can be found in Shah (1998), Pinto and Grossmann (1998), Kallrath (2002), Floudas and Lin (2004), Méndez et al. (2006), Widmer et al. (2008), Majozi et al. (2015), and Copil et al. (2017).

The subsequent sections will review mathematical programming and heuristic (mostly GA-based) optimisation approaches and the related work.

1.4. Mathematical Programming

The vast majority of the capacity planning and scheduling optimisation models are based on a branch of mathematical programming – mixed-integer linear programming (MILP) which is a variation of linear programming (LP) for combinatorial optimisation problems. *Programming* in this context refers to planning and logistics instead of computer programming. LP is a technique for the optimisation of a linear objective function subject to linear equality and inequality constraints. Despite the assumptions of linearity, both LP and MILP have been shown to be effective at solving problems in

a variety of domains including not only capacity planning (e.g. Lazaros G Papageorgiou et al., 2001) and scheduling (e.g. Lorigeon et al., 2002) but also transportation (e.g. Abara, 1989).

The problem of solving a system of linear inequalities can be dated back to as far as Jean-Babtiste Joseph Fourier who published a method for solving such a system in 1827 (Sierksma, 2001). However, the first LP formulation as well as a method for solving it are attributed to Leonid Kantorovich, a Soviet economist, who used it to reduce the costs of the Red Army in 1939 (Schrijver, 1998). Around the same time, Tjalling Koopmans formulated classical economic problems as LP problems. As a result, Kantorovich and Koopmans shared the Nobel prize in economics in 1975 (Sierksma, 2001).

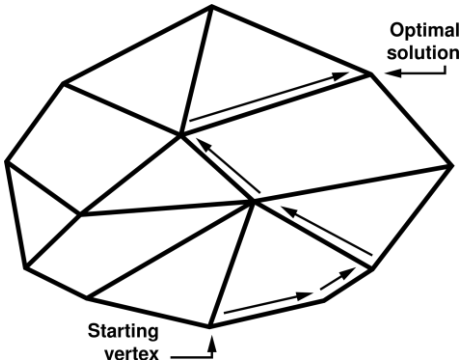


Figure 1.5. A polytope defined as a feasible region by the constraints applied to the objective function. The simplicial cones are the corners (vertices) of a polytope.

The early LP methods were improved by George B. Dantzig who independently developed a general LP formulation and invented the simplex method for solving LP problems (Dantzig, 1951). The name of the algorithm comes from the idea that it operates on simplicial cones which become simplices with additional constraints (Stone & Tovey, 1991). In Figure 1.5, the simplex method explores the feasible region by moving from corner to corner (or vertex) until the optimal solution is found. Only the corners of the polytope need to be explored since no other point on the line will

ever be optimal. The performance of the simplex method is highly dependent on the number of constraints, i.e. the number of constraints is proportional to the number of corner points (vertices) in the polytope that need to be explored. Alternative methods were developed for tackling more complex problems with a large number of constraints.

Karmarkar (1984) developed an interior point method for solving large-scale LP problems. The name “interior” comes from the fact that the best solution is reached by traversing the interior of the feasible region, i.e. the polytope. This method, also known as Karmarkar’s algorithm, was proven to run in polynomial time and enabled solutions of LP problems that were beyond the capabilities of the simplex method. Nevertheless, Paparrizos et al. (2003) reported that on small and medium-sized LP problems, the simplex algorithm actually performs better.

Many practical problems require discrete variables, e.g. explicit decisions are usually modelled using binary variables. One of the critical limitations of the simplex method is that it is only applicable to continuous variables. LP problems with discrete variables, i.e. MILP problems, could be solved by enumerating the solutions for every possible integer value. However, the brute-force method is only feasible when the scale of the problem is relatively small. Large MILP problems are typically solved using techniques that are based on divide-and-conquer algorithmic approaches such as branch and bound (B&B) algorithm. LP relaxations are first solved using the B&B algorithm to bound the objective function, and then branches are created by adding constraints that eliminate non-integer values.

The mathematical programming models for capacity planning and scheduling optimisation can be classified according to the following four main aspects: time representation, material balances, event representation, and objective function

(Méndez et al., 2006). Time representation is the first and most important issue. The optimisation methods typically utilise discrete- or continuous-time representation. Discrete-time representation is based on the discretisation of planning horizon into a number of time periods with predefined durations. The start and end times of tasks can only take place at the boundaries of these periods. Since the time points are known, the overall model structure becomes simpler and easier to solve. However, the computational efficiency of the model and its size depend on the number of time periods defined as a function of the input data and desired accuracy of the solution. Furthermore, the reduction of the domain of timing decisions can often yield sub-optimal or even infeasible solutions. Nevertheless, optimisation models using discrete-time representation have been widely used in the literature.

Continuous-time representation has been adopted to overcome the aforementioned limitations and build data-independent optimisation models. Using this representation, timing decisions are represented as a set of continuous variables defining the timings of events. While the variable time handling allows for more flexible solutions and results in models with fewer variables, more complicated constraints with big-M (large number associated with the artificial variables) terms are required to model resource and inventory limitations which negatively impacts the complexity of the model and the capabilities of the overall method.

Mathematical planning and scheduling models can be further classified based on how batches and their sizes are managed. There are two broad categories: models which assume that the number of batches of each size is known in advance and monolithic models that simultaneously address the optimal number and size of batches, allocation and sequencing of resources, and the timing of processing operations. The first category uses an approximate two-stage approach, i.e. batching and batch scheduling, to address larger practical problems. The second category of models

typically employ the state-task network (STN) (Kondili et al., 1993) or the resource-task network (RTN) (Pantelides, 1994) to represent the problem. The STN is a directed graph that consists of state nodes, task nodes that represent processing operations, and arcs that indicate the flow of materials between the states and tasks. The STN-based optimisation approaches assume that processing events produce and consume states, e.g. raw materials, intermediate and final products. The RTN-based formulation assumes that processing and storage tasks consume and release resource at their start and end times. STN- or RTN-based formulations are able to handle arbitrary network processes but are mostly limited to a small number of processing tasks and short planning horizons.

Méndez et al. (2006) defined five different types of event representations. Figure 1.6 illustrates the same schedule of fives batches (a, b, c, d, e) allocated to two units (J1 and J2) generated using the alternative event representations.

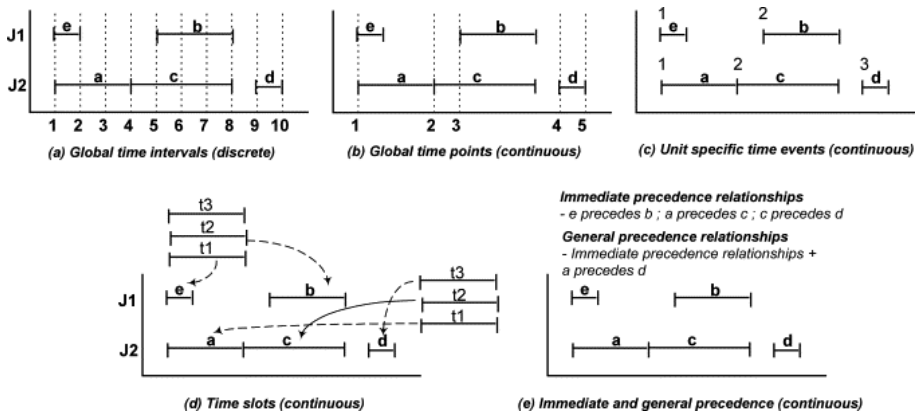


Figure 1.6. Different time representations used in scheduling problems (Source: Méndez et al., 2006).

For discrete-time representations, the definition of global time periods is the only option for general network (processed materials can be mixed and split) and sequential (no mixing of the processed materials, the same batch is assumed to be processed in different stages) processes. A common fixed time grid valid for all shared

resources is predefined and events are scheduled to start and finish exactly at the grid points. The main advantage of a fixed time grid is its simplicity. Continuous-time formulation involves extensive alternative event representations that focus on different types of processes. For sequential processes, time slots and batch precedence-based approaches can be used, whereas in the case of general network processes, global time points and unit-specific time events are employed. The global time period representation corresponds to a generalisation of global time periods where the timing of time periods is modelled as a new variable. Based on the detailed comparison of various continuous-time models for short-term scheduling of batch plant performed by Shaik et al. (2006), the unit-specific event-based models always require fewer event points and yield favourable computational performance compared to both slot-based and global event-based models due to heterogeneous locations of event points used.

Different criteria of solution quality can be used for scheduling problems. The six most commonly used are: makespan, earliness, tardiness, profit, inventory, and cost. The choice of the objective function has a direct effect on the computational performance of the optimisation model. A review of relevant literature on mathematical programming-based biopharmaceutical capacity planning and scheduling is provided in section 1.6.

1.5. Heuristics

While mathematical programming is often the optimisation method of choice, heuristic approaches have also been investigated to address certain limitations of these methods. These include dealing with non-linearities, uncertain parameters, and generating feasible solutions for industrial-sized problems in a reasonable amount of time. The word “heuristic” originates from Ancient Greek word for “find” or “discover”.

Generally, heuristic refers to any approximate problem-solving method that unlike mathematical programming does not guarantee optimality. Instead of being dedicated to the solution of a particular problem, heuristics are typically designed with the aim of being flexible enough to handle as many different combinatorial problems as possible. Despite the lack of guarantee of solution optimality, heuristics provide a number of advantages compared to classical optimisation, including flexibility, lack of assumptions about the problem, and ease-of-implementation in most cases. There have been many papers reporting success stories of applying heuristics to a wide-range of NP-hard problems. Heuristic problem-solving methods can be broadly classified into local search methods and population-based search techniques (Widmer et al., 2008).

In local search methods, the solution space is explored at each step by moving from one solution to a more optimal one in its neighbourhood. According to Hertz and Widmer (2003), local search can be thought of as a traversal of a directed graph $G = (S, A)$ where S is a set of solutions to a particular problem and A is a set of arcs (s, s') if and only if s' is in the neighbourhood of s. The neighbourhood of a solution s is defined as the set of solutions that can be obtained from s by making simple modifications to it. Some of the better-known local search techniques are simulated annealing (Kirkpatrick et al., 1983) and tabu search (Glover, 1986).

Population-based search techniques differ from the previous methods in that they keep a sample of solutions rather than a single candidate solution. The solution population is usually randomly generated and then iteratively tweaked and assessed in the direction of better solutions according to a certain set of rules. Most such methods are based on concepts borrowed from biology. For example, Particle Swarm Optimisation (PSO) algorithm, developed by James Kennedy and Russel Eberhart in the mid-1990s (James & Russell, 1995; Luke, 2013), was inspired by swarming and

flocking behaviours in animals. In PSO, every solution (or particle) is assigned randomised velocity and position vectors which are used to traverse the decision space. A more detailed description of the algorithm can be found in an overview by Poli et al. (2007). Other particularly popular set of optimisation techniques is known collectively as Evolutionary Optimisation (EO), Evolutionary Computation (EC) or Evolutionary Algorithms (EAs).

1.5.1. Introduction to Genetic Algorithms

GAs, invented by John J. H. Holland (1975) at the University of Michigan, are the most widely used class of EAs. Due to the relationship to biology and evolution theory, many biological terms have been used to describe GAs (Table 1.1).

Table 1.1. Most common terms used to describe Genetic Algorithms. Adapted from Luke (2009)

Term	Meaning
Gene	A slot position in a chromosome and a smallest unit of an encoded solution that represents a decision
Chromosome	Encoded solution represented as a string of genes
Individual	A candidate solution
Population	A collection of chromosomes
Fitness	Determines the reproductive success Represents the quality of an encoded solution
Objective function	Function that estimates the fitness of a chromosome
Crossover	Process of combining two chromosomes to create one or more new individuals
Mutation	Random changes made to the encoded solutions
Reproduction/Recombination	Crossover and mutation
Parent	Individual used to generate new solutions (offspring)
Offspring	New solutions generated by applying crossover and mutation to parents
Selection	Process of determining which parent solutions will undergo crossover and mutation
Generation	One iteration of GA which includes selection, crossover, and mutation
Genetic operator	Operator that guides the algorithm towards a solution to a given a problem, e.g. selection, crossover, and mutation

GAs can be divided into generational and steady-state algorithms. Generational algorithms, which are more common, update the entire or most of the sample whereas steady-state algorithms update the same sample a few individuals at a time. Algorithm 1.1 describes a procedure of a basic generational GA. The algorithm initiates from a pool of typically randomly generated chromosomes. Parent chromosomes with a higher objective function value or fitness are selected for crossover and mutation to create new and hopefully better solutions for each new generation of the GA.

Algorithm 1.1. Pseudocode of a basic GA.

```
 1 procedure GeneticAlgorithm(popsize, max_gen, objective_function)
 2   parents = ∅
 3   gen = 0                                          ▻ generation counter
 4   best = □                                         ▻ placeholder for best individual
 5   Generate new parent population of popsize
 6   while gen < max_gens
 7     for each parent in parents
 9       Evaluate parent fitness with objective_function
10       if best = □ or fitness of parent > fitness of best
11         best = parent
12       end if
13     end for
14     offspring = ∅
15     for |parents| / 2 times
16       parent_a = SelectWithReplacement(parents)
17       parent_b = SelectWithReplacement (parents)
18       offspring_a, offspring_b = Crossover(parent_a, parent_b)
19       offspring = offspring U { Mutate(offspring_a), Mutate(offspring_b) }
20     end if
21     parents = offspring
22     gen += 1
23   end while
24   return best
25 end procedure
```

The most common selection operators are fitness-proportional selection (also known as roulette-wheel selection) and tournament selection. As the name suggests, fitness-proportional operator selects parent chromosomes with a probability that is proportional to their fitness. Tournament selection picks the best solution from a random population sample of size *t* (tournament size). Tournament selection with samples comprising two individuals is often referred to as binary tournament. It has

been extensively used due to its computational efficiency and better or equivalent convergence when compared to other selection methods that are available in the literature (Goldberg & Deb, 1991; Melanie, 1996). After the selection, the parent chromosomes are crossed over with a certain probability *pC* to create one or more offspring chromosomes. Commonly used crossover operators include a uniform crossover, which swaps each of the parent genes with a probability of 0.5, a single-point crossover, which selects a random point and swaps all genes beyond that point in either parent's chromosome between the two parents, and a multi-point crossover, which is a generalisation of a single-point crossover (Allmendinger, 2012).

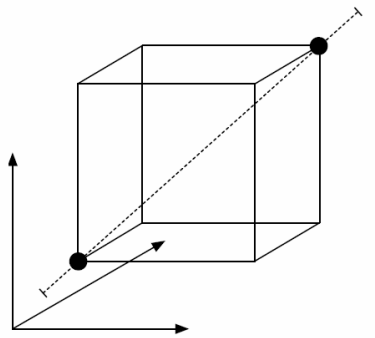


Figure 1.7. A cube formed by three-dimensional vectors (black circles) which represent positions of parent chromosomes in the decision space (Luke, 2009)

The original motivation for crossover was building-block hypothesis (BBH) (Holland & Goldberg, 1989) or, more formally, schema theory (Reeves, 2003). The basic premise of BBH is that highly fit individuals often share certain traits, i.e. building blocks, which are defined as a collection of genes set to certain alleles, i.e. positions in the chromosome. Crossover works by spreading these building blocks throughout the population. However, with the crossover alone, the search capabilities of a GA are severely limited. For example, if parent chromosomes were three-dimensional vectors, they would form a cube in a decision space (Figure 1.7). Crossover of these vectors will result in offspring that would lie at other corners of the cube. Therefore, conventional crossover operators are limited to search inside the bounding box

surrounding the parents (Luke, 2009). Moreover, repeated crossover and selection often eliminate certain genes, create copies of the same individual, and cause the GA to prematurely converge.

The usefulness of crossover operator has been extensively debated which led to the emergence of new recombination operators and Naïve Evolution algorithms that run without crossover (Fogel & Atmar, 1990; Senaratna, 2005). For example, Spears and Anand (1991) reported that for neural network modules and their control circuits GAs performed better without crossover. Naïve Evolution algorithms are supported by the many examples in nature of complex organisms which evolved without crossover, e.g. Bdelloidea – a class of microscopic pseudocoelomate freshwater animals (Senaratna, 2005). Furthermore, biologists consider mutation, not crossover, as the main source of new “raw genetic material” (Hartl, 1988). Commonly used mutation operators in GAs are ones that change each gene in a chromosome independently with some probability *pM*. It is worth noting that the variety of GAs is vast. There are many different strategies for performing selection, crossover, mutation, and even the underlying algorithm.

Unlike classical optimisation methods which make assumptions about the relationships between the variables, constraints, and the objective, GAs are flexible optimisers making minimal assumptions about the problem. Therefore, despite the lack of guarantee of finding the global optimum and the difficulty of designing the objective function, chromosome structure, and operators, GAs have been used to obtain approximate solutions to a wide range of complex linear and non-linear problem such as training neural networks (Chen & Liao, 1998), finding the optimal number, types, and positions of wireless transmitters (Ting et al., 2009), and creating a program capable of solving planning problems described in Planning Domain Definition Language (PDDL) (Brie & Morignot, 2005). Moreover, due to the multiplicity

in solutions, GAs have been quite popular for solving the multi-objective optimisation problems (Kalyanmoy, 2011). Since a population of solutions is processed in each iteration of a GA, the outcome is also a population of solutions. If an optimisation problem has a global optimum, then all chromosomes can be expected to converge to it. Alternatively, if an optimisation problem has multiple optimal solutions, GAs can capture them in its final population (Deb, 2001).

1.5.2. Stochastic and Multi-Objective Approaches

For multi-objective optimisation problems, two or more objective functions need to be evaluated simulatenously. Moreover, these objective functions are often contradictory to each other. A solution that is good for one objective function might do so at the cost of a less optimal value for another function. Solving multi-objective problems with or without the presence of constraints leads to a set of trade-off solutions popularly known as a Pareto front. Each optimal solution in the Pareto front is called a non-dominated solution. For example, in Figure 1.8, solutions A and B are non-dominated. A good survery on the history of multi-objective decision analysis and optimisation methods is provided by Köksalan et al. (2011).

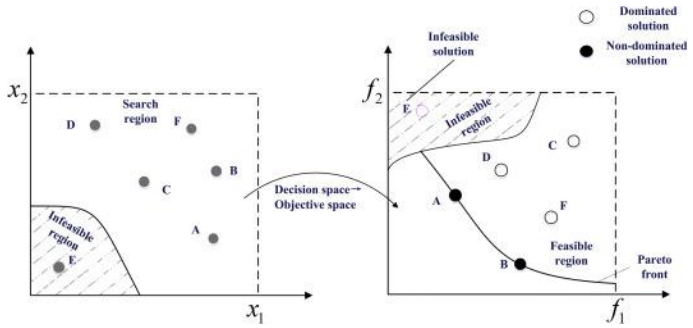


Figure 1.8. Relationship between the design (x1, x2) and objective (f1, f2) spaces of a two-objective optimisation problem (Source: Cui et al., 2017).

The first multi-objective GA, Vector-Evaluated Genetic Algorithm (VEGA), was proposed by (Schaffer, 1985). There have been several other multi-objective evolutionary algorithms (MOEA) developed over the years such as Niched Pareto

Genetic Algorithm (NPGA) (rey Horn et al., 1994), Non-Dominated Sorting Genetic Algorithm (NSGA) (Svinivas, 1995), Strength Pareto Evolutionary Algorithm (SPEA) (Zitzler & Thiele, 1999) and SPEA2 (Zitzler et al., 2001), Pareto Envelope-based Selection Algorithm (PESA) (Corne et al., 2000), Non-Dominated Sorting Genetic Algorithm-II (NSGA-II) (Deb et al., 2002), and many others.

One of the most attractive features of heuristics compared to mathematical programming is that they can be easily integrated with other methods such as Monte Carlo simulation (discussed in Chapter 6) which can be used to represent complex problem features and uncertainties that cannot be straightforwardly modelled by mathematical equations. A general simulation-based optimization method comprises an optimization part that guides the search process and a simulation part used to evaluate performances of candidate solutions. Compared with mathematical programming techniques, simulation-based optimization methods replace the analytical objective function and constraints by one or more simulation models. Iteratively the output of the simulation is used by the underlying optimisation algorithm, such as GA, to guide the search for the optimal solution(s). A comprehensive review of approaches to addressing different uncertainties using EAs is provided by Jin and Branke (2005). A more recent survey by Gutjahr and Pichler (2016) includes reviews of non-scalarising mathematical programming- and heuristic-based stochastic multi-objective optimisation. For example, Eskandari et al. (2005) integrated a simulation model with a stochastic nondomination-based multi-objective GA and introduced new genetic operators to enhance the algorithm's performance. Ding et al. (2006) proposed a multi-objective GA combined with a simulation procedure for supply chain optimisation. Amodeo et al. (2009) combined a discrete-event simulation procedure with SPEA-II, NSGA-II, and multi-objective PSO to determine the inventory policy of a single product supply chain, taking into account the maximization of customer service level and the total inventory cost. Syberfeldt et al. (2009) used a multi-

objective evolutionary algorithm supported by an artificial neural network, combined with a simulation routine to improve a manufacturing cell at Volvo Aero in Sweden.

1.5.3. Lot Sizing using Genetic Algorithms

There have been a number of papers reporting GA-based optimisation approaches for solving lot sizing and job-shop scheduling problems. Most of the approaches can be broadly divided into two classes depending on the encoding strategy: direct representation and indirect representation (Oyebolu et al., 2017). In a direct representation, the sequence and lot sizes are encoded in the chromosome directly. In an indirect representation, a chromosome typically encodes a set of rules or a permutation-based solution. A construction heuristic is then used to derive a schedule from the permutation or encoded rules. For example, Kimms (1999) used a two-dimensional matrix encoding strategy to solve a multi-level, multi-machine proportional lot sizing and scheduling problem formulated as a mixed-integer programming problem. The matrix contained rules for selecting the set up state for a machine at the end of a period. A construction heuristic was used to translate the matrix into the solution starting from the end of planning horizon. There have been multiple construction heuristics developed for a variety of problems. Branke and Mattfeld (2005) demonstrated an approach of penalising early idle times to increase scheduling flexibility and enhance overall performance for dynamic job-shop scheduling problems. Ho et al. (2006) proposed two construction heuristics for the single-level uncapacitated dynamic lot-sizing problem, extending the work of Silver (1973). Almada-Lobo et al. (2007) presented a five step heuristic to solve a multi-item capacitated lot-sizing problem with sequence-dependent setup times and costs from the glass industry. James and Almada-Lobo (2011) developed a general-purpose approach combining heuristics and mixed integer programming to find high quality

solutions to the single- and parallel-machine capacitated lot sizing and scheduling problem with sequence-dependent setup times and costs.

Jans and Degraeve (2008) noted that most of the heuristic-based optimisation methods developed for solving lot sizing problems were validated using artificial data and were limited in terms of the assumptions made, e.g. unlimited capacity, making the application to real-life problems troublesome. This thesis addresses this gap by developing scheduling models that address the most common features of biopharmaceutical industry, e.g. storage and shelf-life limitations, and are validated using industrially-relevant case studies either from real life or from the literature.

1.6. Related Work

Planning and scheduling of biopharmaceutical manufacture is a complex combinatorial optimisation problem further complicated by the unique features of biopharmaceutical production. Saraph (2001) noted that the biopharmaceutical manufacturing process is a mix of discrete and continuous processes, the size of batch and production capacity tend to vary depending on the processing stage, and the common utilities, e.g. water, are shared. Furthermore, most companies typically have a portfolio of various products manufactured across a network of owned and contract manufacturing facilities with wide-ranging production capabilities. Biopharmaceutical products may be unstable and thus have specialised and costly transportation and storage requirements. Biopharmaceutical companies are also required to meet high-quality standards and prove they can deliver a consistent manufacturing process. The high-quality standards are achieved by rigorous cleaning and sterilisation between individual production campaigns. Based on the report by Langer (2009), the top eight factors that create biopharmaceutical production capacity constraints are physical capacity of downstream processing equipment as well as

fermentation/bioreactor equipment, inability to retain and lack of new experienced technical and production staff, lack of financing for production expansion, costs associated with downstream purification, inability to optimise overall system and general inability to meet demands for finished product.

1.6.1. Process Design and Optimisation

The area of planning and scheduling in the biopharmaceutical industry has not received as much attention as bioprocess design and optimisation (Majozi et al., 2015). The development of computer-aided design tools for bioprocessing began in the mid-1980s (Farid et al., 2007). The vast majority of bioprocess design optimisation methods have been based on mathematical programming and simulation techniques.

Simulation-based approaches have been especially popular at modelling the impact of uncertainties within a biopharmaceutical manufacturing environment for more effective use of resources and improved economic performance. Farid and coworkers (Farid et al., 2000, 2001; 2005, and 2007) presented the SimBiopharma software tool to evaluate biopharmaceutical manufacturing alternatives in terms of cost, time, yield, resource utilisation, and risk. Incorporating uncertainty allowed users to make decisions based on both the expected outputs as well as the likelihood of achieving them. The key features of the tool included interactive graphics, task-oriented representation, bioprocess economics, dynamic simulation, risk analysis and multi-attribute decision-making. The benefits of this integrated approach were illustrated with an evaluation of stainless steel versus single-use or disposable facilities for clinical material preparation. Lim et al. (2005 and 2006) and Pollock et al. (2013) built on these decisional tools to evaluate the impact of uncertainty in fermentation titres, DSP yield, contamination rates on the design and robustness of perfusion culture based processes compared to fed-batch processes. Stonier et al. (2012) developed

these decisional tools to identify facility limits of legacy mAb facilities in terms of downstream capacities, assess the current and future robustness of these facilities to increasing cell culture titres and determine robust purification configurations given titre fluctuations. Stonier et al. (2013) and Yang et al. (2014) further leveraged the stochastic datasets generated from such tools with datamining tools (e.g. principal component analysis, clustering algorithms, decision trees) so as to be able to predict the root cause of facility fit issues.

Some of the earliest works to address the optimisation of the design structure and process variables in biopharmaceutical multi-product facilities with mixed-integer non-linear programming (MINLP) were presented by Montagna et al. (2000) and Asenjo et al. (2000). Vasquez-Alvarez and Pinto (2004) developed a MILP-based optimisation model to optimise chromatography unit operations. Brunet et al. (2012) created a hybrid simulation-mixed-integer with dynamic optimisation approach for the design of USP and DSP units in a single-product process. Simaria et al. (2012) proposed a multi-objective GA-based approach for the selection and optimisation of purification sequences and chromatography column sizing strategies. Allmendinger et al. (2012) presented a GA for the discovery of chromatography equipment sizing strategies for antibody purification processes under uncertainty. The optimisation of the chromatography column-sizing design in the mAb purification processes was also addressed by Liu et al. (2013) who applied MINLP to minimise the total cost.

1.6.2. Portfolio Management and Capacity Planning

Early work on biopharmaceutical portfolio management and capacity sourcing decisions used simulation models. For example, Rajapakse et al. (2005) and (2006) presented a Decision Support Tool based on Monte Carlo simulation to predict the process and business outcomes for portfolios of biopharmaceutical products in the

development pathway. At the time of writing, the literature on the use of alternative optimisation techniques such as GAs or hybrid methods in the pharmaceutical and biopharmaceutical industry was somewhat limited. Most of the publications focused on the optimisation of process design (discussed in the previous section) and the management of product portfolios rather than capacity planning and scheduling. The few that exist are discussed below.

On the pharmaceutical portfolio management front, Blau et al. (2004) reported a hybrid discrete simulation and GA-based approach for selecting a sequence of pharmaceutical products that maximises the expected economic returns at an acceptable level of risk for a given level of resources in a new product development pipeline. Varma et al. (2008) expanded the work accomplished by Blau et al. (2004) and proposed an integrated resource management tool to maximise portfolio's expected net present value, while keeping both risk and drug development times under control. The framework was based on the combination of a stochastic simulation of the pharmaceutical work flow process, a MILP formulation that acted as a "resource manager", and a Genetic Algorithm based "strategy learner" which was used to assess how the various strategies of resource allocation affect the financial and cycle time performance of the simulated portfolio of drug candidates. On the biopharmaceutical portfolio management front, George and Farid (2008) developed a stochastic, multi-objective optimisation framework based on probabilistic, model-building GAs for the optimisation of decisions related to portfolio selection, timing, and capacity sourcing decisions. Probabilistic model-building GAs belong to a class of EAs known as Estimation of Distribution Algorithms (EDAs). EDAs differ from most conventional algorithms by using explicit probability distributions represented by a model class, e.g. a Bayesian network or a multivariate normal distribution. Nie et al. (2012) presented a stochastic, GA-based decision-support tool to address the

decisions involved in portfolio management at both the drug development process level and the portfolio level.

Table 1.2. lists all relevant literature on capacity planning and scheduling optimisation methods in the biopharmaceutical industry to date. The vast majority of the research has focused on discrete-time MILP formulations adapted from the pharmaceutical and chemical engineering industries. The problem of task scheduling for new product development in the pharmaceutical industry was first developed by Schmidt and Grossmann (1996). According to Gatica et al. (2003), the first paper addressing the capacity planning problem as well as product selection decisions in the pharmaceutical industry appeared in 1999 by Rotstein et al. (1999). They presented a stochastic capacity planning model incorporating clinical trials uncertainty. A wide range of deterministic and stochastic models addressing such problems in the pharmaceutical industry have been developed since then. Gatica et al. (2003) presented a realistic approach to optimise a product portfolio subject to the outcome of the clinical trials. The proposed model included a multi-stage, multi-scenario case, and four outcomes, i.e. high success, target success, low success, and failure. It was based on previous pharmaceutical product portfolio optimisation models, such as Rotstein et al. (1999) and Papageorgiou et al. (2000), and was capable of considering whether it is more beneficial to outsource the manufacturing process or maintain the investment in the facility. Brastow and Rice (2003) demonstrated how Monte Carlo simulation could be used to identify the probability of having too much or too little manufacturing capacity for a network of pharmaceutical facilities. Levis and Papageorgiou (2004) presented a systematic mathematical programming approach for long-term, multi-site capacity planning under uncertainty in the pharmaceutical industry, simultaneously addressing the problem of product management. They also provided an extensive review of the publications addressing the problem of portfolio optimisation and task scheduling in the pharmaceutical industry.

Table 1.2. Resume of biopharmaceutical literature planning and scheduling optimisation. Adapted from Majozi et al. (2015) and extended.

Source	Model Characteristics	Methods
Samsatli and Shah (1996a), Samsatli and Shah (1996b)	▪ Two stages optimisation: first stage, processing rates and conditions of unit operations/equipment capacities through dynamic optimisation (gProms); second stage, scheduling and design adjustments of intermediate storage.	▪ MILP model (STN framework) ▪ Discrete-time representation ▪ Cyclic schedule (48 h/68 h) ▪ Maximise operating profit
Lakhdar et al. (2005)	▪ Medium-term planning and scheduling (1–2 years) ▪ Determines campaigns durations and sequence, production quantities, inventories, and product sales	▪ MILP model ▪ Discrete-time representation ▪ Maximise operating profit
Lakhdar et al. (2006)	▪ Medium-term planning (1–3 years) considering uncertainty in the fermentation titres ▪ Considers storage constraints ▪ Results compared within deterministic model, a two-stage programming model accompanied by an iterative construction algorithm, and a proposed CCP model	▪ MILP model derived using CCP ▪ Discrete-time representation ▪ Maximise operating profit ▪ Multi-scenario stochastic programming

Table 1.2. (continued) Resume of biopharmaceutical literature planning and scheduling optimisation. Adapted from Majozi et al. (2015) and extended.

Source	Model Characteristics	Methods
Lakhdar et al. (2007)	▪ Long-term planning, first solved as a single objective problem (maximise operating profit) and capacity analysis was conducted; then extended to allow multiple objectives through goal programming	▪ MILP model ▪ Discrete-time representation ▪ Maximise operating profit ▪ Minimise total adverse deviations to targets: cost, customer service level, and capacity utilisation
Lakhdar and Papageorgiou (2008)	▪ Medium-term planning under uncertain fermentation titres ▪ Storage constraints ▪ Proposed future extension to multi-stage framework to allow uncertainty to be revealed gradually at any time period; further, proposed decomposition and approximation solution methods	▪ MILP model ▪ Discrete-time representation ▪ Iterative algorithm for large-scale problem ▪ Maximise operating profit
Miller et al. (2010)	▪ Core mathematical programming solver designed around a uniform discretisation model and customised outer layer to address biologics process behaviour ▪ VirtECS Scheduler Software ▪ Intermediate material storage consideration	▪ MILP model (RTN framework) ▪ Discrete-time representation ▪ Monte Carlo stochastic parameters

Table 1.2. (continued) Resume of biopharmaceutical literature planning and scheduling optimisation. Adapted from Majozi et al. (2015) and extended.

Source	Model Characteristics	Methods
Gicquel et al. (2012)	▪ Hybrid flow shop scheduling problem ▪ Zero intermediate capacity and limited waiting time between processing ▪ Suggest heuristic solution as future work	▪ MILP model ▪ Discrete-time representation ▪ Minimise total weighted tardiness
Kabra et al. (2013)	▪ Unit-specific event-based continuous-time representation ▪ Multi-period scheduling of multi-stage multi-product process ▪ Based on STN representation	▪ MILP model ▪ Continuous-time representation ▪ Maximise operating profit
Siganporia et al. (2014)	▪ Long-term planning ▪ The model comprised specific features to account for products with fed-batch or perfusion culture processes ▪ Utilised rolling time horizon approach to obtain greater optimality in less computational time than the full-scale model	▪ MILP model ▪ Discrete-time representation ▪ Minimise total cost
Vieira et al. (2016)	▪ Multi-period scheduling of multi-stage multi-product process ▪ RTN continuous-time single-grid formulation	▪ MILP model ▪ Continuous-time representation ▪ Maximise operating profit

Table 1.2. (continued) Resume of biopharmaceutical literature planning and scheduling optimisation. Adapted from Majozi et al. (2015) and extended.

Source	Model Characteristics	Methods
Oyebolu et al. (2017)	▪ Inspired by GA approaches to job shop scheduling ▪ Proposed a problem-tailored construction heuristic for scheduling product demands across multiple facilities	▪ GA model ▪ Maximise operating profit
Jankauskas et al. (2017)*	▪ Medium-term planning ▪ Multi-period scheduling of multi-stage multi-product process ▪ Developed a continuous-time scheduling heuristic using variable-length chromosomes	▪ GA model ▪ Continuous-time representation ▪ Maximise operating profit

* This work is part of this thesis.

One of the first frameworks for biopharmaceutical capacity planning and scheduling was developed by Samsatli and Shah (1996b). They addressed the design and short-term scheduling of biopharmaceutical processes using MILP and STN formulations. The first medium-term capacity planning model for a multi-product, multi-suite biopharmaceutical facility was presented by Lakhdar et al. (2005). Their approach helped to determine the optimal durations and sequence of production campaigns together with product inventory, sales, and late deliveries profiles. Furthermore, the proposed MILP based optimisation method was shown to find more optimal solutions than the industrial rule-based approach. Kabra et al. (2013) compared this discrete-time model with a continuous-time multi-period scheduling of multi-stage, multi-product process based on an STN framework, reporting an improved objective function value. Vieira et al. (2016) also compared Lakhdar et al. (2005) discrete-time

model with a new MILP model based on RTN continuous-time single-grid formulation. They reported even better objective function values.

The randomness of the biopharmaceutical manufacturing environment such as uncertain yield and risks of contamination, can cause significant scheduling and planning difficulties for the biopharmaceutical manufacturing campaigns. To address this, Lakhdar and Papageorgiou (2006) compared a chance-constrained programming (CCP) model with a deterministic MILP and two-stage programming approach combined with an iterative construction algorithm for medium-term planning of biopharmaceutical manufacture under uncertain fermentation titres. The proposed methodology was reported to yield better results than a deterministic MILP model. Lakhdar and Papageorgiou (2008) improved their work presented in 2005 with an iterative algorithm for solving large-scale biopharmaceutical capacity planning and scheduling problems with uncertain fermentation titres.

The optimisation of biopharmaceutical manufacturing capacity often involves many multiple conflicting criteria and objectives to be considered. Lakhdar et al. (2007) addressed the challenge of making long-term (15 years), multi-site capacity planning decisions given multiple strategic criteria such as risk, cost, and customer service levels. The problem was first solved as a single objective problem to maximise operating profit, and then extended using goal programming to allow for multiple objectives, i.e. cost, customer service level, and capacity utilisation.

The vast part of the research on biopharmaceutical manufacture planning has been limited to either batch or fed-batch processes. However, a more recent, large-scale discrete-time MILP model was presented by Siganporia et al. (2014) to optimise long-term capacity plans for a portfolio of biopharmaceutical products, with either batch or perfusion bioprocesses, across multiple facilities to meet quarterly demand.

The work presented by Oyebolu et al. (2017) is one of the few GA-based planning and scheduling optimisation models for the biopharmaceutical industry. Taking inspiration from GA-based approaches to job-shop scheduling, they proposed a problem-tailored construction heuristic for scheduling demands of multiple products sequentially across several facilities to generate a long-term manufacturing schedule. Compared to the aforementioned construction heuristics (Section 1.5.3), theirs is different in that it inserts jobs (manufacturing campaigns) in an order of importance determined by the GA and not necessarily in any chronological order. The approach is based on an indirect representation of the problem using a permutation of all the product demands. The sequence of demands encoded in a chromosome determines the order by which the construction heuristic schedules production campaigns. The construction heuristic explores a number of different scheduling alternatives, e.g. schedule as late as possible, schedule next to previous demand, split demand, and picks the best one based on feasibility and the smallest additional cost. The approach outperformed a related discrete-time MILP model on a single-objective long-term biopharmaceutical capacity planning problem from the literature (Lakhdar et al., 2007).

1.7. Aims and Outline of Thesis

As discussed earlier, much of process planning and scheduling research for biochemical engineering processes has been based on MILP formulations using discrete-time representation (Table 1.2). It is acknowledged that the development of models for production planning and scheduling of biopharmaceutical processes has been fairly unexplored (Vieira et al., 2016). This work is particularly motivated by the insufficient research of GA-based capacity planning and scheduling optimisation methods in the biopharmaceutical industry. The key objectives of this work are to investigate the applicability of GAs for capacity planning and scheduling of

biopharmaceutical facilities and to develop a flexible framework that would facilitate the biopharmaceutical industry's strategic and operational decision-making. The following areas will be explored:

- Medium- and long-term planning
- Discrete- and continuous-time representations
- Single- and multi-objective problems
- Deterministic and stochastic optimisation

Chapter 2 describes a general problem statement and lists the key challenges of biopharmaceutical capacity planning and scheduling. It also describes the framework and the technical details of the GA-based DST developed in this thesis for tackling biopharmaceutical scheduling problems.

Chapter 3 serves as a starting point in understanding the implementation challenges of GA-based biopharmaceutical capacity planning and scheduling optimisation. This is accomplished by developing GA-based approaches for solving single-objective biopharmaceutical capacity planning and scheduling problems using the simpler discrete-time representation. The performance of the GA is compared with MILP models on industrial case studies of medium- and long-term planning from the literature. Moreover, a PSO-based meta-optimisation approach is utilised to automatically set the parameters of the GA. With some caveats, such as rolling time horizon, the GA is demonstrated to be capable of generating exact or near-optimal solutions to discrete-time MILP problems of biopharmaceutical capacity planning and scheduling.

The early work of this thesis presented in **Chapter 3** identified the shortcomings of discrete-time representation such as unutilised production time and unnecessarily

high model complexity. **Chapter 4** improves upon **Chapter 3** by presenting a novel variable-length GA (which is the core of the GA-based DST developed in this thesis) and a continuous-time scheduling heuristic for efficient and more realistic medium-term scheduling of biopharmaceutical manufacture. Using the variable-length chromosome structure, the GA is capable of adapting to the planning problem from a single gene by either growing or shrinking in length. The continuous-time scheduling heuristic accounts for constraints and features such as product-dependent changeovers, varying manufacturing yields, multiple intermediate demand due dates, and storage and shelf-life limits. The performance of the method is evaluated on two industrial case studies and contrasted with related discrete- and continuous-time MILP models.

Chapter 5 extends the variable-length GA from **Chapter 4** with a multi-objective component. The continuous-time scheduling heuristic is also adapted to suit a different biopharmaceutical facility model with rolling product sequence-dependent changeovers and to account for product quality control and assurance (QC/QA) checks. The functionality of the multi-objective approahc is highlighted on an industrial case study developed together with Eli Lilly & Company. The GA is used to optimise both the throughput and monthly product inventory levels of a multi-product biopharmaceutical facility over a three year period.

In **Chapter 6**, Monte Carlo simulation is integrated into the multi-objective variable-length GA from **Chapter 5** for generating production schedules under variable product demand. The advantages and performance of the approach are demonstrated on a real life industrial case study and contrasted to a deterministic optimisation approach that neglects the uncertainty in product demand. Moreover, the chapter describes how the computationally intensive Monte Carlo simulation can be accelerated using a GPU that achieved a 30-fold speed-up.

Chapter 7 contains an implementation plan for commercialisation of the work generated in this thesis. It describes not only the architecture and the design details of a proposed user interface but also discusses how the application could be priced and delivered to clients. Finally, the conclusions of this thesis and the plausible directions for future work are provided in **Chapter 8**. A list of publications (published and in progress) resulting from this thesis is given in **Appendix A**.

2. Decision Support Tool: Requirements and Design

The previous chapter provided an overview of the biopharmaceutical industry and the existing, mostly MILP-based methods for assisting biopharmaceutical manufacturers in making decisions about when, where, and how long they should manufacture a product. Despite the number of works available in the literature, the actual adoption of MILP-based optimisation models has been relatively slow in the biopharmaceutical industry. Due to specialist knowledge, high skill requirements, and lack of transparency associated with mathematical programming-based methods (Mustafa et al., 2006; Widmer et al., 2008), production scheduling especially short- and medium-term is still often carried out using mostly manual spreadsheet-based methods. Another reason why simpler methods are so widely used is because they can be easily explained to and understood by the business stakeholders. Fortunately, the research principles at the basis of GAs are generally more accessible to inexperienced users making the algorithm an attractive alternative. Moreover, due to their inherent flexibility, GAs can be easily adapted or combined with other types of methods and applied to a wide-range of problems.

The literature on mathematical programming-based scheduling optimisation methods puts a lot of emphasis on the optimality of solutions. It is nearly impossible to measure the optimality of solutions generated using heuristic methods such as GA. Nevertheless, in most real-life scenarios, it is sufficient to compare the performance of a heuristic-based scheduling tool against a benchmark generated using, for example, an expert system or industrial rule-based planning. If there are significant gains in the objective function values, then the solution does not need to be proven to be optimal.

In this chapter, the requirements and design of a flexible GA-based DST for efficient single- and multi-objective capacity planning and scheduling of multi-product biopharmaceutical facilities are presented. **Section 2.1** outlines a general statement and key challenges of biopharmaceutical capacity planning and scheduling problems which will be tackled in the next chapters of this thesis. **Section 2.2** lists the high-level requirements and defines the components of the tool's framework needed to meet them.

2.1. Problem Statement and Challenges

Despite the wide variety of biopharmaceutical capacity planning and scheduling problem classes, every problem statement can be defined in the following general way:

- Given:
    - Production facility data such as production capacities, number of *USP* and *DSP* suites, and availability of utilities.
    - Processing data such as *USP* and *DSP* processing times and material requirements.
    - Costs (optional), e.g. manufacturing, storage, backlog penalty, and waste disposal.
    - Production targets or product demand.
- Determine:
    - An optimal schedule that would satisfy one or several strategic criteria.

Biopharmaceutical facilities can have various manufacturing capabilities and plant topologies with multiple *USP* and *DSP* suites (see Figure 2.1). For the simple case, both *USP* and *DSP* can be treated as a monolithic, black-box process without explicit

discretisation into individual suites. However, allowing the various biopharmaceutical manufacturing stages to be modelled separately can yield more realistic production schedules though at the cost of higher computational complexity and increased modelling challenges.

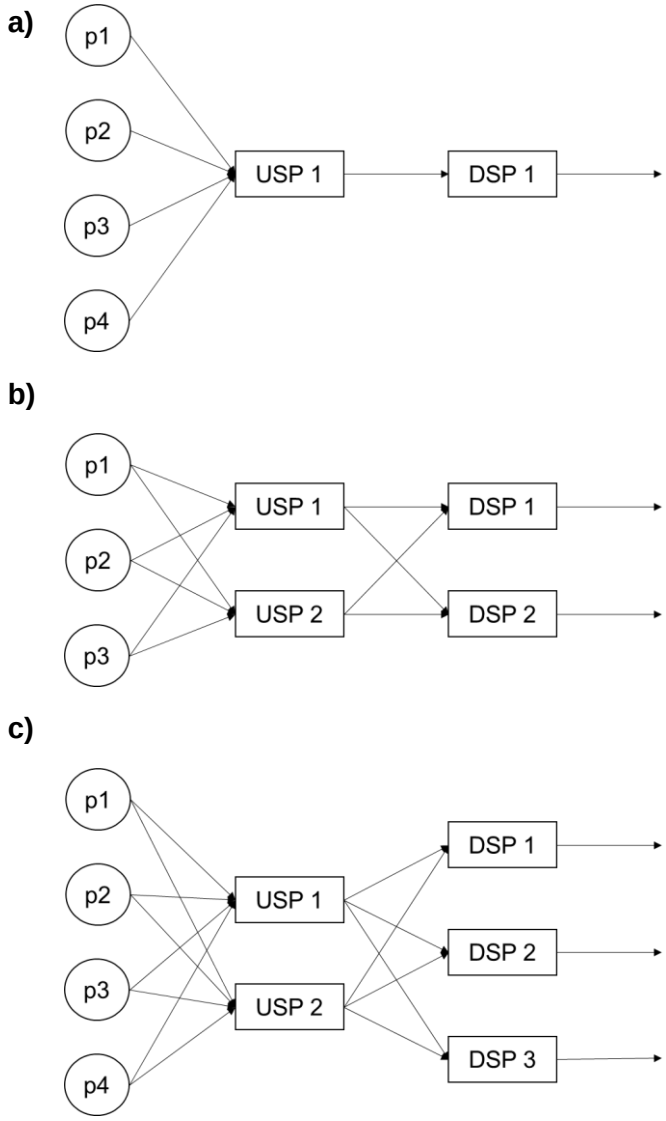


Figure 2.1. Examples of different biopharmaceutical facility topologies (different USP to DSP suite number ratios): (a) 1:1, (b) 2:2, (c) 2:3. All three examples will be tackled in the later chapters of this thesis.

Capacity planning and scheduling problems are often subject to several constraints, e.g. biopharmaceutical companies are required to produce a minimum number of batches for the regulatory bodies, the product demand must be met on time, and product waste needs to be minimised or avoided completely. Furthermore, most real-life biopharmaceutical capacity planning and scheduling problems have multiple objectives. It is generally desirable to maximise the facility throughput and maintain the strategic product inventory levels at specific monthly targets. A straightforward way to maximise facility throughput is to run fewer but longer manufacturing campaigns (Figure 2.2.a) which reduces the number of product changeovers and increases the available time for manufacturing. However, having longer and infrequent campaigns can lead to uneven product inventory levels and periods where the stock is dangerously low (Figure 2.2.b). Therefore, running shorter but more frequent manufacturing campaigns (Figure 2.2.c.) would ensure that product inventory is re-stocked often and there is enough of it at any point in time to meet the product demand for the next 6 or 9 months in case of unplanned facility shutdowns or other emergencies (Figure 2.2.d). One of the key challenges that the biomanufacturers face is striking a balance between these two objectives.

The capacity planning and scheduling problem of biopharmaceutical manufacture is further complicated by other factors such as limited shelf-life, storage capacity limitations, and the types and durations of product changeovers. Figure 2.3. depicts an example of a biopharmaceutical product changeover that is widely used in the capacity planning and scheduling MILP-based models reported in the literature (Lakhdar et al., 2005; Lakhdar et al., 2007; Siganporia et al., 2014). During this type of product changeover, all tasks of the previous manufacturing campaign need to be completed before the clean-up process and the subsequent manufacturing campaign can begin. The time required to switch between products includes the time required to clean the suites and equipment, and it often depends on the sequence of

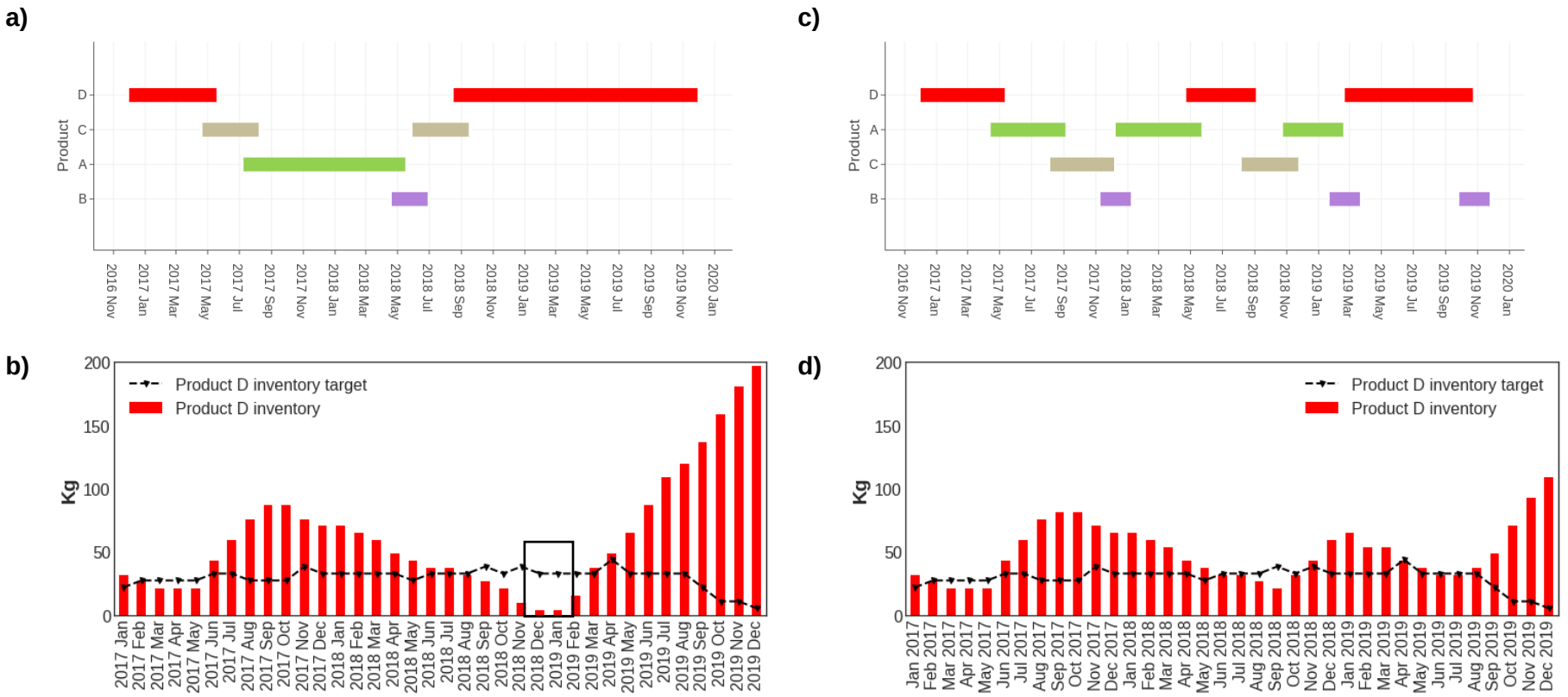


Figure 2.2. A comparison between two production schedules and the corresponding inventory levels of product D. In (a), a schedule with fewer but longer manufacturing campaigns has higher total throughput albeit at the cost of (b) unbalanced product inventory with periods of extremely low stock (highlighted by the rectangle). On the other hand, a product schedule in (c) has more frequent but shorter product campaigns and, as a result, lower total throughput but also (d) better maintained product inventory levels.

the product campaigns. Provided that the different stages of biomanufacturing process (Figure 1.1) are carried out in separate, self-contained processing suites, a more efficient *rolling product changeover* can be implemented. For example, while Figure 2.4.a depicts what looks like two overlapping manufacturing campaigns of different products, Figure 2.4.b shows that a rolling changeover takes place between the different early manufactured stages while the product is still being produced in other suites. In this way, more time is made available for the manufacturing of products by minimising the idle waiting times in-between production campaigns.

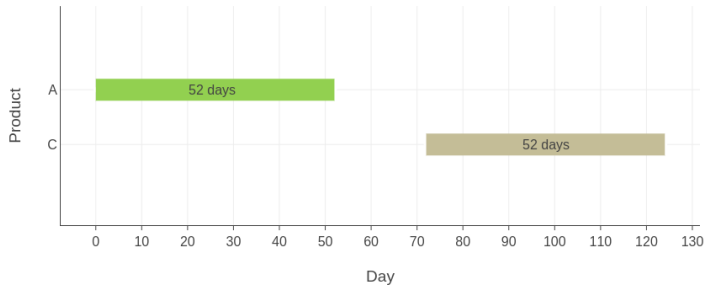


Figure 2.3. An example of a traditional product changeover. The new manufacturing campaign of product C can only take place after all tasks of product A campaign are finished.

a)

b)

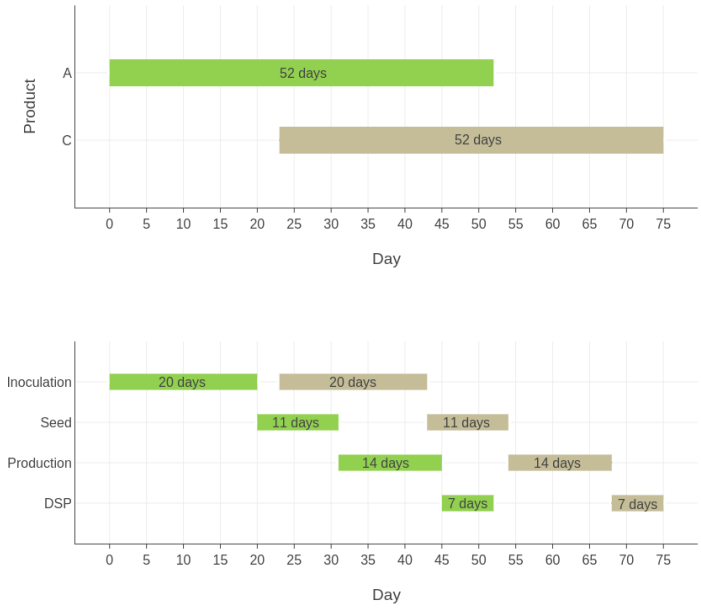


Figure 2.4. An example of a rolling product changeover. Numbers inside the blocks correspond to the duration of the corresponding task while a gap between the different task blocks denotes a changeover.

Given the unique features of the biopharmaceutical manufacture and the wide variety of objectives and constraints, the manual creation of capacity plans can quickly become an unsustainable practice. Even for simpler discrete-time based models, the complexity of biopharmaceutical capacity planning and scheduling problems grows exponentially with the increasing number of products and time periods (see Table 2.1).

Table 2.1. Minimum number of solutions for different cases over 8 years with a time period of one month (Source: Siganporia, 2016).

Number of products	Number of facilities	Number of solutions
2	2	10^{91}
4	4	10^{289}
6	6	10^{522}
10	10	10^{1056}

There is an obvious, strong need for methods that would help find the best use of production resources in order to satisfy production goals, i.e. addressing the production capacity requirements and anticipating sales opportunities over a planning horizon of choice (Karimi et al., 2003). The three typical types of planning horizons are short-term, medium-term and long-term. Short-term planning decisions are comprised of every day scheduling of operations, e.g. job sequencing; medium-term planning involves making decisions on material requirements planning and lot sizing over the planning horizon in order to meet the demand and minimise overall costs; long-term planning comprises strategic decisions on product, equipment, process, facility location and design choices and resource planning (Karimi et al., 2003). The main scope of this thesis is efficient medium-term multi-objective scheduling biopharmaceutical manufacture in an existing multi-product facility.

2.2. Requirements and Design

There have been several discrete- and continuous-time MILP-based models for biopharmaceutical capacity planning and scheduling reported in the literature. In order to be considered as a feasible alternative to these models, a requirements specification was developed that describes what the GA-based DST should be able to achieve. The tool requirements are outlined below:

- *Ability to specify multiple objectives and constraints:* the tool needs to capture the most common objectives and constraints of biopharmaceutical production such as maximisation of profit and minimisation of costs. However, the availability of cost data is usually a bottleneck thus the tool needs to allow for other objectives that are non-monetary, including maximisation of production throughput, maintaining strategic inventory targets, meeting all product demands on time, and avoiding product waste.
- *Ability to specify product-specific characteristics:* in order to have practical value and reflect the biopharmaceutical manufacturing environment in a realistic way, the tool has to address the aforementioned complexities such as varying process durations and yields, product sequence-dependent changeovers, QC/QA approvals, storage and shelf-life limits.
- *Ability to instantiate new models or add new scheduling logic*: the description of the scheduling problem often changes during the initial stages of implementation. It is common for the original problem formulation to be continually modified and enhanced as additional information becomes available. Therefore, it is important to empower the production scheduler not only with the ability to make non-structural changes to the scheduling model such as adding new products and revising product demand but also with the ability to include different scheduling models. The tool needs to be able to cope with a variety of different

biopharmaceutical capacity planning and scheduling problems, biopharmaceutical facility designs, and manufacturing strategies.

- *Ability to achieve solutions in a timely manner*: the tool needs to be capable of not only generating optimal or close-to-optimal solutions but also do it so in a reasonable amount of time. The shorter the time to report a good schedule is, the more scenarios can be tested by production schedulers.
- *Ability to optimise under uncertainty*: the tool needs to be able to address the inherent uncertainties of biopharmaceutical manufacture such as product demand and to solve the scheduling problem with probability distribution-based input.

The tool generated in this thesis meets all of the aforementioned requirements which will be covered in the subsequent chapters. For example, Chapters 4-6 will demonstrate how the tool is used to meet a variety of monetary and non-monetary scheduling objectives and constraints, including maximisation of profit and simultaneous optimisation of throughput and product inventory levels subject to various constraints. The ability to specify product-specific characteristics and the ability to instantiate new models or add new scheduling logic are demonstrated in Chapters 4 and 5. For example, in Chapter 4 the tool is used to generate a 1-year long schedule for a biopharmaceutical facility with 2 *USP*/2 *DSP* suites manufacturing 3 products and a 1.5-year long schedule for a biopharmaceutical facility with 2 *USP*/3 *DSP* suites manufacturing 4 products. Moreover, in Chapter 4 the tool is used to schedule production for biopharmaceutical facilities with traditional product-dependent changeovers, whereas in Chapter 5 the tool is applied to a biopharmaceutical facility with rolling product sequence-dependent changeovers.

Choosing the right set of technologies and programming languages for the development of DSTs is an important decision that can have an impact on the ultimate flexibility and usability of the tool. For the DST to receive continuous support and

attention in the future from the developers and researchers alike, the chosen programming language(s) need to be sufficiently flexible and have a large and active community.

According to a popular yearly survey (Stack-Overflow, 2018), Python is the fastest-growing major programming language. Approximately 40% of over 100,0000 developers from around the world said Python was their primary programming language. Some of the reasons behind such popularity include succinct and intuitive language syntax, powerful open source libraries for data analysis and visualisation, and web-based application development toolkits which will be crucial to a commercial application of the tool (see Chapter 7). However, one of the major drawbacks of Python programming language is its performance. For example, multi-threading is not available out-of-the-box due to what is known as a Global Interpret Lock (GIL) that prevents more than one thread running in the interpreter (Beazley, 2010). Fortunately, Python can be easily integrated with other, lower level programming languages such as C and C++ that can help improve the performance. Faster execution speeds can benefit the user by allowing them to perform more runs and test more scenarios in less amount of time.

In thesis, C++ was used to develop most of the work presented, e.g. GAs, scheduling heuristics, Monte Carlo simulation. The main reasons for the choice were the performance benefits, relatively straightforward shared-memory parallelism using OpenMP compiler directives (demonstrated in Algorithm 2.1) (Dagum & Menon, 1998), and support for CUDA – an Application Programming Interface (API) for parallel programming using GPU resources (Nvidia, 2011). According to the survey mentioned earlier, C++ is still among the top 10 programming languages despite its complexity and relatively low safety. Both Python and C++ have the added benefit of being cross-platform development languages.

Algorithm 2.1. Parallel fitness assessment in C++ using OpenMP compiler directives. `#pragma` compiler directive tells the compiler to auto-parallelize the `for` loop with OpenMP. If a user is using a quad-core processor, the performance can be expected to be increased by up to 300% (in most cases).

```
#pragma omp parallel for
for (int i = 0; i < parents.size(); ++i) {
    fitness_function(parents[i]);
}
```

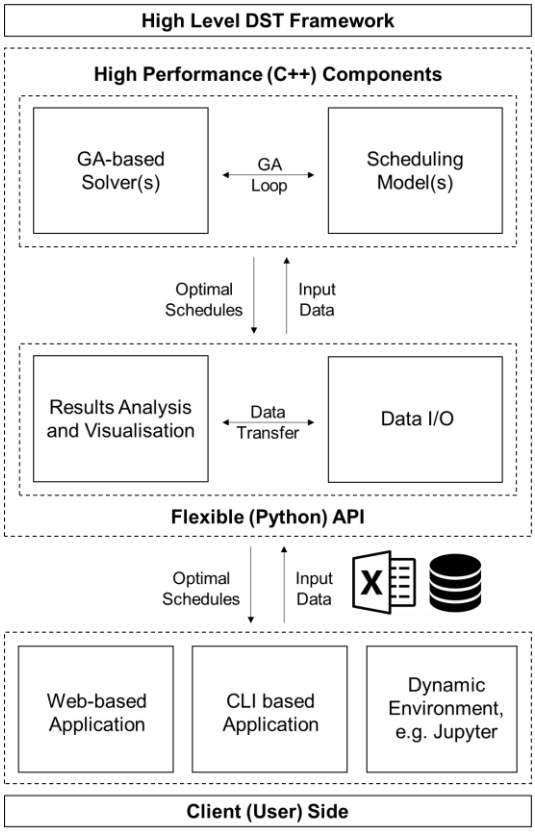


Figure 2.5. A high-level structure of the GA-based Decision Support Tool framework.

Figure 2.5 illustrates the framework of the GA-based DST developed in this thesis from a high level. The scheduling heuristics and GAs are only a piece of the overall framework. The framework supporting the execution of the models must be capable of communicating with other business applications as well as a number of spreadsheets, document files, and databases. Therefore, an API has been developed in Python that can be used in a variety of settings, including web-based applications,

command line interface (CLI) programs, and integrated development environments (IDEs). The API wrap-ups the high-performance extensions written in C++ containing the GAs and scheduling heuristics and provides an interface layer written in Python for data input/output (I/O) and results analysis and visualisation. The data can be stored in and read from either a relational database or document files. API usage examples can be found in Appendix B.

Chapter 7 will discuss the commercial application of the tool developed in this work, including trend charts for displaying the evolution of product inventory and delivery profiles, easy viewing and manipulation of input data, and the reporting on the timings of scheduled campaigns. Below are some of the example outputs from the GA-based DST. Figures 2.6 and 2.7 show examples of Gantt chart outputs generated for two case studies which will be discussed individually in later chapters of this thesis. The Gantt charts show the allocation of different product campaigns and allow a user to view the exact start and end dates of each campaign as well as the number of batches and/or kilograms produced. If needed the GA-based DST can also generate a Gantt chart illustrating the allocation of products to different biomanufacturing stages allowing user to view the start and finish of each individual batch (Figure 2.8).

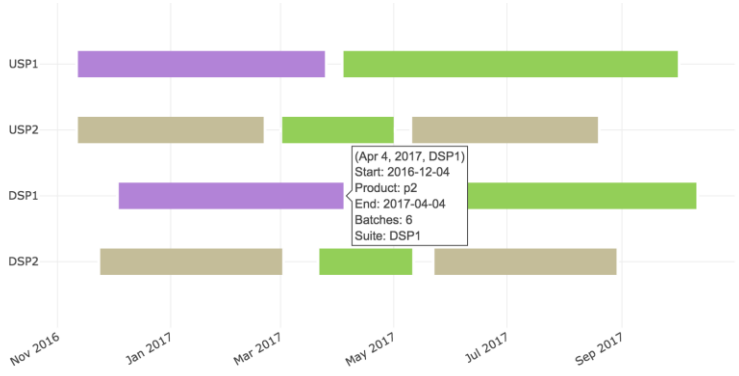


Figure 2.6. Gantt chart generated with the GA-based Decision Support Tool for a biopharmaceutical facility with traditional product changeovers and a 2:2 USP to DSP ratio manufacturing three products.

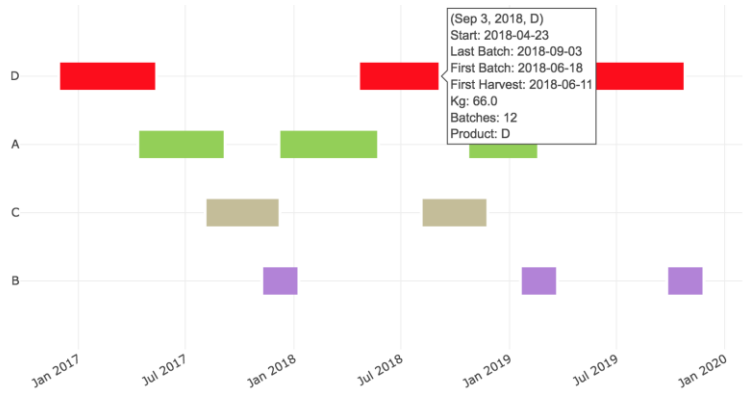


Figure 2.7. Gantt chart generated with the GA-based Decision Support Tool for a biopharmaceutical facility with rolling product sequence-dependent changeovers and a 1:1 USP to DSP ratio manufacturing four products.

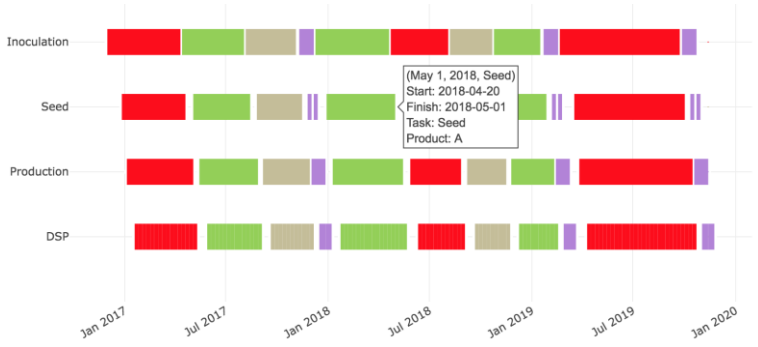


Figure 2.8. Production tasks Gantt chart example.

2.3. Summary

This chapter described the unique features and challenges of the biopharmaceutical capacity planning and scheduling problems. It also defined a framework for developing the GA-based DST which will be used to tackle single- (**Chapter 4**) and multi-objective problems with deterministic (**Chapter 5**) and uncertain product demand (**Chapter 6**) to illustrate its value.

3. Discrete-Time Biopharmaceutical Capacity Planning and Scheduling

3.1. Introduction

This chapter presents a fast GA-based approach to both medium- and long-term capacity planning and scheduling of single- and multi-site biopharmaceutical manufacture using discrete-time models. The proposed GA is demonstrated as a valid alternative to MILP to obtain near-exact solutions to close to real-world industrial case studies of capacity planning and scheduling of biopharmaceutical manufacture. Other contributions presented in this chapter include the chromosome encoding strategy, the algorithms describing the single-site/multi-suite and multi-site biopharmaceutical manufacture, the rolling horizon approach for solving larger, long-term capacity planning problems, and a PSO-based meta-optimisation approach for tuning the GA hyperparameters.

The performance of the GA depends on its hyperparameter values. For example, the rate of crossover controls the capability of the GA in exploiting the known parts of the search space, whereas the mutation rate controls the speed of the GA in exploring of new areas (Lin et al., 2003). The values of these parameters are quite often tuned one by one, i.e. by trial and error. However, this can be a time consuming process leading to suboptimal results, since the interactions between the parameters are ignored this way (Eiben et al., 1999). There have been a number of suggestions and theoretical investigations into the optimal values of crossover, mutation, and population size (e.g. Schaffer & Morishima, 1987; Goldberg & Deb, 1991; Back, 1993; Chipperfield & Fleming, 1995). The typical values of crossover and mutation rate have been reported to lie in the range 0.5-1.0 and 0.001-0.05 respectively. However, most

investigations were based on simple function optimisation problems with traditional chromosome encoding strategies and genetic operators. Therefore, their applicability for other types of problems and custom genetic operators is quite limited. An alternative to manual parameter tuning is meta-optimisation, i.e. the use of another optimisation algorithm to tune the GA hyperparameters. For example, Grefenstette (1986) applied a meta-GA to optimise the hyperparameters of another GA. An approach to automatically set the parameters of evolutionary algorithms can also be considered as antecedents of hyper-heuristics – a set of approaches that are motivated by the goal of automating the design of heuristic methods to solve hard computational search problems (Burke et al., 2013).

In this work, a PSO algorithm is used to tune the GA. Meta-PSO was chosen due to its simplicity and relatively low computational overhead (compared to using another GA) (Pandey et al., 2010) and suitability for the optimisation of functions with continuous inputs (Hassan et al., 2004).

3.2. Notation

3.2.1. Case Study 1

SETS

i	*USP* suites
j	*DSP* suites
p	products
t, θ	time periods

PARAMETERS

C_p	*USP* storage capacity of product *p* [batches]
F_p	*DSP* storage capacity of product *p* [batches]
CR_p	*USP* production rate of product *p* [batches/day]
FR_p	*DSP* production rate of product *p* [batches/day]
CT_p^{min}	min production time for product *p* in *USP* suite *i* [days]
CT_p^{max}	max production time for product *p* in *USP* suite *i* [days]
FT_p^{min}	min production time for product *p* in *DSP* suite *j* [days]

FT_p^{max}	max production time for product *p* in *DSP* suite *j* [days]
α_p	*USP* lead time of product *p* [days]
β_p	*DSP* lead time of product *p* [days]
ρ_p	*USP* storage cost of product *p* [RMU/batch]
ω_p	*DSP* storage cost of product *p* [RMU/batch]
ζ_p	*USP* product lifetime *p* [time periods]
σ_p	*DSP* shelf-life of product *p* [time periods]
λ_p	correspondence factor for *USP* to *DSP* production of product *p*
η_p	manufacturing cost of product *p* [RMU/batch]
ψ_p	changeover cost of product p [RMU / changeover]
τ_p	waste disposal cost of product *p* [RMU/batch]
v_p	sales price of product *p* [RMU/batch]
δ_p	backlog penalty of product *p* [RMU/batch]
D_{pt}	demand of product *p* at time period *t* [batches]

INTEGER VARIABLES

$product_{it}$	part of the chromosome containing product labels allocated at time period *t* to *USP* suite *i*
$product_{jt}$	part of the chromosome containing product labels allocated at time period *t* to *DSP* suite *j*
$time_{it}$	part of the chromosome containing the number of production days allocated at time period *t* to *USP* suite *i*
$time_{jt}$	part of the chromosome containing the number of production days allocated at time period *t* to *DSP* suite *i*
B_{ipt}	number of batches of product *p* produced at time period *t* in *USP* suite *i*
B_{jpt}	number of batches of product *p* produced at time period *t* in *DSP* suite *j*
CI_{pt}	number of batches of *USP* product *p* stored at time period *t*
FI_{pt}	number of batches of *DSP* product *p* stored at time period *t*
CW_{pt}	number of batches of *USP* product *p* wasted at time period *t*
FW_{pt}	number of batches of *DSP* product *p* wasted at time period *t*
S_{pt}	number of batches of product *p* sold at time period *t*
Δ_{pt}	number of batches of product *p* in backlog at time period *p*

BINARY VARIABLES

Y_{ipt}	1 if product *p* is produced in *USP* suite *i* at time period *t*; 0 otherwise
Y_{jpt}	1 if product *p* is produced in *DSP* suite *j* at time period *t*; 0 otherwise
Z_{ipt}	1 if a new campaign of product *p* is produced in *USP* suite *i* at time period *t*; 0 otherwise
Z_{jpt}	1 if a new campaign of product *p* is produced in *DSP* suite *j* at time period *t*; 0 otherwise

CONTINUOUS VARIABLES

CT_{ipt}	production time for product *p* in *USP* suite *i* during time period *t* [days]
FT_{jpt}	production time for product *p* in *DSP* suite *j* during time period *t* [days]
Profit	total profit (objective function) [RMU]

3.2.2. Case Study 2

SET

i	facilities
p	products
t, θ, ξ	time periods
PI_i	set of products that can be produced by facility i
TI_i	set of time periods in which facility i is available

PARAMETERS

C_p	storage capacity of product p [kg]
T_{ip}^{min}	min production time for product p in facility i [days]
T_{ip}^{max}	max production time for product p in facility i [days]
r_{ip}	production rate of product p at facility i [batches/day]
α	lead time [days]
ζ	shelf-life of product [time periods]
η_{ip}	manufacturing cost of product p at facility i [RMU/batch]
ρ	storage cost [RMU/kg]
ψ	changeover cost [RMU/changeover]
v	sales price [RMU/kg]
δ	lateness penalty [RMU/kg]
ζ	product lifetime [time periods]
π	backlog decay factor
yd_{ip}	yield conversion factor for product p in facility i [kg/batch]
D_{pt}	demand of product p at time period t [kg]

INTEGER VARIABLES

$product_{it}$	part of the chromosome containing product labels allocated at time period t to facility i
$time_{it}$	part of the chromosome containing the number of production days allocated at time period t to facility i
B_{ipt}	number of batches of product p produced at time period t in facility i

BINARY VARIABLES

Y_{ipt}	1 if product p is produced in facility i at time period t; 0 otherwise
Z_{ipt}	1 if a new campaign of product p is produced in facility i at time period t; 0 otherwise

CONTINUOUS VARIABLES

T_{ipt}	production time for product p at facility i during time period t [days]
K_{ipt}	amount of product p produced in facility i at time period t [kg]
I_{pt}	amount of product p stored at time period t [kg]
W_{pt}	amount of product p wasted in at time period t [kg]
S_{pt}	amount of product p sold at time period t [kg]
Δ_{pt}	amount of product p in backlog at time period p [kg]
$Profit$	total profit – objective function [RMU]

3.3. Problem Definition

In this section, the industrial case studies of capacity planning and scheduling of biopharmaceutical manufacture from two different literature sources are described. In case study 1, a medium-term capacity planning and scheduling problem of a multi-suite, multi-product biopharmaceutical manufacture from Lakhdar et al. (2005) is presented. In case study 2, a long-term capacity planning and scheduling problem of multi-site, multi-product bio-manufacture from Lakhdar et al. (2007) is solved.

3.3.1. Case Study 1

The objective of the planning problem presented here is to generate a yearlong production schedule that would maximise the manufacturing profits of multi-suite biopharmaceutical facility. The topology of this facility is illustrated in Figure 3.1. All relevant parameters and product demand profiles for case study 1 are listed in Tables 3.1 and 3.2, respectively. The problem statement adapted from Lakhdar et al. (2005) is as follows:

- Given:
  - Biopharmaceutical products $p = \{ p_1, p_2, p_3 \}$
  - *USP* suites $i = \{ i_1, i_2 \}$ and *DSP* suites $j = \{ j_1, j_2 \}$
  - A planning horizon of 360 days made of equal time periods $T = \{ t_1, t_2, \ldots, t_6 \}$
  - Product-dependent production rates, lead times, and production throughputs (correspondence factors)
  - *USP* and *DSP* product shelf-life, storage capacities and costs
  - Product demands, sales price and backlog penalty costs
  - Manufacturing and campaign changeover costs

  - Minimum and maximum campaign durations

- Determine:
  - Duration and sequence of campaigns
  - Production quantities along with inventory profiles
  - Product sales and late deliveries profile
- To:
  - Maximise the profitability of the schedule

Table 3.1. All relevant parameters used in case study 1.

	Product		
	p_1	**p_2**	**p_3**
USP production rate CR_p [batches / day]	0.05	0.045	0.08
USP lead time α_p [days]	30	32	22.5
USP product lifetime ζ_p [time periods]	1	1	1
USP storage capacity C_p [batches / time period]	10	10	10
USP minimum campaign length CT_p^{min} [days]	20	21	12.5
USP minimum campaign length CT_p^{max} [days]	60	60	60
DSP production rate FR_p [batches/days]	0.1	0.1	0.1
DSP lead time β_p [days]	40	42	34.5
DSP product lifetime σ_p [time periods]	3	3	3
DSP storage capacity F_p [batches / time period]	40	40	40
DSP minimum campaign length FT_p^{min} [days]	10	10	10
DSP minimum campaign length FT_p^{max} [days]	60	60	60
Production factor λ_p	1	1	1
Sales price v_p [RMU / batch]	20	20	20
Production cost η_p [RMU / batch]	2	2	2
Backlog penalty δ_p [RMU / batch]	20	20	20
Changeover cost ψ_p [RMU / changeover]	1	1	1
Waste disposal cost τ_p [RMU / batch]	5	5	5
USP storage cost ρ_p [RMU / batch]	5	5	5
DSP storage cost ω_p [RMU / batch]	1	1	1

Table 3.2. Product demand profile [batches] for case study 1.

	Time period (each period represents 60 days)					
Product	**t_1**	**t_2**	**t_3**	**t_4**	**t_5**	**t_6**
p_1	0	0	0	6	0	6
p_2	0	0	6	0	0	0
p_3	0	8	0	0	8	0

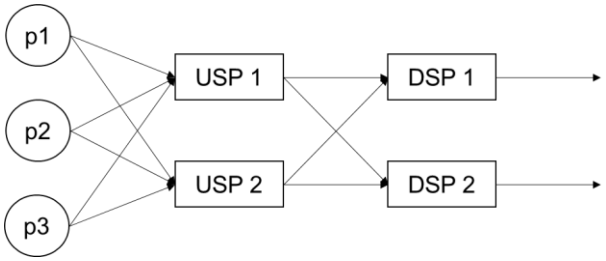


Figure 3.1. Biopharmaceutical facility topology for the case study 1.

3.3.2. Case Study 2

The goal of the planning problem presented in this case study is to generate a 15-yearlong production schedule to maximise manufacturing profits. The problem presented here is a single-objective problem adapted from Lakhdar et al. (2007). All relevant data, e.g. demand profile, parameters, are listed in Tables 3.3-3.8. The following is a brief problem statement:

- Given:
  - A network of multi-product facilities $i = \{ i_1, i_2, \ldots, i_{10} \}$
  - Biopharmaceutical products $p = \{ p_1, p_2, \ldots, p_{15} \}$
  - A planning horizon of 15 years with equal time periods $t = \{ t_1, t_2, \ldots, t_{60} \}$
  - Production rates, yields, and lead times
  - Product lifetimes and storage capacities
  - Product demands and sales prices

- Backlog decay factor
- Manufacturing, changeover, storage costs, and late delivery penalties
- Minimum and maximum campaign durations

- Determine:
  - Campaign durations and sequence of campaigns
  - Production quantities along with inventory profiles
  - Product sales and late deliveries profile
- To:
  - Maximise manufacturing profits

Table 3.3. Parameter data for case study 2.

Parameter	Value
Production lead time α [days]	14
Product lifetime ζ [time periods]	8
Sales price v [RMU / kg]	2.5
Storage cost ρ [RMU / kg]	0.01
Backlog penalty δ [RMU / kg]	0.1
Changeover cost ψ [RMU / changeover]	2
Backlog decay π	0.5

Table 3.4. Production yields yd_{ip} [kg / batch] for industrial case study 2.

	Product														
Facility	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
i_1	10	1	0	8	0	6	0	10	2	9	7	1	0	12	12
i_2	9	0	0	8	0	6	0	9	0	8	10	0	10	12	11
i_3	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0
i_4	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0
i_5	0	0	0	10	0	0	0	10	0	8	8	0	0	11	11
i_6	0	0	0	12	0	0	0	10	0	8	17	0	0	17	14
i_7	0	0	0	0	0	0	10	0	0	10	0	0	0	0	0
i_8	0	0	36	0	19	0	0	0	0	0	0	0	0	0	0
i_9	10	0	0	12	0	5	0	0	0	8	16	0	0	12	13
i_{10}	9	1	0	12	0	5	0	10	2	8	14	1	10	12	12

Table 3.5. Product demand profile [kg] for case study 2.

	Time period (each period represents 87 days)														
Product	t_4	t_8	t_{12}	t_{16}	t_{20}	t_{24}	t_{28}	t_{32}	t_{36}	t_{40}	t_{44}	t_{48}	t_{52}	t_{56}	t_{60}
p_1	21	32	18	28	61	104	153	156	164	163	161	162	162	163	165
p_2	6	5	4	4	4	3	3	3	3	3	3	3	2	2	2
p_3	12	43	38	5	22	52	97	132	133	135	137	118	109	100	90
p_4	583	628	655	687	758	921	989	941	993	649	621	573	521	468	421
p_5	12	12	11	10	9	7	6	5	4	3	2	2	2	2	3
p_6	211	200	245	246	257	266	284	274	226	180	166	151	137	123	110
p_7	4	5	5	7	6	5	8	9	8	9	7	7	6	5	5
p_8	5	5	5	7	6	5	8	9	8	9	7	7	6	5	5
p_9	15	15	15	13	12	9	8	6	5	4	3	3	2	2	2
p_{10}	72	99	104	102	111	120	130	139	188	120	106	93	81	69	58
p_{11}	552	615	699	737	743	733	684	572	518	471	424	381	342	307	274
p_{12}	5	5	5	7	6	5	8	9	8	9	7	7	6	5	5
p_{13}	211	252	290	298	286	216	169	153	150	145	110	100	93	84	102
p_{14}	2	2	4	3	3	3	16	11	13	16	16	16	16	17	17
p_{15}	4	4	5	6	16	11	24	32	37	40	41	42	42	43	44

Table 3.6. Production rates r_{ip} [kg / day] for case study 2.

	Product														
Facility	**p_1**	**p_2**	**p_3**	**p_4**	**p_5**	**p_6**	**p_7**	**p_8**	**p_9**	**p_{10}**	**p_{11}**	**p_{12}**	**p_{13}**	**p_{14}**	**p_{15}**
i_1	0.35	0.39	0	0.45	0	0.29	0	0.35	0.25	0.39	0.41	0.39	0	0.12	0.35
i_2	0.6	0	0	0.61	0	0.6	0	0.6	0	0.43	0.56	0	0.6	0.6	0.6
i_3	0	0	0	0	0	0	0	0	0	0	0	0	0.23	0	0
i_4	0	0	0	0.12	0	0	0	0	0	0	0	0	0	0	0
i_5	0	0	0	0.45	0	0	0	0.45	0	0.45	0.45	0	0	0.45	0.45
i_6	0	0	0	0.45	0	0	0	0.45	0	0.45	0.45	0	0	0.45	0.45
i_7	0	0	0	0	0	0	0.45	0	0	0.45	0	0	0	0	0
i_8	0	0	0.58	0	0.45	0	0	0	0	0	0	0	0	0	0
i_9	0.45	0	0	0.45	0	0.45	0	0	0	0.45	0.45	0	0	0.45	0.49
i_{10}	0.45	0.45	0	0.45	0	0.45	0	0.45	0.45	0.45	0.49	0.45	0.45	0.45	0.45

Table 3.7. Production costs η_{ip} [RMU / kg] for case study 2.

	Product														
Facility	**p_1**	**p_2**	**p_3**	**p_4**	**p_5**	**p_6**	**p_7**	**p_8**	**p_9**	**p_{10}**	**p_{11}**	**p_{12}**	**p_{13}**	**p_{14}**	**p_{15}**
i_1	1	1	0	10	0	3	0	1	1	1	3	1	0	1	1
i_2	10	0	0	5	0	2	0	5	0	10	2	0	2	5	2
i_3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
i_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
i_5	0	0	0	20	0	0	0	20	0	20	20	0	0	5	20
i_6	0	0	0	10	0	0	0	10	0	10	10	0	0	1	10
i_7	0	0	0	0	0	0	10	0	0	10	0	0	0	0	0
i_8	0	0	1	0	5	0	0	0	0	0	0	0	0	0	0
i_9	10	0	0	10	0	10	0	0	0	10	8	0	0	1	10
i_{10}	15	15	0	15	0	15	0	15	15	15	15	15	15	15	15

Table 3.8. Facility capability PI_i [boolean value] for case study 2.

	Product														
Facility	**p_1**	**p_2**	**p_3**	**p_4**	**p_5**	**p_6**	**p_7**	**p_8**	**p_9**	**p_{10}**	**p_{11}**	**p_{12}**	**p_{13}**	**p_{14}**	**p_{15}**
i_1	1	1	0	1	0	1	0	1	1	1	1	1	0	1	1
i_2	1	0	0	1	0	1	0	1	0	1	1	0	1	1	1
i_3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
i_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
i_5	0	0	0	1	0	0	0	1	0	1	1	0	0	1	1
i_6	0	0	0	1	0	0	0	1	0	1	1	0	0	1	1
i_7	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
i_8	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
i_9	1	0	0	1	0	1	0	0	0	1	1	0	0	1	1
i_{10}	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1

In their paper, Lakhdar et al. (2007) stated that the presented MILP model was an extension of the one already discussed in case study 1 described earlier. The core mathematical formulation for the single-objective problem remained mostly the same with the only most noticeable change being the lack of explicit model of separation between *USP* and *DSP* suites. Nevertheless, the complexity of the problem in case study 2 is much higher compared to case study 1 due to a greater number of products, facilities, and time periods (refer to Table 3.9 for a comparison). A 15-year time horizon is assumed comprising 60 time periods. Each individual time period *t* is 87 days long compared to 60 in case study 1. There are 15 products that need to be

allocated to 10 facilities. Additional subsets are introduced to define facility capability and availability: PI_i, the set of products that can be manufactured in facility i (Table 3.8), and TI_i, the set of time periods during which facility i is available for use. All facilities are assumed to be available throughout the time horizon, apart from facility i_6 which is unavailable until time period t_5, and facility i_9 which is unavailable until time period t_{41}. Minimum T_{ip}^{min} and maximum T_{ip}^{max} campaign durations are assumed to be 0 and 87, respectively. Production yield yd_{ip} (Table 3.4), rate r_{ip} (Table 3.6), and cost η_{ip} (Table 3.7) of each product p depend on facility i it is being manufactured in.

Table 3.9. The comparison of MILP model complexity between case study 1 and 2.

	Case Study 1	**Case Study 2**
Single equations	535	19,430
Single variables	457	25,018
Discrete variables	252	9,382
Non-zero elements	1,750	72,244

3.4. Methods

The implementation of mathematical models using algebraic modelling systems such as GAMS is very different compared to general-purpose programming languages such as C++. GAMS allows the mathematical models to be implemented in a way that is similar to their mathematical notation, while the general-purpose programming languages require an explicit definition of every expression. Another critical challenge of developing an efficient GA-based approach was identifying the smallest number of independent variables so as to maintain the dimensionality of the problem low and the shortest sequence of steps needed to evaluate the candidate solutions for the case studies to ensure good performance.

In this section, the structure of the proposed GA-based approach and the steps of the algorithms that captured capacity planning objectives for multiple products across

multiple suites and facilities are outlined. Most of the GA methods is explained in the methods section for case study 1. In case study 2, the focus is on the rolling horizon strategy taken to improve the performance of the standard GA for solving the long-term capacity planning problem. The relative complexity of the optimisation problems is illustrated by the summary of the MILP model statistics shown in Table 3.1. The MILP models were recreated in GAMS 23.9.5 and solved with a CPLEX 12.4.0.1 solver. GA and PSO algorithms were implemented using C++ programming language and compiled using the Microsoft Visual C++ Compiler v14 (MSCV). The mathematical models are summarised in Appendices C and D; however, the reader is advised to refer to the original papers for a more in-depth explanation. Both case studies were performed on an Intel i5-6500 based Windows 10 system with 16GB of RAM.

3.4.1. GA Parameter Tuning

The process of identifying the optimal parameters for an optimisation algorithm or a machine learning one is usually costly, involves the search of a large, possibly infinite, space of candidate parameter sets, and may not guarantee optimality (Camilleri et al., 2014). A simple PSO algorithm is implemented as a meta-optimiser to automatically tune the crossover and mutation parameter values in both case studies of this chapter (and throughout this thesis). Each particle, i.e. a potential solution, is initiated with randomised position and velocity vectors. The particle's position in a decision space is defined by its position vector comprising the parameter values of the crossover and mutation. The particle's velocity is the speed and direction at which the particle is traversing the decision space during each *epoch*. The fitness of each particle is assessed by running the GA using the parameter values encoded in the position vector for a specified number of independent algorithm runs with a fixed population size measuring the average of the best objective function values achieved at the end

of each run. The concept of the meta-optimisation is illustrated in Figure 3.2 while Algorithm 3.1 lists a pseudocode for it. The parameter values of PSO algorithm (Table 3.2) were chosen based on the studies performed by Eberhart and Shi (2000) and Trelea (2003). Meta-optimisation is also applied in other chapters of this thesis (mainly Chapter 4) to automatically set the parameters of the GAs.

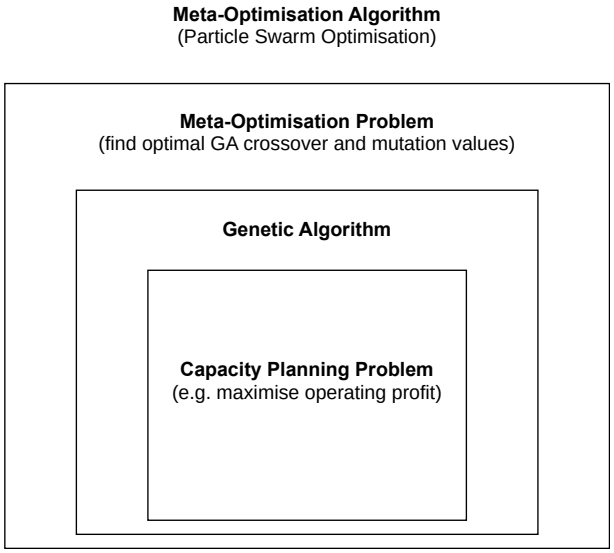


Figure 3.2. The meta-optimisation approach. Adapted from Camilleri et al. (2014).

Table 3.10. Meta-optimisation parameters used in case study 1 and 2 to find the optimal crossover and mutation parameter values for the GA.

	Case Study 1	Case Study 2
PSO swarm size[1]	20	
Number of PSO epochs[2]	200	
PSO inertia weight w[3]	0.729	
PSO local weight c_1[4]	1.494	
PSO global weight c_2[5]	1.494	
Number of GA runs	100	50
GA population size	100	200

[1] The number of candidate solutions, i.e. particles.
[2] An equivalent of generations in the GA.
[3] Determines how much of the original velocity is retained.
[4] Determines how much the personal best position of a particle affects the global search process. Larger local weights drive the particles towards their own personal bests thus breaking the swarm apart.
[5] Determines how much the global best position affects the global search process. Larger global weights tend to keep the swarm tighter turning it into one large hill-climber.

Algorithm 3.1. PSO-based meta-optimisation of the GA.

```
1  swam = ∅
2  epoch = 0                                                    ▻ epochs counter
3  best⃗ = <>                                                   ▻ placeholder for best position vector
4  for swarm_size times
5     x⃗ = A position vector with random values from 0.0 to 1.0 for each GA parameter
6     v⃗ = A velocity vector with random values from 0.0 to 1.0 for each GA parameter
7     particle = { x⃗, v⃗ }
8     swarm = swarm U { particle }
9  end for
10 while epoch < epochs
11    for each particle x⃗ in swarm
12       particle's fitness = n-run performance of the GA using parameter values encoded in x⃗
13       if best⃗ == <> or particle's fitness > best⃗ fitness
14          best⃗ = x⃗
15       end if
16    end for
17    for each particle x⃗ and v⃗ in swarm
18       x⃗+ = previous fittest location of the current particle
19       for each dimension i                  ▻ update particle's position x⃗ and velocity v⃗ vectors
20          r1 = random number from 0.0 to 1.0 inclusive
21          r2 = random number from 0.0 to 1.0 inclusive
22          vi = wvi + c1r1(xi+ – xi) + c2r2(besti – xi)
23          xi = xi + vi
24          Ensure xi is in 0.0-1.0 range     ▻ can be either reinitialised or set to the closest bound
25       end for
26    end for
27    epoch += 1
28 end while
29 return best⃗
```

3.4.2. Case Study 1

3.4.2.1. Chromosome Structure

In case study 1, the GA-based approach uses a semi-direct representation, i.e. only the *USP* part of the schedule is encoded. Each chromosome is an | *i* |-*by*-| *t* | array of tuples where *i* is a set of *USP* suites, and *t* is a set of discrete-time periods (illustrated in Figure 3.3). Each tuple comprises a product label *p* and production time CT_{ipt} in *USP* suite *i* at time period *t* measured in days. Both variables are randomly generated at the beginning of the GA during the initial population generation. The variable CT_{ipt} is generated randomly within the minimum CT_{ip}^{min} and maximum CT_{ip}^{max} production

time range. The *DSP* part of the schedule is constructed during fitness evaluation using the *USP* variables.

		t_1	t_2	…	t_n
USP suites	i_1	(p, CT_{ipt})	(p, CT_{ipt})	…	(p, CT_{ipt})
	i_2	(p, CT_{ipt})	(p, CT_{ipt})	…	(p, CT_{ipt})
	…	…	…	…	…
	i_n	(p, CT_{ipt})	(p, CT_{ipt})	…	(p, CT_{ipt})

Figure 3.3. Chromosome encoding strategy for case study 1. Each (p, CT_{ipt}) pair represents a gene encoding which product p and how many days CT_{ipt} have been allocated to USP suite i at a time period t.

3.4.2.2. Genetic Algorithm

The GA comprises the following steps: fitness evaluation, tournament selection, crossover, mutation, and replacement. In case study 1, chromosomes for crossover and mutation are selected using a binary tournament with replacement strategy which favours individuals with a higher objective function value, i.e. schedules with a larger profit value. A uniform crossover operator with a rate of *pC* is used to exchange the tuples between the chromosomes. Each tuple is also mutated with a rate *pM* to avoid premature convergence and improve the quality of the final solution. During mutation, the product label is changed by replacing it with a different random value from the set of available products *P.* The length of production is varied by adding or subtracting a random number of days, ensuring the allocated campaign time is within the constrained range, CT_{ip}^{min} and CT_{ip}^{max}. If the length of production after mutation happens to fall outside of the constrained range, it is set to the value of the closest bound. In both case studies, the GAs are augmented with elitism (the term was originally coined by De Jong (1988)) which is a highly exploitative method of preserving the fittest chromosomes from the previous population (Luke, 2013). In case study 1, a single best chromosome is re-inserted into the population whenever

it is lost. Finally, the GA is set to terminate early if the fitness of the best individual has not improved for 100 consecutive generations.

3.4.2.3. Fitness Evaluation

In both case studies, the fitness evaluation procedures contain algorithmic adaptations of the MILP models (Lakhdar et al., 2005; Lakhdar et al., 2007) of multi-product biopharmaceutical manufacture. In case study 1, the fitness evaluation procedure generates a complete production schedule (fills the *DSP* part) and estimates the values of binary and continuous variables, e.g. Z_{ipt}, B_{ipt}, CI_{pt}, which are used in the objective function to calculate the profitability of the schedule (Equation 3.1). The pseudo algorithm of the fitness evaluation procedure for case study 1 is presented in Algorithm 3.2.

Algorithm 3.2. Pseudocode for fitness evaluation in case study 1

```
1  for each time period t
2     for each upstream suite i
3        p = products_it
4        CT_ipt = time_it
5        Z_ipt = 1 – (t > 0 and p == products_i,t-1)
6        B_ipt = Z_ipt + CR_p(CT_ipt – α_p Z_ipt)
7        CI_pt = CI_pt + B_ipt
8     end for
9     for each product p
10       if t > ζ_p
11          CW_pt = CI_p,t-ζp-1 – ( ∑_j ∑_θ=t-ζp^ζp B_jpθ + ∑_θ=t-ζp^ζp CW_pθ )
12       end if
13       CI_pt = CI_pt + CI_p,t-1 – CW_pt
14       if CI_pt > C_p
15          CW_pt = CW_pt + CI_pt – C_p
16          CI_pt = C_p
17       end if
18       for each downstream suite j
19          if products_jt == 0
20             B_jpt = λ_p CI_pt
21             Z_jpt = 1 – (t > 0 and p == products_j,t-1)
22             while ( FT_ipt = β_p Z_jpt + (B_jpt - Z_jpt)/FR_p ) > FT_p^max
23                B_jpt = B_jpt – 1
```

Algorithm 3.2. (continued) Pseudocode for fitness evaluation in case study 1

```
24              end while
25              CI_pt = CI_pt − B_jpt / λ_p
26              FI_pt = FI_pt + B_jpt
27              products_jt = p
28              time_jt = FT_jpt
29            end if
30          end for
31          if t > σ_p
32            FW_pt = FI_p,t-σ_p^-1 − (∑_θ=t-σ_p^σ_p S_pθ + ∑_θ=t-σ_p^σ_p FW_pθ )
33          end if
34          FI_pt = FI_pt + FI_p,t-1 − FW_pt
35          if FI_pt > F_p
36            FW_pt = FW_pt + FI_pt − F_p
37            FI_pt = F_p
38          end if
39          if D_pt > 0
40            if D_pt ≤ FI_pt
41              S_pt = D_pt
42              FI_pt = FI_pt − S_pt
43            else
44              S_pt = FI_pt
45              FI_pt = 0
46              Δ_pt = D_pt − S_pt
47            end if
48          end if
49          if Δ_p,t-1 > 0
50            if Δ_p,t-1 ≤ FI_pt
51              S_pt = S_pt + Δ_p,t-1
52              FI_pt = FI_pt − Δ_p,t-1
53            else
54              S_pt = S_pt + FI_pt
55              FI_pt = 0
56              Δ_pt = Δ_pt + Δ_p,t-1 − S_pt
57            end if
58          end if
59        end for
60      end for
```

In Algorithm 3.2, Lines 3 and 4 retrieve the product label *p* and the number of production days allocated to *USP* suite *i* at time period *t*, CT_{ipt}, from the chromosome which is an | *i* |-*by*-| *t* | array. Lines 5 and 6 calculate the number of changeovers and batches produced in *USP* suite *i* at time period *t*. In Line 5, the value of the changeover variable Z_{ipt} will be equal to 1 if and only if product *p* has not been produced in *USP*

suite *i* at a previous time period *t – 1*. Line 7 accumulates the production from all *USP* suites. Lines 10-12 calculate the amount of product *p* wasted in *USP* suites at time period *t* which is equal to the number of batches left unprocessed from ζ_p periods ago. The amount of *USP* inventory of product *p* at time period *t* is calculated in Line 13 by adding the cumulative value obtained in Line 7 from time period *t – 1* and subtracting the amount of waste CW_{pt}. Lines 14-17 ensure that the *USP* inventory level CI_{pt} does not exceed the storage limit C_p. Any excess inventory of product *p* during time period *t* is calculated as waste CW_{pt} (Line 15).

Line 19 ensures that the assignment of product *p* to *DSP* suite *j* at time period *t* is performed only once. Line 20 calculates how many batches will be produced in a DSP suite *j* at time period *t*. This is performed by multiplying the *USP* inventory value CI_{pt} by the production correspondence factor λ_p which specifies the respective throughputs from *USP* and *DSP* suites. For example, a factor of 0.5 signifies that for every two *USP* batches one *DSP* batch is produced. Line 21 evaluates the number of changeovers in *DSP* suites similarly to Line 5. Line 22 estimates the campaign duration FT_{jpt} of product *p* at *DSP* suite *j* during time period *t.* It also checks whether the *DSP* campaign length does not exceed the allowed maximum FT_p^{max}. If it does, the value of variable B_{jpt} is iteratively decremented until the production time FT_{jpt} is below or equal to FT_p^{max} (Line 23). Line 25 updates the value of *USP* inventory of product *p* at time period *t* by subtracting the number of batches that are processed in *DSP* suite *j.* Line 26 accumulates the production from all *DSP* suites. Lines 27 and 28 assign the product *p* and *DSP* production time FT_{jpt} to the *DSP* part of the chromosome.

The amount of *DSP* waste FW_{pt} and inventory levels FI_{pt} of final product *p* at time period *t* are calculated in Lines 31-38 similarly to Lines 10-17. In Lines 39-48, if there

is a demand D_{pt} for product p at time period t, then the amount of product sold S_{pt} is calculated based on the number of batches stored in *DSP* inventory FI_{pt}. If there are more batches in storage than there are in demand (Line 40), the variable S_{pt} will be equal to the value of demand (Line 41). Otherwise, Line 44 assigns the value of *DSP* storage FI_{pt} to S_{pt}, and the backlog is recorded using variable Δ_{pt} for that time period in Line 46. If the inventory allows it (Line 50), the backlog from a previous time period $\Delta_{p,t-1}$ is sold in Line 45. Otherwise, it is accumulated in Line 51.

The fitness of each chromosome is equal to the profit achieved by the schedule which is calculated with the same objective function (Equation 3.1) as presented by Lakhdar et al. (2005) using the aforementioned binary and continuous variables. The objective function value is equal to the difference between the total sales $\sum_p \sum_t \nu_p S_{pt}$ and the total costs of manufacturing $\sum_p \sum_t (\sum_i \eta_p B_{ipt} + \sum_j \eta_p B_{jpt})$, product changeovers $\sum_p \sum_t (\sum_i \psi_p Z_{ipt} + \sum_j \psi_p Z_{jpt})$, intermediate and final product storage $\sum_p \sum_t (\rho_p CI_{pt} - \omega_p FI_{pt})$, late deliveries $\sum_p \sum_t \delta_p \Delta_{pt}$, and waste disposal $\sum_p \sum_t (\tau_p \mathrm{CW}_{pt} - \tau_p \mathrm{FW}_{pt})$.

$$Profit = \sum_p \sum_t (\nu_p S_{pt} - \sum_i \eta_p B_{ipt} - \sum_i \psi_p Z_{ipt} - \sum_j \eta_p B_{jpt} - \sum_j \psi_p Z_{jpt} - \rho_p CI_{pt} - \omega_p FI_{pt} - \delta_p \Delta_{pt} - \tau_p CW_{pt} - \tau_p FW_{pt}) \qquad \textit{Equation 3.1.}$$

3.4.3. Case Study 2

3.4.3.1. Chromosome Structure

The increased complexity of the planning problem in case study 2 presented a challenge for the GA-based approach. Encoding the chromosomes as full-scale | *i* |-*by*-| *t* | arrays was found to be computationally costly. A rolling time horizon method was taken to explore the large search space in a more efficient manner by dividing the 15-yearlong planning problem into 15 equal sub-problems solved consecutively. In order to accomplish this, each chromosome encoded a sub-problem as an | *i* |-*by*-| τ | array of product *p* and the length of production T_{ipt} values where $\tau \subset t$ and | τ | = 4. τ represents the extent of the rolling time horizon, i.e. a dynamic subset of 4 time periods which correspond to the timeline of the sub-problem being solved. For example, $\tau = \{ t_1, t_2, t_3, t_4 \}$ and $\tau = \{ t_{57}, t_{58}, t_{59}, t_{60} \}$ contain the time periods for the first and last sub-problems, respectively. The best solution from each sub-problem is stored in the final, full-scale | *i* |-*by*-| *t* | solution, before proceeding to solve the following sub-problem. The values of the variables corresponding to the best solution such as the number of batches B_{ipt} of product *p* produced in each facility *i* during time period *t* are fixed so they would not need to be recalculated for the next sub-problem. To distinguish the rolling time horizon approach-based GA from the standard one , which uses a direct full-scale encoding strategy, it will be referred to it as the *dynamic GA.*

3.4.3.2. Genetic Algorithm

This section explains the dynamic GA procedure. Algorithm 3.3 lists the pseudocode for the dynamic GA. Figure 3.4 illustrates the concept of it.

Algorithm 3.3. Pseudocode for the dynamic GA applied in case study 2.

```
1  for each subproblem
2     parents = ∅
3     gen = 0                                          ▻ subproblem generation counter
4     num_restarts = 0                                 ▻ tracks the number of times the GA was restarted
5     subproblem_best = □                              ▻ placeholder for the best solution to the current subproblem
6     Generate new parent population
7     while gen < max_gens
8        if gen ≥ x and subproblem_best == subproblem_best from x generations ago
9           if num_restarts < desired number of GA restarts
10             Generate new parent population
11             num_restarts += 1
12          else
13             break
14          end if
15       end if
16       for each parent in parents
17          EvaluateFitness(parent)
18          if subproblem_best = □ or fitness of parent > fitness of subproblem_best
19             subproblem_best = parent
20          end if
21       end for
22       offspring = { top n of the fittest individuals in parents, breaking ties at random }
23       for (|parents| – | offspring |) / 2 times
24          parent_a = BinaryTournament(parents)
25          parent_b = BinaryTournament(parents)
26          offspring_a, offspring_b = Crossover(parent_a, parent_b)
27          offspring = offspring U { Mutate(offspring_a), Mutate(offspring_b) }
28       end if
29       parents = offspring
30       gen += 1
31    end while
32    Extend the full-scale solution with subproblem_best   ▻fix solved variables
33 end for
```

A new parent population is generated for every sub-problem with the values of product *p* for each facility *i* selected randomly from the set of allowable products for that facility, PI_i, making sure the facility *i* is also available for use at time period $t \in TI_i$. A product label with a value of 0 is also included in the set to denote facility *i* idling at time period *t*. The parent population of *gen + 1* is made up of the top 5% of the fittest individuals from the previous parent population and the offspring (recombined parents) (see Lines 22-29, Algorithm 3.3). A uniform crossover operator with a probability *pC* is used to create two offspring from two parent chromosomes. The product label *p* and production time encoded in each chromosome are mutated independently with

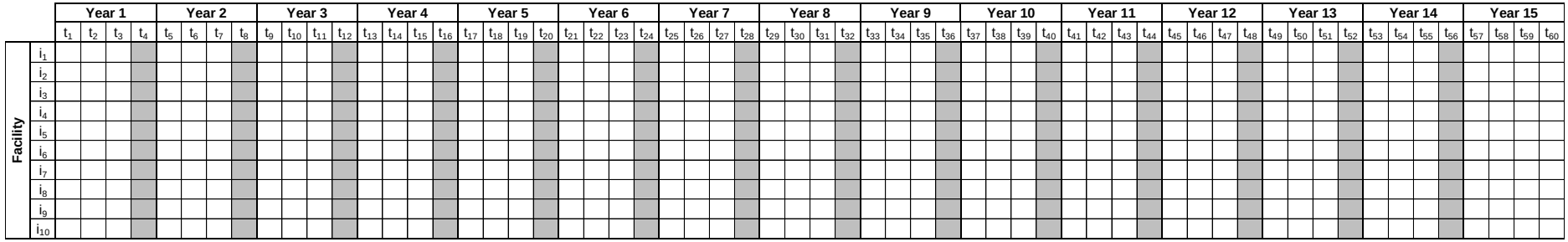


Figure 3.4. An illustration of how the long-term capacity planning problem from case study 2 can be divided into smaller sub-problems. The full solution and each sub-problem are | i |-by-| t | and | I |-by-| τ | arrays respectively. When | τ | = 4, the sub-problems overlap with one another on the parts that are shaded in grey. For example, once the first sub-problem is solved { t_1, t_2, t_3, t_4 }, some of the fixed binary and continuous variables from time period t_4 will be used to estimate the variable values over time period t_5 for the second sub-problem { t_5, t_6, t_7, t_8 }. The dynamic GA generates a solution to the full-scale problem by solving the sub-problems in a chronological order and concatenating the best solutions from each one.

probabilities *pMutP* and *pMutT*, respectively. Provided that the facility *i* is available for use at time period *t*, the value of product label *p* is mutated by assigning 0 or a random value from the subset PI_i (products that can be manufactured in facility *i*). Production time is mutated by adding or subtracting a random number of days, similarly to the mutation procedure employed in case study 1. A completely new parent population is generated when the best fitness value remains unchanged for a specified number of consecutive generations (Lines 8-16, Algorithm 3.3). When this repeats, the GA stops solving the sub-problem (Line 13, Algorithm 3.3) and extends the full-scale solution with the best solution to the most recent sub-problem (Line 32, algorithm 3.3).

3.4.3.3. Fitness Evaluation

The fitness evaluation procedure in the dynamic GA of case study 2 is very similar to that of case study 1. In Algorithm 3.4, the variable ξ is used to iterate through the values of the $|\, i \,|$-*by*-$|\, \tau \,|$ array encoded by each chromosome. The product label *p* and production time T_{ipt} are retrieved from the chromosomes in Lines 4 and 5. The value of the binary changeover variable Z_{ipt} is set to 1 in Line 6 if variable $B_{ip,t-1}$, the number of batches of product *p* produced in facility *i* in the previous time period slot, is 0. The value of the number of batches variable B_{ipt} during time period *t* is calculated in Line 7 and converted into kilograms K_{ipt} using the yield conversion factor yd_{ip} in Line 8. The value of yd_{ip} depends on the facility *i* which the product *p* is being manufactured in. Line 9 accumulates the value of K_{ipt} into the variable I_{pt} – the amount of product *p* in kilograms stored at time period t. The amount of product waste W_{pt} is estimated in Lines 13-15. The value of this variable is equal to the amount of product *p* that was not sold and remained in storage for more than ζ time periods. The rest of the pseudocode logic in Algorithm 3.4, i.e. from Line 17 and onwards is nearly identical to Lines 39-58 in Algorithm 3.2 (fitness evaluation for case study 1). The only notable

differences are the lack of storage capacity constraints and the addition of backlog decay factor π which diminishes the importance of the backlogged orders over time.

Algorithm 3.4. Pseudocode for fitness evaluation in case study 2.

```
1  ξ = 0
2  for each time period t in subproblem                    ▻ or for ξ from 0 to | τ |
3     for each facility i
4        p = products_iξ
5        T_ipt = time_iξ
6        Z_ipt = 1 – (t > 0 and B_ip,t-1 == 0)
7        B_ipt = Z_ipt + r_ip(T_ipt – αZ_ipt)
8        K_ipt = B_ipt yd_ip
9        I_pt = I_pt + K_ipt
10    end for
11    ξ = ξ + 1
12    for each product p
13       if t > ζ
14          W_pt = I_p,t-ζ-1 – ( ∑_θ=t-ζ^ζ S_pθ + ∑_θ=t-ζ^ζ W_pθ )
15       end if
16       I_pt = I_pt + I_ip,t-1 – W_pt
17       if D_pt > 0
18          if D_pt ≤ I_pt
19             S_pt = D_pt
20             I_pt = I_pt – S_pt
21          else
22             S_pt = I_pt
23             I_pt = 0
24             Δ_pt = D_pt – S_pt
25          end if
26       end if
27       if Δ_p,t-1 ≥ 0
28          if Δ_p,t-1 ≤ I_pt
29             S_pt = S_pt + Δ_p,t-1
30             I_pt = I_pt – Δ_p,t-1
31          else
32             S_pt = S_pt + I_pt
33             I_pt = 0
34             Δ_pt = Δ_pt + πΔ_p,t-1 – S_pt
35          end if
36       end if
37    end for
38 end for
```

The fitness of each chromosome is evaluated using the objective function of profit maximisation (Equation 3.2) defined by Lakhdar et al. (2007). The objective function value is equal to the difference between the total sales $\sum_p \sum_{t\in TI_i} vS_{pt}$ and the total operating costs which include the costs of manufacturing and changeovers $\sum_p \sum_{t\in TI_i} \sum_{i\in IP_i}(\eta_{ip}B_{ipt} + \psi Z_{ipt})$, storage $\sum_p \sum_{t\in TI_i} \rho I_{pt}$, and late deliveries $\sum_p \sum_{t\in TI_i} \delta \Delta_{pt}$.

$$Profit = \sum_p \sum_{t\in TI_i}(vS_{pt} - \rho I_{pt} - \delta\Delta_{pt} - \sum_{i\in IP_i}(\eta_{ip}B_{ipt} + \psi Z_{ipt}))$$ *Equation 3.2.*

3.5. Results

In this section, the results to the case studies of capacity planning and scheduling of biopharmaceutical manufacture from the literature are presented. In case study 1, the problem consists of a multi-suite facility, with 2 *USP { i_1, i_2 }* and 2 *DSP { j_1, j_2 }* suites to produce 3 products *{ p_1, p_2, p_3 }* with multiple intermediate demand dates due over a 360-day long production time horizon. The horizon is discretised into 6 time periods *{ t_1, t_2, …, t_6 }* of 60 days. In case study 2, the problem consists of 10 facilities *{ i_1, i_2, …, i_{10} }* with different manufacturing capabilities PI_i (subset of facilities capable of producing product *p*) and availability TI_i (subset of facilities available at time period *t*) to produce 15 products *{ p_1, p_2, …, p_{15} }* due annually over a 15-yearlong production time horizon. The horizon consists of 60 discrete time periods *{ t_1, t_2, …, t_{60} }* of 87 days.

The GAs discussed in the previous sections for case study 1 and case study 2 are used to solve the respective scheduling problems, and the results are compared with the recreated MILP models in Tables 3.11 and 3.13. A comparison between the production schedules generated using MILP and a GA is also provided in Figures 3.5 and 3.6.

3.5.1. Case Study 1

The proposed GA developed in this chapter was first applied to case study 1 on medium-term capacity planning for a single-site, multi-suite, multi-product biopharmaceutical facility. Initially, a MILP model was developed for the problem as a benchmark for comparison with the GA performance. In their original MILP work, Lakhdar et al. (2005) reported an objective function value of 487 relative monetary units (RMU) with a 5% optimality gap for this problem. The margin of optimality (also known as a relative optimality gap) is defined as the relative distance between the relaxed MILP solution and the current best integer MILP solution (Brooke et al., 1998). In other words, it is the relative difference between the "best estimate" solution and "the best integer" solution that satisfies all integer requirements/constraints. Lakhdar et al. (2005) reported that it took 16 seconds to solve the optimisation problem. Using the reproduced MILP model an objective function value of 490 RMU was achieved with 0% optimality gap indicating a global optimum.

Table 3.11. Case study 1 results and model statistics for MILP and GA models.

	MILP	**GA[a]**	**GA[b]**
Max obj. function value	490	490[1]	490[1]
Solution time (s)	0.22	0.07[2]	0.07[2]
Optimality gap	0%	0%[3]	0%[3]
Avg. obj. function value[4]	-	490 ± 0	489 ± 5
Population size	-	100	
Crossover rate, *pC*[5]	-	0.710	
Mutation rate, *pM*[5]	-	0.070	
Termination[6]	-	100	

[a] Results obtained using the same random number generator seed from the meta-optimisation.
[b] Results obtained using a different random number generator seed.
[1] Max objective function value obtained from 100 independent GA runs
[2] An average solution time of a single GA run
[3] An optimality estimate relative to the global optimal obtained using the recreated MILP model
[4] Mean objective function value of 100 independent GA runs (mean ± 1 standard deviation).
[5] The parameter values were selected using the PSO algorithm.
[6] Each run was terminated when the fitness had not improved for 100 generations or maximum generation limit (1000) had been reached.

In contrast to the mathematical programming approaches, such as MILP, GA is not guaranteed to converge on the same value every time it is run. As a result, the search process for the optimal value(s) is typically performed by running the GA for a number of independent runs (generating a new population for each one). Literature suggests values in the range of 20 and 50 runs, e.g. Taherdangkoo et al. (2013), Allmendinger et al. (2014). In case study 1, because of the fast execution speeds of the GA, i.e. less than a second per single run, the number of runs was set to 100. Each run was terminated when the fitness had not improved for 100 generations or maximum generation limit (1000) had been reached.

		Time periods (t_n = 60 days)					
		t_1	t_2	t_3	t_4	t_5	t_6
Suites	USP 1	4 (60)	4 (50)	2 (50)	3 (60)	2 (40)	3 (60)
	USP 2	2 (54)	2 (44)	2 (44)	4 (60)	4 (50)	2 (50)
	DSP 1	3 (55)	5 (50)	2 (50)	3 (30)	2 (20)	5 (50)
	DSP 2	2 (52)	2 (20)	2 (20)	3 (55)	5 (50)	

Product 1 Product 2 Product 3

Figure 3.5. Production schedule for case study 1 with an objective function value of 490 RMU and 0% optimality gap. Both the MILP model and the proposed GA generated the same schedule. The first number in each cell denotes the number of batches produced which is followed by the production time [days] in brackets. The shading of the box indicates which product is being manufactured.

The aforementioned PSO-based meta-optimisation approach was used to tune the crossover and mutation parameter values, *pC* and *pM*. Using this approach, the optimal values of crossover rate (*pC = 0.710*) and mutation rate (*pM = 0.070*) were identified, and the GA achieved the global optimum of 490 RMU for 100 consecutive, independent algorithm runs. The GA also generated a production schedule with the product allocation pattern identical to the one from the recreated MILP model (Figure 3.5). The average solution time of the GA was 0.07 s. In contrast, MILP took an average of 0.22 s to find the global optimum (even though MILP is a deterministic

technique, the running time can be affected by the background processes thus the MILP model was run 10 times to obtain a more accurate estimate).

Given the fast performance of the proposed GA-based method and the optimality of the results, it can be considered as a viable alternative for addressing medium-term capacity planning and scheduling problems similar in structure and complexity to case study 1.

3.5.2. Case Study 2

Having tackled medium-term, single-site facility scheduling, the GA was then extended to address long-term planning across multi-site, multi-product biopharmaceutical manufacturing facilities in case study 2. To set the benchmark for the GA, the recreated single-objective MILP model was used to achieve an objective function value of 66,360 RMU with a 0% optimality gap for this problem. It took approximately 16.7 min to find the global optimum. With the optimality gap increased to 1%, the MILP model achieved an objective function value of 65,940 RMU in 8.77 s.

As discussed earlier, two versions of a GA (*standard* and *dynamic*) were applied to solve the long-term capacity planning problem presented in case study 2. Using the *standard* version, each chromosome encoded the full-scale problem as an $| i |$-*by*-$| t |$ array (where $| i | = 10$ and $| t | = 60$), and the GA was set to terminate after 1000 generations had elapsed. In the *dynamic* version, a rolling time horizon approach was utilised to break down the full-scale 15-yearlong scheduling problem into 15 sub-problems. Each chromosome encoded only a part of the full schedule as an $| i |$-*by*-$| \tau |$ array (where $\tau \in t$ and $| \tau | = 4$) corresponding to the sub-problem being solved. Both GA versions were run 50 times. The crossover, mutation, and elitism operators were identical in both *standard* and *dynamic* versions.

The PSO-based meta-optimisation was applied to tune both the *standard* and *dynamic* GAs to ensure a fair comparison of the two versions. The *dynamic* GA was restarted once the fitness value remained unchanged for a set number of consecutive generations defined by a termination criterion. In an attempt to achieve a higher objective function value using the *dynamic* GA, different population sizes (100, 200, 300) and termination criteria (25, 50, 75) were tested (see Table 3.12).

Table 3.12. Case study 2 results and model statistics for the dynamic GA model using different population sizes and termination criteria.

Avg. obj. function value[1]	Max obj. function value [2]	Avg. solution time	Population size	Termination criterion[3]
65,399 ± 131	65,653	3.91 s	100	25
65,518 ± 144	65,799	6.11 s	100	50
65,543 ± 144	65,818	8.30 s	100	75
65,652 ± 112	**65,849**	**8.09 s**	**200**	**25**
65,755 ± 105	65,934	12.87 s	200	50
65,797 ± 92	65,987	17.20 s	200	75
65,806 ± 66	65,921	12.66 s	300	25
65,855 ± 86	65,997	19.86 s	300	50
65,883 ± 92	66,068	26,85 s	300	75

[1] Average of best objective function values from 50 independent GA runs (mean ± 1 standard deviation)

[2] Max objective function value obtained from 50 independent GA runs.

[3] If the best objective function value remained unchanged for a given number of consecutive generations, the GA is restarted with a new parent population. The second time the best objective function value stayed the same for the same number of generations, the GA was terminated.

As expected, increasing the population size and termination criterion had a positive impact on the maximum and mean objective function values. For example, with a population size of 300 and a termination criterion of 75, the mean and maximum objective function values achieved with the *dynamic* GA after 50 runs were 65,883 ± 92 and 66,068, respectively. In comparison, the global optimum achieved with MILP was 66,360. However, the improvements to the objective function value came at the cost of longer execution times, i.e. upwards of 15 s for a single run on average. Therefore, for the best trade-off between the solution quality, i.e. the objective function

value, and the performance of the *dynamic* GA, the population size and the termination criterion were set to 200 and 25, respectively.

The comparison of the results between the MILP and the two GA versions is summarised in Table 3.13. After 50 runs, the mean best objective function value using the *standard* GA was 61,186 ± 437 while the *dynamic* GA (with a population size of 200 and a termination criterion of 25 generations) achieved 65,652 ± 112**.** The rolling time horizon approach led to significant performance gains. Not only the mean objective function value obtained with the *dynamic* GA was higher and had lower standard deviation than the *standard* GA, but also the execution time was approximately 2.7 times faster (8.09 s vs 21.56 s). The *dynamic* GA was also comparable to the relaxed MILP model both in terms of the speed (8.09 s vs 8.77 s) and solution quality. Using the known global optimum of 66,360 as an upper bound, the average and the lowest optimality gaps achieved with the *dynamic* GA (with a population size of 200 and a termination criterion of 25 generations) were estimated to be 1.1% and 0.8%, respectively. In comparison, the relaxed MILP model returned an objective function value of 65,940 with a 0.6% optimality gap. The comparison of the Gantt charts in Figure 3.6 shows that the scheduling pattern of the *dynamic* GA (Figure 3.6.b) is similar to that of the relaxed MILP model (Figure 3.6.a), for example:

- Facilities i_1 and i_2 run with little to no idle time and with a variety of different products allocated to them.
- Facility i_3 is busier in the first half of the scheduling table with more product allocations.
- Product p_4 is almost exclusively produced in the facility i_4.
- Facility i_4 has no idle time periods.
- Certain facilities such as i_5 and i_{10} are completely idle.

Table 3.13. Case study 2 results and model statistics for MILP and GA models.

	MILP		**GA**			
	Global optimum	Relaxed	Dynamic[a]	Dynamic[b]	Standard[a]	Standard[a]
Max obj. function value	66,360	65,940	65,849[1]	65,877[1]	61,880[1]	62,193[1]
Time (s)	1000.36	8.77	8.09[2]	8.18[2]	21.56[2]	24.08[2]
Optimality gap[3]	0%	0.6%	0.8%[3]	0.7%[3]	7.8%[3]	7.4[3]
Avg. obj. function value[4]	-		65,652 ± 112	65,686 ± 105	61,186 ± 437	61,490 ± 469
Population size	-		200		200	
Crossover rate, *pC*[5]	-		0.935		0.597	
Mutation rate, *pMutP*[5]	-		0.018		0.001	
Mutation rate, *pMutT*[5]	-		0.867		0.295	
Elitism	-		70%		5%	
Termination	-		25[6]		1000	

[a] Results obtained using the same random number generator seed from the meta-optimisation.

[b] Results obtained using a different random number generator seed.

[1] Max obj. function value obtained from 50 independent GA runs.

[2] An average solution time of a single GA run.

[3] An optimality estimate relative to the global optimum obtained using the recreated MILP model, i.e. *1 – obj. function value / global optimum*

[4] Average of best objective function values from 50 independent GA runs
(mean ± 1 standard deviation)

[5] The parameter values were selected using the PSO algorithm.

[6] If the best objective function value remained unchanged for 25 consecutive generations, the GA was restarted with a new parent population. The second time the best objective function value stayed the same for the same number of generations, the GA was terminated. The maximum generation limit was set to 1000.

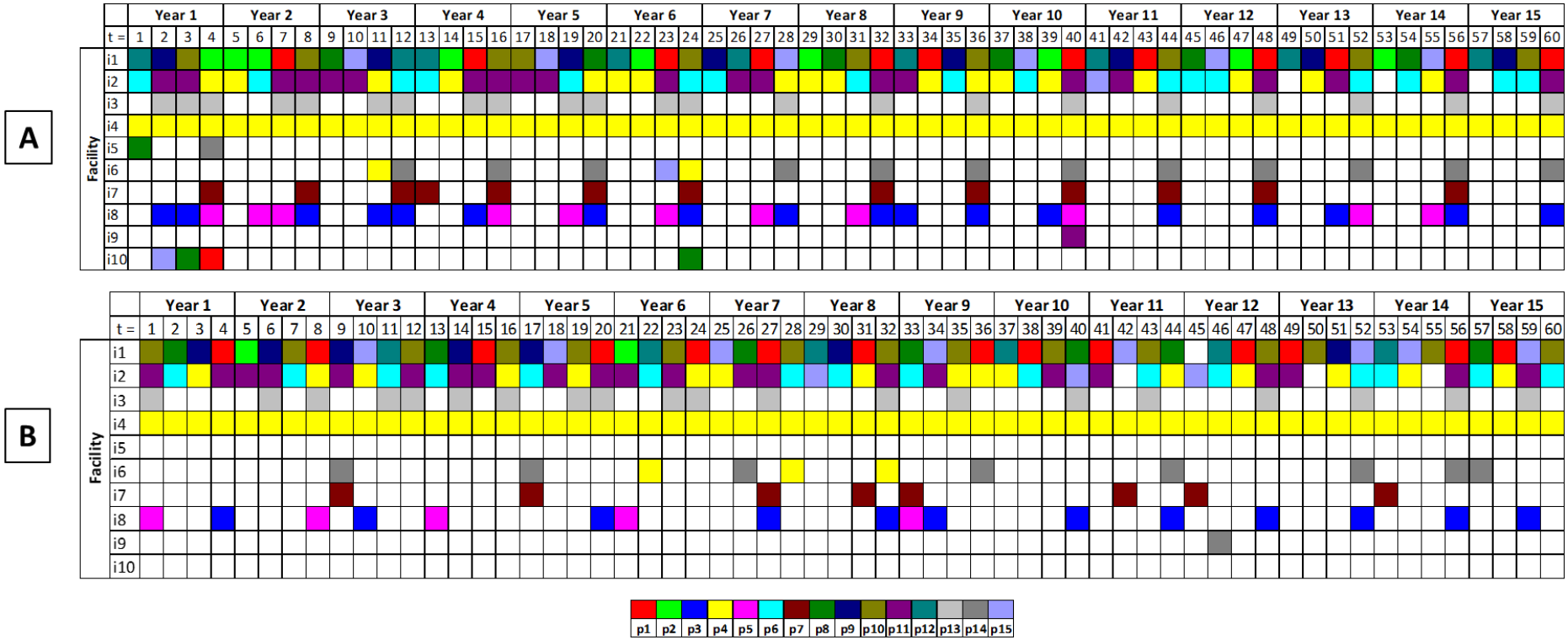


Figure 3.6. Production schedules for case study 2. Each product $p \in \{ p_1, p_2, \ldots, p_{15} \}$ is denoted by a color label displayed in the legend below the schedules. The numbers of batches of each product produced have been removed for clarity purposes.
(a) generated using the MILP model. An objective function value of 65,940 RMU was obtained with 0.6% optimality margin (based on the known global optimum as the upper bound).
(b) generated using the dynamic GA. An objective function value of 65,849 RMU was obtained (0.8% estimated optimality margin)

3.6. Summary

This chapter has demonstrated how a GA can be applied to solve medium- and long-term biopharmaceutical capacity planning problems formulated as discrete-time mixed integer programs in a fast and efficient manner. The key enabling features of the GA-based approaches included a chromosome encoding strategy, a rolling time horizon approach to improving the performance of the GA for tackling the long-term planning problem, and algorithms that captured capacity planning objectives for multiple products across multiple suites and facilities. A PSO-based meta-optimisation method was also presented for automatically setting crossover and mutation parameter values based on the average best objective function value achieved with the GA. The viability of the GA-based scheduling optimisation approaches was demonstrated on two industrially-relevant case studies from the literature.

In case study 1, a medium-term capacity planning problem of a single-site, multi-suite biopharmaceutical facility was solved. The proposed GA obtained the global optimum faster than a related MILP model. In case study 2, a more computationally complex, long-term capacity planning problem of a multi-site biopharmaceutical manufacture was solved. Using the rolling horizon approach, the full-scale problem was divided into 15 sub-problems which were solved consecutively. Using the parameters for the best trade-off between the performance and solution quality, the average run time of the *dynamic* GA was 8.09 s whereas the average optimality gap of the solutions was 1.1%, according to the known global optimum.

4. Continuous-Time Biopharmaceutical Capacity Planning and Scheduling

4.1. Introduction

In the preceding chapter, GA approaches were compared with MILP for discrete-time based optimisation of biopharmaceutical capacity plans. In the first case study, both GA and MILP models generated a globally optimal solution. The discretisation of the time horizon into a number of time intervals of uniform durations was advantageous in terms of making it simpler to model the planning problem but it also had several shortcomings. The key one was the inability to meet the product demand on time (Figure 4.1). This was because of the inherent limitation of the discrete-time representations adopted in the original MILP formulation by Lakhdar et al. (2005) in their biopharmaceutical capacity planning model. The constraints of fixed time periods and the manufacturing of at most one product at any given time period irrespective of the sufficient time available for further production resulted in several days of unutilised production time (see Figure 4.2). For example, the *USP1* and *USP2* suites were occupied for approximately 89% and 84% of the total available production time, respectively. So, the demand for product p_1 at time period t_4 was not met even though the facility had spare capacity.

Other shortcomings of discrete-time based models have been reported in the literature. These include inaccuracy, due to the aforementioned approximation of the time horizon, as well as unnecessary increases in of the overall size of the resulting mathematical programming problems, due to the introduction of a large number of binary variables associated with each discrete time interval have been reported in the literature (Floudas & Lin, 2004). To address these drawbacks, methods based on

continuous-time representations have received a substantial amount of attention. They provide greater potential for the development of more efficient and realistic modeling and solution approaches. In the continuous-time models, the manufacturing campaigns (more broadly referred to as *events*) are allowed to take place at any point in the continuous domain of time. This kind of flexibility is accomplished using variable event times that can be either made to be specific to each unit/product or defined globally. Using the continuous-time approach, the mathematical programming problems can sometimes end up being smaller in size and easier to solve because of the elimination of the inactive time periods.

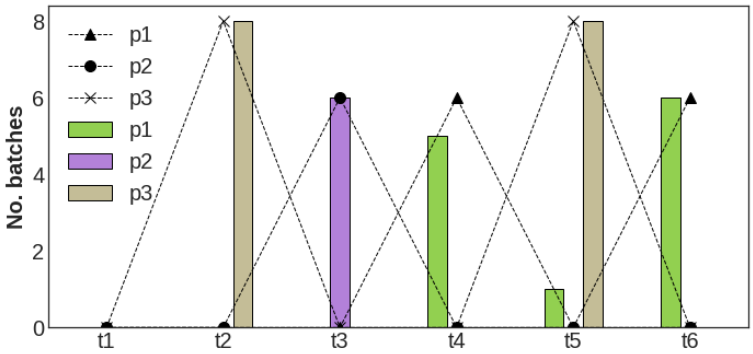


Figure 4.1. Supply (bar) and demand (line) profile of the globally optimal solution to the case study 1. The demand for product p1 at time period t4 was not met on time.

a)

		Time periods (t_n = 60 days)					
		t_1	t_2	t_3	t_4	t_5	t_6
Suites	USP 1	4 (60)	4 (50)	2 (50)	3 (60)	2 (40)	3 (60)
	USP 2	2 (54)	2 (44)	2 (44)	4 (60)	4 (50)	2 (50)
	DSP 1	3 (55)	5 (50)	2 (50)	3 (30)	2 (20)	5 (50)
	DSP 2	2 (52)	2 (20)	2 (20)	3 (55)	5 (50)	

b)

Suites	USP 1		10	10		20	
	USP 2	6	16	16		10	10
	DSP 1	5	10	10	30	40	10
	DSP 2	8	40	20	5	10	

Product 1 Product 2 Product 3

Figure 4.2. Unutilised production time. (b) The numbers in the cells indicate how many days were left unutilised by (a) the globally optimal solution.

However, due to the variable and more flexible nature it becomes more difficult to model scheduling problems and the continuous-time based MILP models often tend to have even more complicated formulations than the discrete-time based alternatives. Moreover, the usefulness and computational efficiency of the continuous-time formulation depend on the number of predefined event points (Méndez et al., 2006). If the global optimum of the scheduling problem requires at least *n* points then fewer points will lead to sub-optimal or even infeasible solutions whereas a large number of points will lead to long computation times. Since the number of points is not known in advance, it is usually determined iteratively by increasing it until there is no improvement in the objective function. In certain cases, a substantial number of model instances need to be solved for each scheduling problem. Furthermore, this stopping criterion does not guarantee the optimality of the schedule and may terminate with a sub-optimal solution.

Inspired by the NeuroEvolution strategies, e.g. Stanley and Miikkulainen (2002) , this chapter presents a novel variable-length chromosome structure and a set of new genetic operators to automatically determine the optimal permutation, number, and length of production campaigns to satisfy the capacity planning problem objectives and constraints. This variable-length GA-based scheduling optimisation method is validated on two industrially-relevant case studies adapted from the literature and compared with related discrete- and continuous-time MILP models.

This chapter is organised as follows: **Section 4.2** defines the scheduling problems of the two examples in more detail. **Section 4.3** describes the key components of the novel continuous-time GA-based approach for biopharmaceutical capacity planning. **Section 4.3.1** explains the variable-length chromosome structure and encoding strategy. **Section 4.3.2** introduces new genetic operators and **Section 4.3.3** explains the scheduling heuristic used for evaluating the fitness of each candidate solution and

constructing the Gantt charts. Case study 1 and 2 results are presented in **Section 4.4**. The flexible GA-based approach is compared with discrete- and continuous-time based MILP models on the example 1 in **Section 4.3.1**. In **Section 4.3.2**, a discrete-time MILP model is used to benchmark the performance of the flexible GA-based approach on the example 2.

4.2. Problem Definition

In this chapter, the novel variable-length GA is validated on two examples adapted from Lakhdar et al. (2005). Both examples are based on industrially-relevant data and cover the most common aspects of the biopharmaceutical manufacturing.

4.2.1. Case Study 1

This case study has been already presented in the first case study of the previous chapter. This particular scheduling problem was first solved by Lakhdar et al. (2005) using a discrete-time based MILP model, then later by Kabra et al. (2013) using a continuous-time MILP model based on an STN framework, and finally by Vieira et al. (2016) using a continuous-time MILP model based on RTN framework. In the original problem statement in Lakhdar et al. (2005), the planning horizon was discretised into time periods of uniform durations (60 days). Therefore, the problem data and most of the constraints were time period-based. In this chapter, the problem statement and the original data are adjusted to suit the continuous-time domain. The rightmost boundary of each time period is assigned as a due date for product demand. For example, any product demand due in the first time period in the discrete-time model is equivalent to being due on the 60^{th} day from the beginning of the schedule in the continuous-time model. It is assumed that overproduction is not allowed, sales are possible only at the demand date, and the backlog can be sold by the next 60^{th} day.

For example, if the delivery on the first 60th day from the start of schedule was missed, that late order can be sold after the next 60 days, i.e. on the 120th day. Additional adjustments include the conversions of the time period-based shelf-life durations and the continuous rates of *USP* and *DSP* production into an actual number of days. Both Kabra et al. (2013) and Vieira et al. (2016) had to make similar assumptions and adjustments to suit their continuous-time MILP-based models. The problem statement for case study 1 is as follows:

- Given:
  - 3 biopharmaceutical products $p = \{ p_1, p_2, p_3 \}$
  - A biopharmaceutical facility with 2 *USP* suites $i = \{ i_1, i_2 \}$ and 2 *DSP* suites $j = \{ j_1, j_2 \}$
  - A continuous planning horizon of 360 days
  - Product-dependent production and changeover durations.
  - Finite product shelf-life and storage capacity
  - Product demand with multiple intermediate due dates
  - Manufacturing, storage, waste disposal, backlog, and changeover costs
- Determine:
  - The number, duration, and sequence of manufacturing campaigns
  - Production quantities along with sales and inventory profiles
- To:
  - Maximise total profit

Table 4.1. Product demand profile [batches] for case study 1. The due date is the nth day from the start of the schedule.

	Due date					
Product	**60**	**120**	**180**	**240**	**300**	**360**
p_1	0	0	0	6	0	6
p_2	0	0	6	0	0	0
p_3	0	8	0	0	8	0

Table 4.2. All relevant parameters for case study 1.

	Product		
	p_1	**p_2**	**p_3**
USP duration [days]	20	22.2	12.5
USP lead time [days]	10	10	10
DSP duration [days]	10	10	10
DSP lead time [days]	10	10	12.5
Shelf-life [days]	180	180	180
Storage limit [batches]	40	40	40
Sell price [RMU / batch]	20	20	20
USP production cost [RMU / batch]	2	2	2
DSP production cost [RMU / batch]	2	2	2
Storage cost [RMU / batch]	1	1	1
Waste disposal cost [RMU / batch]	5	5	5
Backlog penalty [RMU / batch]	20	20	20
USP changeover cost [RMU / batch]	1	1	1
DSP changeover cost [RMU / batch]	1	1	1

4.2.2. Case Study 2

To further demonstrate the features of the variable-length GA-based scheduling optimisation approach developed in this chapter, case study 2 introduces a more complex scheduling problem with more products, more *DSP* suites, and a planning horizon that is nearly twice as long. The topology of the multi-product, multi-suite biopharmaceutical facility in case study 2 is shown in Figure 4.3 and the problem statement is as follows:

- Given:
    - 4 biopharmaceutical products $p = \{ p_1, p_2, p_3, p_4 \}$
    - A biopharmaceutical facility with 2 *USP* suites $i = \{ i_1, i_2 \}$ and 3 *DSP* suites $J = \{ j_1, j_2, j_3 \}$
    - A continuous planning horizon of 540 days and
    - Product-dependent production and changeover durations.
    - Finite product shelf-life and storage capacity
    - Product demand with multiple intermediate due dates
    - Manufacturing, storage, waste disposal, backlog, and changeover costs

- Determine:
    - The number, duration, and sequence of manufacturing campaigns
    - Production quantities along with sales and inventory profiles
- To:
    - Maximise total profit

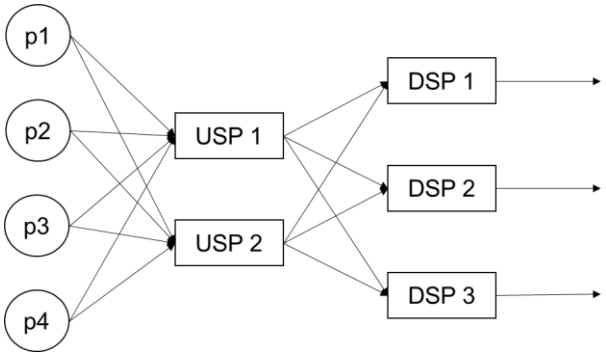


Figure 4.3. Biopharmaceutical facility topology for the example 2.

Table 4.3. Production data for example 2.

	Product			
	p_1	**p_2**	**p_3**	**p_4**
USP duration [days]	20	22.2	12.5	12.5
USP lead time [days]	10	10	10	10
DSP duration [days]	10	10	10	10
DSP lead time [days]	10	10	12.5	12.5
Shelf-life [days]	180	180	180	180
Storage limit [batches]	40	40	40	40
Sell price [RMU / batch]	25	20	17	17
USP production cost [RMU / batch]	5	2	1	1
DSP production cost [RMU / batch]	5	2	1	1
Storage cost [RMU / batch]	1	1	1	1
Waste disposal cost [RMU / batch]	5	5	5	5
Backlog penalty [RMU / batch]	20	20	20	20
USP changeover cost [RMU / batch]	1	1	1	1
DSP changeover cost [RMU / batch]	1	1	1	1

Table 4.4. Product demand profile [batches] for case study 2. The due date is the n^{th} day from the start of the schedule.

	Due date								
Product	**60**	**120**	**180**	**240**	**300**	**360**	**420**	**480**	**540**
$\mathbf{p_1}$	0	0	0	6	0	4	0	0	4
$\mathbf{p_2}$	0	4	0	0	0	0	4	0	0
$\mathbf{p_3}$	0	0	0	0	10	0	0	0	10
$\mathbf{p_4}$	0	6	0	8	0	0	0	0	0

4.3. Methods

In this section, the key components of the GA such as chromosome structure, crossover, and mutation are described. The details of the continuous-time scheduling heuristic for evaluating the fitness of each chromosome and constructing schedules are also outlined. The GA parameters have been tuned using the PSO-based meta-optimisation approach which has been described earlier in Section 3.4.1 of Chapter 3. The fitness of the PSO particle, i.e. GA parameter vector, was assessed by measuring the mean best objective function value achieved after 20 GA runs with a population size of 100 for 100 generations using that parameter vector.

The GA-based DST discussed in Chapter 2 was applied in this chapter to solve the industrially-relevant case studies of multi-suite, multi-product biopharmaceutical manufacture. Python API developed in this thesis provided with the methods for data I/O and visualisation, e.g. to generate Gantt charts and inventory profiles. The variable-length GA with its components and the scheduling heuristic have been implemented in C++ programming language and compiled using a g++-8 compiler. Appendix B discusses the technical details and demonstrates an example of the GA-based DST application using Python API. The discrete-time MILP model was recreated in GAMS 23.9.5 and solved with a CPLEX 12.4.0.1 solver. Both case studies have been solved on an Intel i5-6500 based Ubuntu 16.04 LTS system with 16GB of RAM.

4.3.1. Chromosome Structure

In the previous chapter, the chromosomes encoded the product labels and production time into a table of fixed dimensions which were defined by the problem statement, i.e. the number of products, time periods, facilities/*USP* suites. Unlike a discrete-time representation, a continuous-time one does not have such a grid that is well-defined by the problem variables, e.g. divided by the number of products and time periods of uniform durations. Without the discretised planning horizon, it becomes more challenging to encode the candidate solutions. However, it is still possible to use fixed-length chromosomes in the continuous-time domain, but this approach would have the same aforementioned limitations as continuous-time based MILP models – the number of genes encoding the *events*, i.e. the chromosome length, would have to be determined iteratively thus adding another hyper-parameter that needs to be tuned and possibly worsening the overall GA performance. To eliminate the need for this variable, a variable-length GA is developed to explore the decision space by simultaneously varying both the number as well as the length of individual product campaigns.

The key to the flexible GA-based approach presented in this chapter is a variable-length chromosome structure. At the time of writing, there were not any known works in the literature using variable-length chromosomes to solve process design or capacity planning problems in the biopharmaceutical industry. However, they were applied in other domains such as finding the optimal number, types, and positions of wireless transmitters to meet the objectives of maximum coverage and minimum cost (Ting et al., 2009) and creating an interpreter capable of solving Artificial Intelligence (AI) planning problems described in the standardised Planning Domain Definition Language (PDDL) (Brie & Morignot, 2005). The main source of inspiration for the variable-length chromosome structure presented in this chapter is NeuroEvolution of

Augmenting Topologies (NEAT) method developed by Stanley and Miikkulainen (2002). The artificial evolution of neural networks using a GA has shown great promise in reinforcement learning tasks outperforming standard methods in many benchmark tasks. NEAT enables the neural networks to evolve not only their weights but also the connections and the overall topology from basic elements. This is achieved by employing a flexibile encoding strategy and a set of special genetic operators. This chapter adapts the idea of evolution from the most basic, unit element into a complex solution to create a variable-length chromosome structure for continuous-time scheduling.

In this chapter, every variable-length chromosome comprises the most basic, unit elements called genes. Each gene encodes a single *USP* manufacturing campaign with a product label *p*, a *USP* suite *i* the product campaign would take place in, and the number of batches to be produced. Since the *DSP* campaigns are dependent on the output from the *USP* suites, it is not necessary to encode the *DSP* campaigns information into the variable-length chromosomes. This information can be inferred from the *USP* campaigns when a chromosome is decoded into a production schedule using a continuous-time scheduling heuristic. Figure 4.4.a illustrates the gene and chromosome structures using UML diagrams whereas Figure 4.4.b visualises the overall variable-length chromosome structure at the start and end of a GA. More detailed UML diagrams and C++ implementations of the gene and chromosome are provided and explained in Appendix B.

Even though it is possible to set how many genes within each chromosome would be generated at the beginning of the GA, the algorithm presented in this chapter is designed to evolve the candidate solutions from a single gene, i.e. a single *USP* manufacturing campaign of one batch of a random product assigned to a random *USP* suite. This is accomplished by modifying certain traditional genetic operators,

e.g. uniform crossover, as well as introducing a few new ones to add a new random gene at the end of every GA generation and to mutate the old ones.

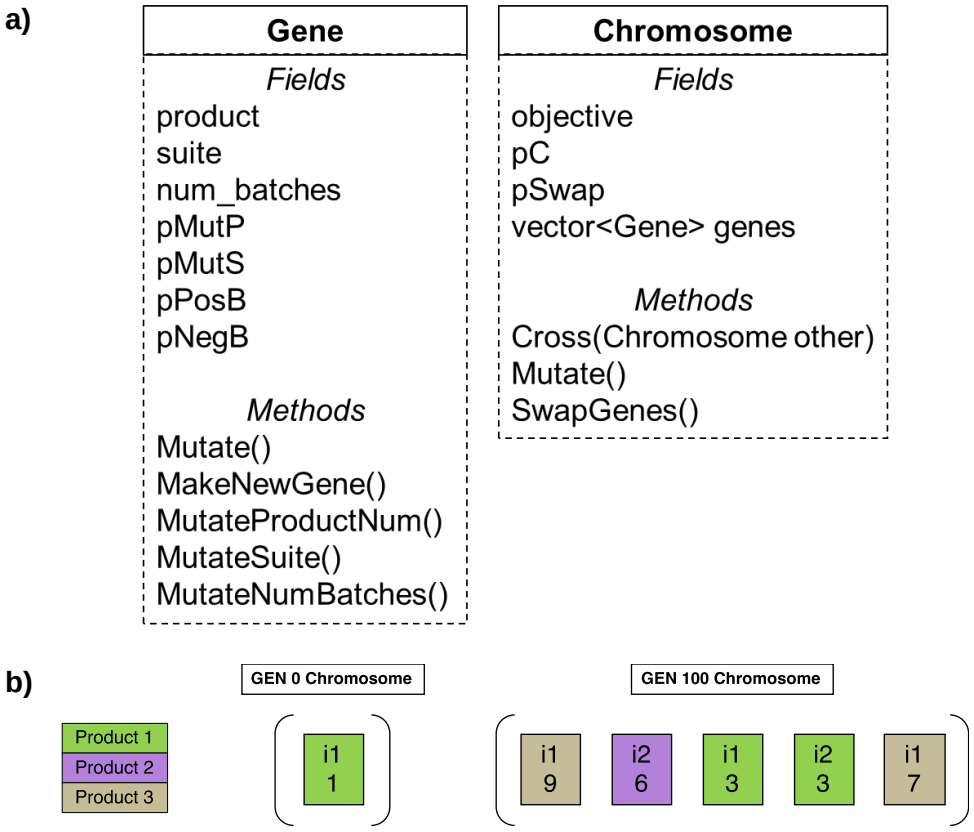


Figure 4.4. Variable-length chromosome:
(a) UML diagram representations of the gene and chromosome structures
(b) An example of a variable-length chromosome at the start (GEN 0) and end of the GA (GEN 100). The values in the boxes correspond to the USP suite label followed by the number of batches produced. The product label is denoted by the colour.

In Figure 4.4.a, chromosome's *Mutate()* method would call gene's *Mutate()* method which comprises individual mutation operators that are discussed in Section 4.3.2.2.

4.3.2. Genetic Algorithm

The search process, i.e. the evolution of the variable-chromosomes, is based on a standard generational scheme using parent and offspring populations. The parent population is not only used to create an offspring population through binary tournaments with replacement, a modified uniform crossover, and a set of special

mutation genetic operators but also to keep a memory of the fittest individuals found by the GA. The offspring replace the parents only if they have a better objective function value. In other words, the parent and offspring populations are combined and the new parent population of *gen + 1* is created by selecting the best solutions from the combined pool.

4.3.2.1. Crossover

The traditional uniform crossover is adapted to suit the variable-length chromosome structure. Before the crossover is applied, the chromosomes are sorted according to the number of genes they possess. This way the crossover operator is performed on similar individuals. Provided that both parent chromosomes have a sufficient number of genes (at least 3), the genes are exchanged with a rate of 0.5 until the end of the shorter chromosome is reached. The extra genes from a longer parent are copied to the shorter one with a rate of 0.5. The crossover operator is illustrated by Figure 4.5.

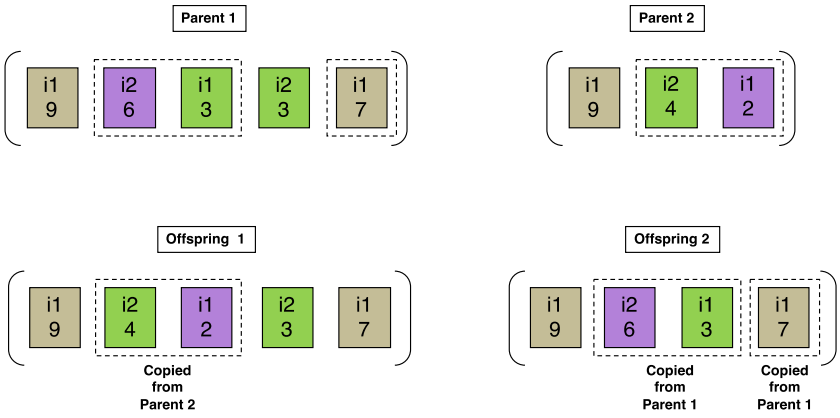


Figure 4.5. An example of a modified uniform crossover between two variable-length chromosomes: genes 2 and 3 are exchanged between the parent chromosomes and gene 5 from the first parent chromosome is copied to the second one.

4.3.2.2. Mutation

Several special gene- and chromosome-level mutation operators are introduced (see Figures 4.4.a and Figure 4.6) to perform the following in an order:

1. *MutateProductNum()*: to mutate product label mutation with a rate of *pMutP.*
2. *MutateSuiteNum():* to mutate *USP* suite label with a rate of *pMutS*.
3. *MutateNumBatches():* to increase or decrease the number of batches by one with a rate of *pPosB and pNegB*, respectively.
4. *MakeNewGene():* to add a new random gene to the end of the chromosome (occurs unconditionally).
5. *SwapGenes():* to swap two genes within the same chromosome with a rate of *pSwap*.

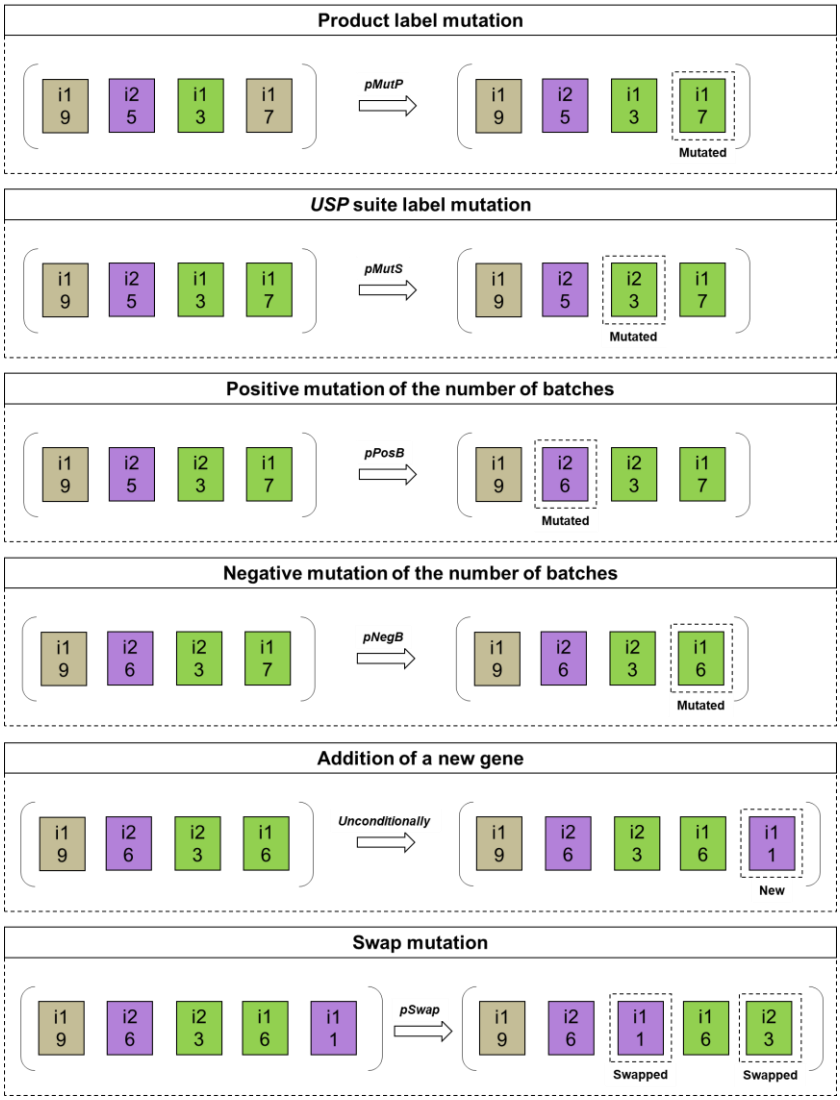


Figure 4.6. Variable-length mutation steps. pMutP, pMutS, pPosB, and pNegB denote the rate of each gene undergoing the corresponding mutation. The addition of a new gene and swap mutation occur once per chromosome.

4.3.3. Continuous-Time Scheduling Heuristic

In both case studies, the fitness of each chromosome is equal to the total profit achieved by the schedule encoded in the variable-length chromosome. However, before the profit can be calculated the schedule and the resulting product inventory, sales, backlog, and waste profiles need to be constructed. One of the main challenges of developing the continuous-time scheduling heuristic of this chapter was finding a way to track the values of various variables over time. In Chapter 3, fitness evaluation was made easier because of the discrete-time representation. The values of binary and continuous variables were stored in arrays of a fixed size that was defined by the problem e.g. number of products and time periods. Therefore, it was relatively simple to "look up" the value of any variable over any given time period.

In order to be able to accurately track information such as the expiry date of each individual batch and how many batches are available for any given demand, the continuous-time scheduling heuristic was developed using Object Oriented Programming (OOP)-based approach. The heuristic is based on three key objects: *Batch, Campaign,* and *Schedule* (see Figure 4.6). The ability to keep track of individual batches makes it possible to generate very detailed production schedules.

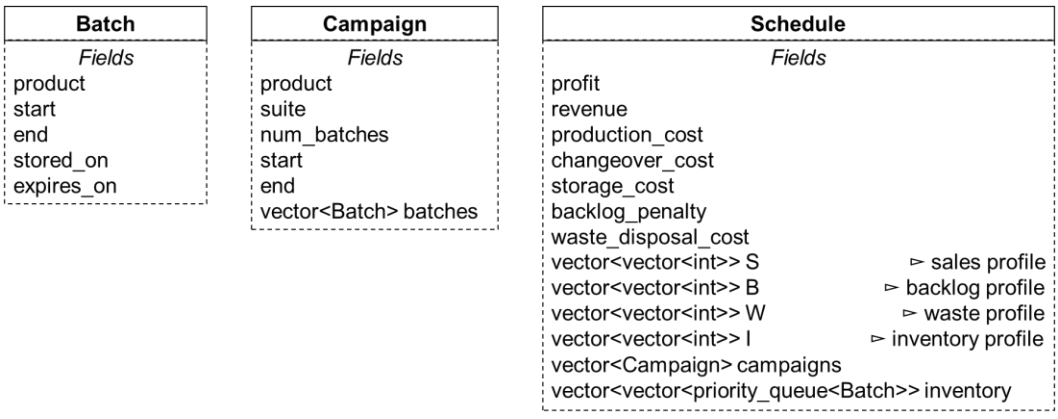


Figure 4.6. UML diagrams of the key objects used in the scheduling heuristic to construct a schedule from a variable-length chromosome.

The *Batch* object represents one whole batch of a specific product and it contains information when the batch was stored (or manufactured) and when it is expected to expire. The *Campaign* object represents the product campaign of a specific product in one of the available processing suites and it contains the following: the start date, end date, and the list of batches (*Batch* objects) produced. The *Schedule* object characterises the final decoded solution or schedule which also comprises the objective function value, i.e. total profit in this chapter, as well as the costs of production, product changeovers, storage, waste disposal, and backlog penalty. The product inventory is implemented using a *priority queue* data structure. A *priority queue* is a data structure containing elements, e.g. batches, such that each one has been assigned a priority based on a specific attribute, e.g. the expiry date. A batch with a higher priority (imminent expiry date) will be processed (or sold) before any batch with lower priority. This way the amount of product wasted due to expired shelf-life is minimised. Each product has an individual *priority queue* for every demand due date. *Schedule* is constructed and evaluated in the following four core steps (see Figure 4.7).

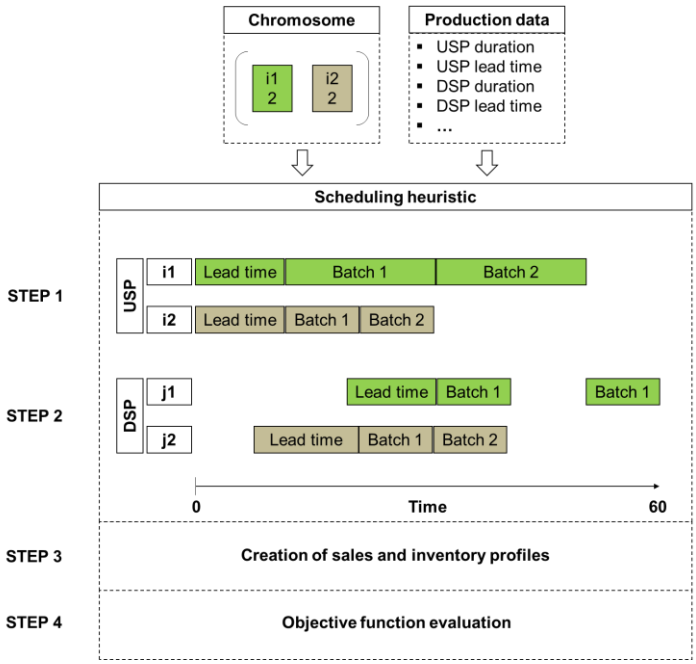


Figure 4.7. Scheduling heuristic. A high-level illustration of how the continuous-time scheduling heuristic is used to decode and evaluate a variable-length chromosome containing two genes.

4.3.3.1. Step 1

First, a schedule of *USP* campaigns is constructed for each *USP* suite based on the production data and the information encoded by the genes within the variable-length chromosome. Each gene is mapped to a *Campaign* object by assigning the product and *USP* suite labels and the number of batches to it. The order of the genes within the chromosome determines the chronological order of the campaigns. The start date of a campaign is equal to the end of the previous one (0 if it is the very first campaign) plus the number of days needed for the equipment set-up and cleaning, i.e. the *USP* lead time (see Table 4.2). The end date of a *USP* production campaign is estimated by adding the product of the total number of batches of that campaign and the number of days needed to produce one batch to the start date. It is ensured that all *USP* campaigns are set to end within the planning horizon defined by the scheduling problem, i.e. 360 days for case study 1 and 540 days for case study 2. Genes encoding the *USP* campaigns beyond the planning horizon are removed from the chromosomes. Algorithm 1 lays out a brief pseudocode for Step 1.

Algorithm 4.1. Pseudocode of the step 1 of the scheduling heuristic.

```
1 procedure CreateUSPSchedule(chromosome, schedule)
2   for each gene in chromosome
3     Create a campaign object
4     Map the values encoded in the gene (product, suite, no. batches) to the campaign
5     if this is the first campaign in the corresponding USP suite
6       campaign.start = USP lead time of the corresponding product
7     else
8       Get the prev_campaign in the current USP suite from the schedule
9       if prev_campaign.product == campaign.product
10        Continue prev_campaign
11        if prev_campaign.end > planning horizon
11          Adjust the prev_campaign.num_batches so that prev_campaign.end ≤ planning horizon
12        end if
12        continue
13      end if
14      campaign.start = prev_campaign.end + USP lead time of the corresponding product
15    end if
16    campaign.end = campaign.start + USP no. days to produce the gene.num_batches
17    if campaign.end > planning horizon
18      Adjust the campaign.num_batches so that campaign.end ≤ planning horizon
19    end if
```

Algorithm 4.1. (continued) Pseudocode of the step 1 of the scheduling heuristic.

```
20      if campaign.num_batches == 0
21          Remove the corresponding gene from the chromosome
22      else
23          Add campaign to schedule.campaigns for the corresponding USP suite
24      end if
25    end for
26 end procedure
```

4.3.3.2. Step 2

Using the information from the previous step, a schedule of *DSP* campaigns is created for each *DSP* suite. The earliest *USP* campaigns are assigned to the *DSP* suites with the earliest availability. The start of each *DSP* campaign depends on the day the first *USP* batch becomes available and whether it is necessary to allocate extra time to set-up a *DSP* campaign. For example, if the *USP* batch becomes available on the 10^{th} day for *DSP* but the lead time of a *DSP* campaign is 15 days then the *DSP* campaign will start on the 15^{th} day. It is quite common in the biopharmaceutical industry to take the intermediate product through the *DSP* processing stage as soon as it leaves the *USP* stage generally due to the low stability of the intermediate molecules. Therefore, the scheduling model schedules every *DSP* campaign to start immediately once the batch from the *USP* stage is ready. Similarly to Step 1, every *DSP* manufacturing campaign is represented by a *Campaign* object which, in addition to the product label, the number of batches, the start and end dates, also contains a list of *Batch* objects for each batch of final product. Algorithm 4.2 lists brief pseudocode for Step 2.

Algorithm 4.2. Pseudocode of the step 2 of the scheduling heuristic.

```
 1 procedure CreateDSPSchedule(schedule)
 2   for each earliest usp_campaign in schedule.campaigns
 3     Create a dsp_campaign object
 4     dsp_campaign.product = usp_campaign.product
 5     Find a DSP suite with the earliest availability
 6     if this is the first campaign in the corresponding DSP suite
 7       if DSP lead time > the day the first USP batch is available
 8         dsp_campaign.start = DSP lead time
 9       else
10         dsp_campaign.start = the day the first USP batch is available
```

Algorithm 4.2. (continued) Pseudocode of the step 2 of the scheduling heuristic.

```
11        end if
12      else
13        Get previous_dsp_campaign in the current DSP suite from the schedule.campaigns
14        if previous_dsp_campaign.end + DSP lead time > the day the first USP batch is available
15          dsp_campaign.start = previous_dsp_campaign.end + DSP lead time
16        else
17          dsp_campaign.start = the day the first USP batch is available
18      end if
19      dsp_campaign.end = dsp_campaign.start + DSP duration
20      Create a dsp_batch object
21      dsp_batch.product = dsp_campaign.product
22      dsp_batch.stored_on = dsp_campaign.end
23      dsp_batch.expires_on = dsp_batch.stored_on + shelf-life of the corresponding product
24      dsp_campaign.num_batches = 1
25      Add dsp_batch to dsp_campaign.batches list
26      Add dsp_batch to schedule.inventory for the earliest demand due date
27      for each remaining usp_batch in usp_campaign
28        if the day usp_batch is available + DSP duration > planning horizon
29          break
30        end if
31        dsp_campaign.end = the day usp_batch is available + DSP duration
32        Create another dsp_batch object
33        dsp_batch.product = dsp_campaign.product
34        dsp_batch.stored_on = dsp_campaign.end
35        dsp_batch.expires_on = dsp_batch.stored_on + shelf-life of the corresponding product
36        dsp_campaign.num_batches += 1
37        Add batch to dsp_campaign.batches list
38        Add dsp_batch to schedule.inventory for the earliest demand due date
39      end for
40    end for
41 end procedure
```

4.3.3.3. Step 3

Having both the *USP* and *DSP* schedules constructed in Steps 1 and 2, the next step is to create the profiles for how many batches will be sold, stored, in backlog, and wasted due to expired shelf-life or overproduction, e.g. exceeded storage limits. The product profiles are later used in Step 4 to evaluate the objective function value.

Algorithm 4.3. Pseudocode of the step 3 of the scheduling heuristic.

```
1 procedure CreateProductProfiles(schedule)
2   S ∈ ℤ^(|p|×|d|)                    ▻ supply profile
3   B ∈ ℤ^(|p|×|d|)                    ▻ backlog profile
4   W ∈ ℤ^(|p|×|d|)                    ▻ waste profile
5   I ∈ ℤ^(|p|×|d|)                    ▻ inventory profile
6   for each product p
```

Algorithm 4.3. (continued) Pseudocode of the step 3 of the scheduling heuristic.

```
    for each p demand due date d
      Get the inventory queue for p and d from schedule.inventory
      Add the leftover inventory of p from the last demand due date d – 1 to the current one
      Count the backlog from the last demand due date d – 1, i.e. B_pd += B_p,d – 1
      Remove any expired and excess batches of p from the inventory and add the count to W_pd
      if the number of batches in the inventory ≥ p demand on d
        S_pd = p demand on d
        if there are any batches of p remaining in the inventory
          Use the remainder to fill the backlog orders B_pd and update S_pd
        end if
      else
        S_pd = all available batches in the inventory
        Add the count of late deliveries to backlog B_pd
      end if
      Add the count of the remaining p batches in the inventory to I_pd
    end for
  end for
  Assign S, B, W, I to schedule
end procedure
```

The sales, backlog, waste, and inventory profiles are created on the basis of product demand due dates, i.e. the product profiles are integer arrays of | *p* |-*by*-| *d* | dimensions where | *p* | is the number of products and | *d* | is the number of due dates. The inventory profile is not the same as the inventory of final product. The former is used to record how many batches were left in storage on any given due date *d* and the later is a *priority queue* which gives the highest sales or delivery priority to the older batches. All batches are sold in the order of the date they were stored on, which in return minimises the amout of waste due to expired shelf-life. Any extra amount of unsold product incurs inventory costs which effectively penalises overproduction in the objective function of profit maximisation. Backlog is penalised until it is cleared. Both backlog and inventory costs are cumulative. Step 3 procedure is summarised in Algorithm 4.3.

4.3.3.4. Step 4

The final step evaluates the objective function value by calculating the profit which is equal to the difference between the total revenue, i.e. sales, and the total costs of

USP and *DSP* production, product changeovers, storage, waste disposal, and backlog. The objective function from the original discrete-time Lakhdar et al. (2005) model (see Equation 1 in Section 3.4.2.3) was modified slightly to suit the proposed approach. The production and changeover costs are estimated on the basis of a manufacturing campaign. The costs of storage, waste disposal, and backlog penalty are estimated on the basis of a product demand due date.

Algorithm 4.4. Pseudocode of the step 4 of the scheduling heuristic.

```
1 procedure EvaluateSchedule(chromosome, schedule)
2   for each usp_campaign in schedule.campaigns
3     schedule.production_cost += usp_campaign.num_batches × USP production cost per batch
4     schedule.changeover_cost += USP changeover cost
5   end for
6   for each dsp_campaign in schedule.campaigns
7     schedule.production_cost += dsp_campaign.num_batches × DSP production cost per batch
8     schedule.changeover_cost += DSP changeover cost
9   end for
10  S, B, W, I = CreateProductProfiles(schedule)
11  for each product p
12    for each p demand due date d
13      schedule.revenue += S_pd × sales price of p
14      schedule.backlog_penalty += B_pd × backlog penalty of p
15      schedule.waste_disposal_cost += W_pd × waste disposal cost of p
16      schedule.storage_cost += I_pd × storage cost of p
17    end for
18  end for
19  schedule.profit = (
20    schedule.revenue –
21    schedule.production_cost –
22    schedule.changeover_cost –
23    schedule.backlog_penalty –
24    schedule.waste_disposal_cost –
25    schedule.storage_cost
26  )
27  chromosome.objective = schedule.profit
28 end procedure
```

4.4. Results

In this section, the validity of the variable-length GA-based scheduling optimisation method is demonstrated on two industrially-relevant case studies adapted from the literature. In case study 1, the GA-based optimisation method is compared with a

recreated discrete-time (Lakhdar et al., 2005) and reported continuous-time MILP models (Kabra et al., 2013; Vieira et al., 2016). The problem consists of a multi-suite facility with 2 *USP* and 2 *DSP* suites producing 3 products with multiple intermediate demands due over a 360-day (1 year) planning horizon. In example 2, the GA is compared with a recreated discrete-time MILP model only. The problem consists of a multi-suite facility with 2 *USP* and 3 *DSP* suites producing 4 products with multiple intermediate demands due over a 540-day (1.5 year) planning horizon.

As mentioned earlier, the original input data has been adapted from Lakhdar et al. (2005) to suit the continuous-time domain and the scheduling heuristic presented in this chapter. The continuous production rates are converted from batches per day into production days per batch. The lead times as used in discrete-time model in Lakhdar et al. (2005) include not only the cleaning and set-up time but also the time for the production of the first batch of the product. Therefore, they are adjusted to account for the cleaning and set-up time only. Product lifetime variables are also converted from time periods to the corresponding number of days.

4.4.1. Case Study 1

The model statistics and the comparison of the results between the flexible GA-based approach and the MILP-based models are provided in Table 4.5 for the industrial case study of multi-product, multi-suite biopharmaceutical production. The Gantt charts from the different models are shown in Figure 4.9. In their original work, Lakhdar et al. (2005) reported an objective function value of 487 with 5% optimality gap in 16 s; in contrast the recreated model achieved an objective function value of 490 by solving the problem to zero gap in 0.22 s. Due to the aforementioned limitations of the discrete-time model, the product demand for product p_1 was not met on time during time period t_5 (see Figure 4.8.b).

Kabra et al. (2013) solved the example 1 problem using a continuous-time MILP formulation based on an STN representation and reported an objective function value of 517 with 0% optimality gap, zero backlogs, and zero wastage (Table 4.5.b). Vieira et al. (2016) proposed a continuous-time MILP formulated based on an RTN representation. They solved the problem to zero gap and achieved an objective function value of 519 with no wastage and all product demands met on time (Table 4.5.c). Both models were reported to take more CPU time to solve the example 1 problem (85.5 s and 46.9 s respectively) than the discrete-time MILP due to a large number of constraints needed for accurate monitoring of storage tasks and product changeovers.

It is important to note that both Kabra et al. (2013) and Vieira et al. (2016) made several assumptions about the case study 1 problem. For example, Vieira et al. (2016) had to relax certain storage constraints in order to compare the results with Lakhdar et al. (2005). Furthermore, the number of batches in both continuous-time MILP models was set as a continuous variable in contrast to the original Lakhdar et al. (2005) model. This is also reflected by the continuous values in Gantt charts shown in Figure 4.9.b and Figure 4.9.c. Without the continuous variable assumption, Vieira et al. (2016) reported a lower objective function value of 513. In comparison, the GA achieved an objective function value of 518 maintaining the integer constraints.

The GA-based scheduling optimisation approach developed in this chapter achieved an objective function value of 518 during every single one of the 20 independent runs in 0.05 s on average (the parameters of the GA are listed in Table 4.5.a). The best solution met all product demands on time (Figure 4.8) with zero wastage and had a better objective function value than the discrete-time MILP (490), Kabra et al. (2013) (517), and Vieira et al. (2016) (513) using an integer batch-extent variable.

Table 4.5. Case study 1 scheduling problem: comparison of results from the novel continuous-time GA approach with other discrete-time and continuous-time models.

	Model type			
	GA[a]	**MILP (STN)[b]**	**MILP (RTN)[c]**	**MILP[d]**
Time representation	Continuous	Continuous	Continuous	Discrete
Best obj. function value	518[1]	517[b]	513 (519)[c]	490 (487)[d]
Mean. obj. function value	518 ± 0[2]	-	-	-
Optimality gap	-	0%[b]	0%[c]	0% (5%)[d]
Run time (s)	0.05[3]	85.5[b]	2.2 (46.9)[c]	0.22 (16)[d]
No. runs	20		-	
No. generations	100		-	
No. chromosomes	100		-	
Starting length	1[4]		-	
pC	0.027		-	
pMutP	0.005		-	
pMutS	0.016		-	
pPosB	0.900		-	
pNegB	0.854		-	
pSwap	0.403		-	

[1] Best obj. function value achieved out of 20 runs
[2] Mean best obj. function value ± its standard deviation of 20 runs
[3] Mean running time of a GA single run
[4] Number of genes per chromosomes at the beginning of the GA
[a] Continuous-time GA presented in this chapter
[b] Reported by Kabra et al. (2013)
[c] Reported by Vieira et al. (2016) using an integer batch-extent variable and continuous batch-extent variable in brackets
[d] Recreated model result and the reported one by Lakhdar et al. (2005) in brackets

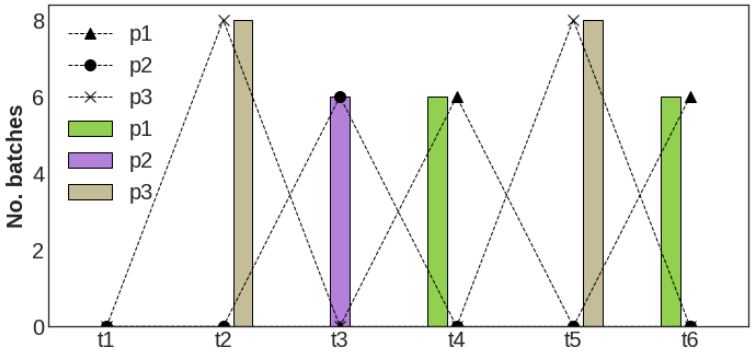
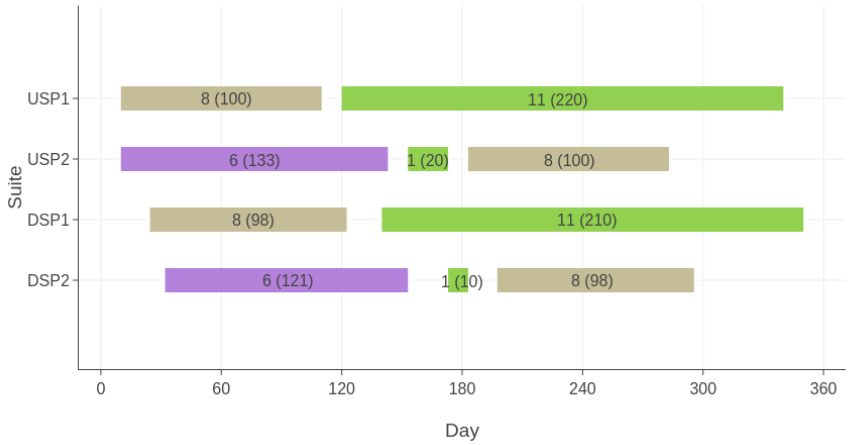


Figure 4.8. Supply (bar) and demand (line) profile of the best case study 1 solution generated with the continuous-time GA-based approach (obj. function value of 518).

In the biopharmaceutical industry, the term *batch* is typically used to denote a complete biopharmaceutical process (see Figure 1.1). If the number of batches is continuous then this could mean either an unfinished process or lower than typical yield. Either way, it is uncommon to have the number of batches set as a continuous

a)

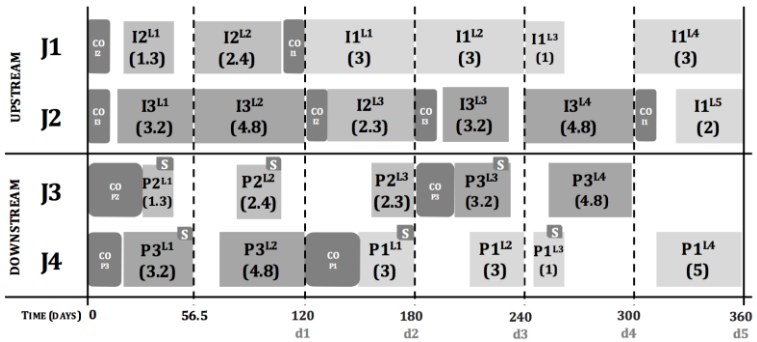


b)

		Time periods (t_n = 60 days)					
		t_1	t_2	t_3	t_4	t_5	t_6
Suites	USP 1	4 (60)	4 (50)	2 (50)	3 (60)	2 (40)	3 (60)
	USP 2	2 (54)	2 (44)	2 (44)	4 (60)	4 (50)	2 (50)
	DSP 1	3 (55)	5 (50)	2 (50)	3 (30)	2 (20)	5 (50)
	DSP 2	2 (52)	2 (20)	2 (20)	3 (55)	5 (50)	

Product 1 Product 2 Product 3

c)



d)

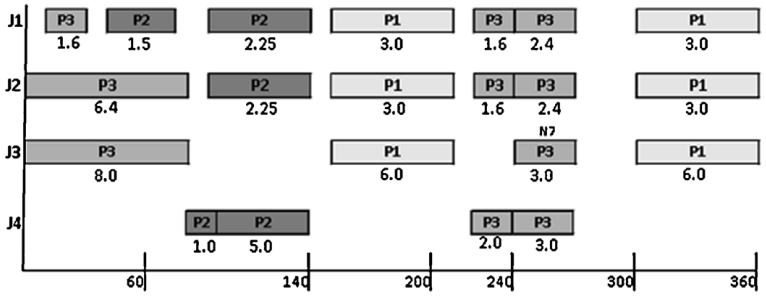


Figure 4.9. Gantt charts generated for the case study 1:
(a) continuous-time GA-based approach (obj. function value of 518). Each box displays the number of batches followed by the campaign length.
(b) discrete-time MILP (obj. function value of 490). Each box displays the number of batches produced and production time.
(c) RTN-based continuous-time MILP (Vieira et al., 2016) (obj. function value of 519, CO indicates a changeover)
(d) STN-based continuous-time MILP (Kabra et al., 2013) (obj. function value of 517)

variable as it makes the interpretation of the result and Gantt charts (see Figure 4.9.c and Figure 4.9.d) more difficult.

The gaps between the campaigns in Figure 4.9.a correspond to the lead time needed to set-up a new campaign or to switch between two different ones. Using the novel GA-based scheduling optimisation approach, the capacity utilisation of *USP1* and *USP2* suites was 94% and 79%, respectively. In contrast, using an integer batch-extent variable, Vieira et al. (2016) reported utilisation rates of 96% and 80%.

It is interesting to see that the scheduling pattern of the novel variable-length GA presented in this chapter (Figure 4.9.a) is very similar to that of the discrete-time MILP model (Figure 4.9.b). For example, there are 6 batches of product p_2 and 8 batches of product p_3 scheduled for manufacture at the beginning of both schedules. Moreover, both models achieved the same average USP capacity utilisation of 86.5%. Nevertheless, the variable-length GA enabled by the continuous-time scheduling heuristic was more effective at utilising the available production time which made it possible not only to meet all product demands on time but also presented an opportunity for additional production capacity.

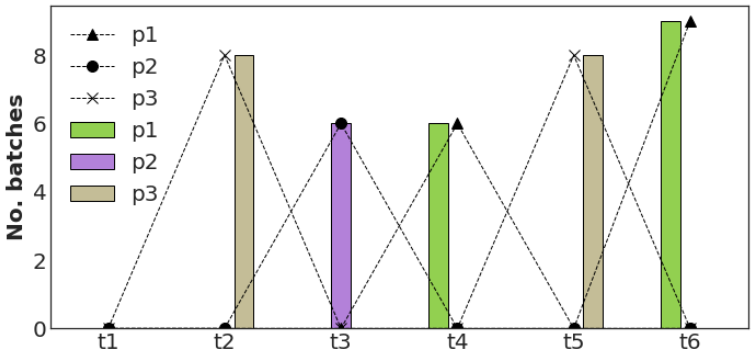


Figure 4.10. Supply (bar) and demand (line) profile of the best solution (obj. function value of 562) generated using the continuous-time GA for the case study 1 with an increased demand for product p1.

The variable-length GA was further tested by increasing the demand for product p_1 at the end of planning horizon by 3 batches, i.e. from 6 to 9 on 540th day. Using the same hyper-parameter values (Table 4.5), the variable-length GA generated a production schedule (see Figure 4.11) with an objective function value of 562 and all product demands met on time (displayed in Figure 4.10). The capacity utilisation of *USP1* and *USP2* suites increased to approximately 97% and 96%, respectively.

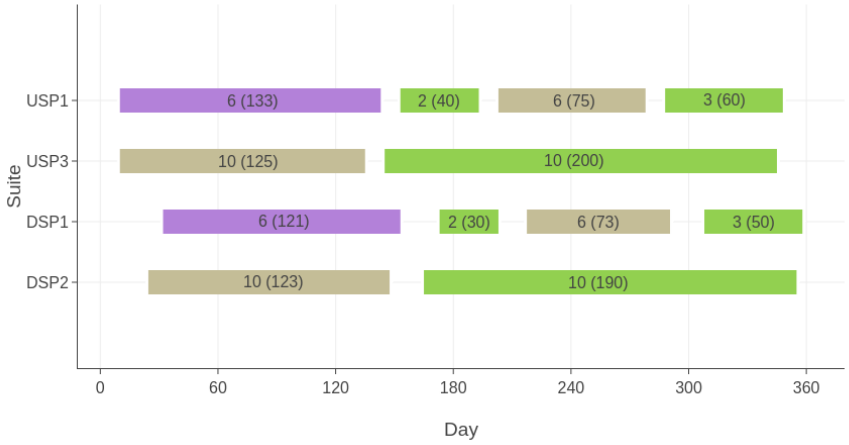


Figure 4.11. Gantt chart generated using the continuous-time GA-based approach for the case study 1 with an increased demand for product p1.

4.4.2. Case Study 2

In this section, a more complex case study of multi-product, multi-suite biopharmaceutical manufacture is used to demonstrate that the novel variable-length GA-based scheduling optimisation approach can be extended for facilities with more manufacturing suites, more products, and longer demand profiles. In case study 2, the proposed GA was used to generate a 1.5 production plan for biopharmaceutical facility with 2 *USP* and 3 *DSP* suites manufacturing 4 distinct products.

The comparison of the results and schedules between the GA and the discrete-time MILP is provided in Table 4.6 and Figures 4.12 and 4.13. The discrete-time MILP model solved the case study 2 problem to 0% optimality gap achieving an objective

function value of 598 in approximately 11 s. Despite the global optimality, due to the aforementioned inherent limitations of the discrete-time domain, the model was only capable of meeting approximately 86% of all product demands on time (see Figure 4.10.b). The capacity utilisation rate was 88% for both *USP* suites.

The GA-based scheduling optimisation approach, on the other hand, significantly outperformed the discrete-time MILP model achieving mean and best objective function values of 725 ± 37 and 801 respectively. Additionally, the best solution generated using the GA met all of the product demands on time (see Figure 4.12) (compared to 8 late deliveries in the MILP solution) without product waste. The capacity utilisation rates of the *USP* suites were 97% and 99%. The GA was also approximately 14 times faster on average than the discrete-time MILP model.

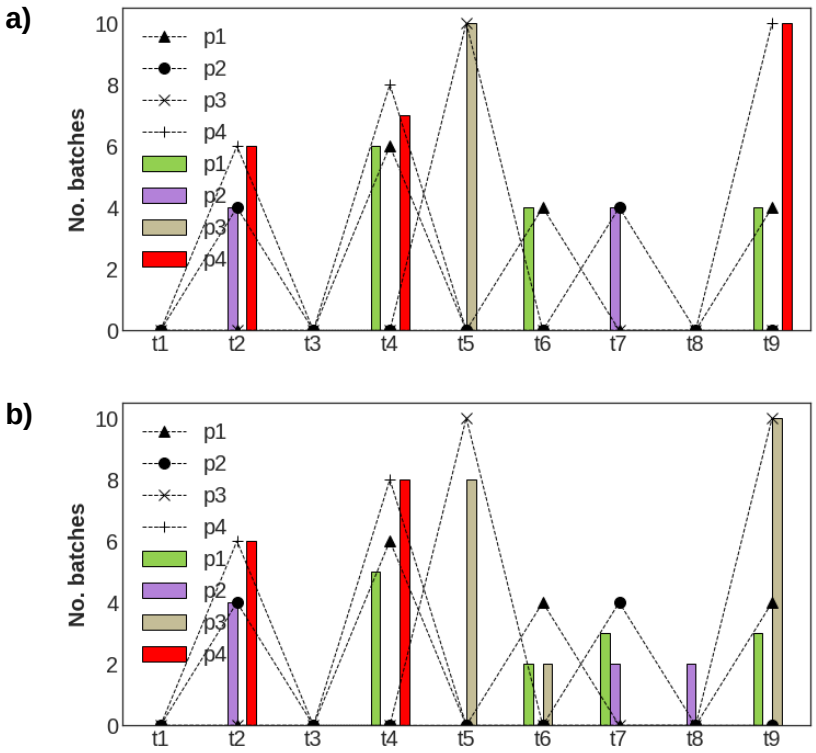


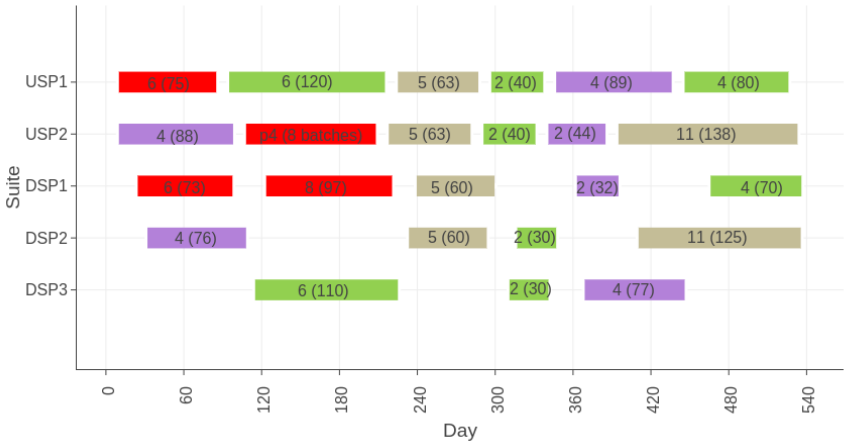
Figure 4.12. Supply (bar) and demand (line) profile of the best case study 2 solution generated with (a) continuous-time GA-based approach (obj. function value of 801) and (b) discrete-time MILP (obj. function value of 598)

Table 4.6. Case study 2 scheduling problem: comparison of results from the novel continuous-time GA approach with discrete-time MILP model.

	Model type	
	GA[a]	**MILP[b]**
Time representation	Continuous	Discrete
Best obj. function value	801[1]	598
Mean obj. function value	725 ± 37[2]	-
Optimality gap	-	0%
Run time (s)	0.79[3]	11.03
No. runs	20	-
No. generations	1000	-
No. chromosomes	100	-
Starting length	1[4]	-
pC	0.027	-
pMutP	0.005	-
pMutS	0.016	-
pPosB	0.900	-
pNegB	0.854	-
pSwap	0.403	-

[1] Best obj. function value achieved out of 20 runs
[2] Mean best obj. function value ± its standard deviation of 20 runs
[3] Mean running time of a single GA run
[4] Number of genes per chromosomes at the beginning of the GA

a)



b)

		Time periods (t_n = 60 days)								
		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
Suites	USP 1	2 (54)	2 (44)	2 (50)	3 (60)	4 (60)	4 (50)	2 (54)	2 (44)	4 (60)
	USP 2	2 (35)	4 (50)	4 (50)	4 (50)	4 (60)	2 (50)	3 (60)	3 (60)	4 (60)
	DSP1			2 (50)	3 (30)		2 (30)	3 (30)	3 (30)	
	DSP 2	2 (52)	2 (20)			6 (60)	4 (40)	2 (52)	2 (20)	2 (54.5)
	DSP 3	2 (54.5)	4 (40)	4 (40)	4 (40)	2 (54.5)				6 (60)

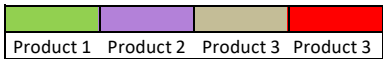


Figure 4.13. Gantt charts generated for the case study 2 using different models:
(a) continuous-time GA-based approach
(b) discrete-time MILP

4.4. Summary

In this chapter, a new variable-length GA-based optimisation approach has been developed for the optimisation of medium-term capacity plans of multi-product, multi-suite biopharmaceutical facilities. The flexible GA-based approach accounts for the same features as its discrete- and continuous-time MILP-based counterparts including but not limited to product-dependent changeovers, multiple intermediate demand due dates, backlogs, limited storage capacity, shelf-life, and waste disposal. The validity of the new approach has been demonstrated on two industrially-relevant case studies previously solved using both discrete- and continuous-time based MILP models from the literature. In case study 1, the proposed GA-based scheduling optimisation approach generated a solution that had higher objective function value than the globally optimal medium-term schedules created related using discrete- and continuous-time MILP models. In example 2, the continuous-time GA-based approach was tested on a problem with a more complex facility topology as well as a longer demand profile. The GA solution met all of the product demands on time significantly outperforming the discrete-time MILP solution, which despite the global optimality and available production capacity only met approximately 86% of all product demands on time.

5. Multi-Objective Biopharmaceutical Capacity Planning and Scheduling

5.1. Introduction

In the previous chapter, a novel variable-length GA-based optimisation approach was developed for continuous-time medium-term capacity planning and scheduling of multi-suite, multi-product biopharmaceutical facility. The novel variable-length GA was demonstrated to be an efficient and flexible optimisation approach outperforming both discrete- and continuous-time MILP models on literature-based industrial case studies. However, both case studies were single-objective while in reality biopharmaceutical companies have to consider multiple objectives and constraints that are often conflicting.

Hence this chapter builds upon the variable-length GA and scheduling heuristic described earlier by incorporating multiple objectives, including maximising the total production throughput, minimising the cumulative deviations from the strategic product inventory targets whilst satisfying demands on time and avoiding product waste over a 3-year period. The continuous-time scheduling heuristic described in the previous chapter is extended with additional constraints and features such as rolling product sequence-dependent changeovers, varying manufacturing yields, and product QC/QA approval times. The functionality of the multi-objective variable-length GA is illustrated on an industrially-relevant case study. The GA-based scheduling optimisation approach developed in this chapter is demonstrated to generate a set of production schedules with optimal number and length of manufacturing campaigns to satisfy the aforementioned objectives and constraints. The importance of the genetic

operators introduced in Chapter 4 and the impact of the starting number of genes on the performance of the GA are also investigated.

This chapter is organised as follows: **Section 5.2** lists the input data and describes the multi-objective biopharmaceutical scheduling problem in more detail. The methods of this chapter are described in **Section 5.3**. **Section 5.3.1** explains how the variable-length chromosome structure described in Chapter 4 was modified to suit the scheduling problem of this chapter. **Section 5.3.2** describes the key parts of the GA with a focus on multi-objective selection and constraint satisfaction components. **Section 5.3.3** presents the extended continuous-time scheduling heuristic for evaluating the objective values of each candidate solution and decoding chromosomes into production schedules. The results and discussion are given in **Section 5.4. Section 5.4.1** defines the bounds of the objective space and sets a benchmark for the multi-objective GA by first solving the scheduling problem of this chapter with a single-objective GA. **Section 5.4.2** evaluates the individual impact of the population size and the number of generations on the performance of the multi-objective GA. **Section 5.4.3** investigates the importance of each genetic operator. **Section 5.4.4** evaluates the impact of the starting number of genes. Finally, the results of the multi-objective GA are discussed and compared with the single-objective GA in **Section 5.4.5.**

5.2. Problem Definition

The focus of this chapter is on multi-objective capacity planning and scheduling of a multi-product biopharmaceutical facility with 1 *USP* and 1 *DSP* suite. The topology of the facility is illustrated in Figure 5.1. The problem statement of the industrial case study is as follows:

- *Given:*
  - A start date (1-Dec-2016) and a planning horizon of 3 years
  - A set of biopharmaceutical products *{ A, B, C, D }*
  - *USP* and *DSP* processing times
  - Product-dependent manufacturing yields
  - Product sequence-dependent changeovers
  - Varying amounts of product stock available at the beginning of the schedule
  - Desired minimum and maximum number of batches per individual product campaign
  - Unique manufacturing requirements to produce the batches in multiples of a specified number
  - QC/QA approval times
  - 3-year profile of strategic product inventory targets
  - 3-year profile of uncertain monthly product demand
- *Determine:*
  - A set of production schedules and the number and length of manufacturing campaigns for each one
  - Production quantities along with inventory and late delivery profiles
- *So as to (constrained deterministic multi-objective problem):*
  - Maximise the total production throughput
  - Minimise the total inventory deficit, i.e. cumulative differences between the monthly product inventory levels and the strategic inventory targets
- *Subject to:*
  - The total backlog being no greater than 0 kg, i.e. meet all product demands without delays
  - The total waste being no greater than 0 kg

The demand forecast comprises a planning horizon of 3 years (1096 days) with realistic monthly due dates. It is often not enough just to be able to meet the product demand on time. In order to be able to deal with unforeseen events and uncertainties such as unplanned facility shutdowns or higher-than-anticipated product demands, biopharmaceutical companies strive to meet specific strategic product inventory

targets. The strategic inventory targets are listed alongside product demand in Table 5.3.

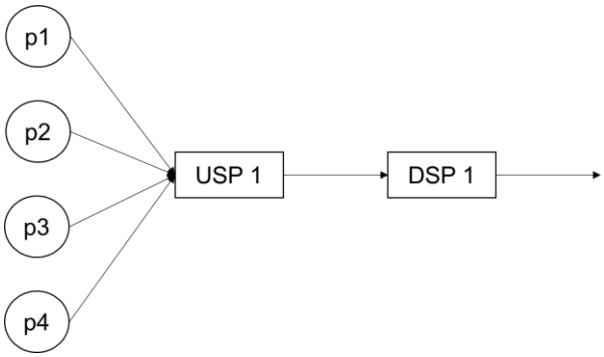


Figure 5.1. Biopharmaceutical facility topology.

The facility is assumed to be available for the entirety of the 1096-day planning horizon. Before the biopharamceutical products can be shipped to meet the demand, they have to pass the 90-day QC/QA process. For example, if a demand for a certain product is due on the 31 March 2018, then the material must be manufactured by the 31 December 2017. Product sequence-dependent changeover time (Table 5.1) is incurred only when there is a switch between different product manufacturing campaigns. Any excess or expired product is considered as wasted material which must be avoided/minimised. Each product has a different manufacturing yield which determines how many kilograms are produced in a single batch. Additionally, due to specific *DSP* requirements, product D needs to be produced in multiples of 3 batches. The complete process data for the industrial case study is provided in Table 5.2.

Table 5.1. Product-dependent changeovers [days].

		To product			
		A	**B**	**C**	**D**
From product	**A**	0	10	16	20
	B	16	0	16	20
	C	16	10	0	20
	D	18	10	18	0

Table 5.2. Process data for the industrial case study.

	Product			
	A	**B**	**C**	**D**
Inoculation duration [days]	20	15	20	26
Seed duration [days]	11	7	11	9
Production duration [days]	14	14	14	14
USP duration [days][1]	45	36	45	49
DSP duration [days]	7	11	7	7
QC/QA duration	90	90	90	90
Shelf-life [days]	730	730	730	730
Yield per batch [kg]	3.1	6.2	4.9	5.5
Storage limit [kg]	250	250	250	250
Opening stock [kg]	18.6	0	19.6	32.0
Minimum batch throughput per campaign	2	2	2	3
Maximum batch throughput per campaign	50	50	50	30
Produce batches per campaign in multiples of	1	1	1	3

[1] USP duration is a sum of inoculation, seed, and production durations

The first objective is to maximise the total kilogram throughput of the production schedule. It is calculated as the sum of throughputs from individual manufacturing campaigns. The second objective is to minimise the total inventory deficit – a cumulative difference between the inventory level and the corresponding strategic target whenever the latter is greater than the former. The multi-objective optimisation problem is also subject to the following constraints: the total amount of backlog and product waste must be ≤ 0 kg. The way the constraints were handled in the model will be explained in the subsequent sections.

Table 5.3. Product demand followed by the strategic inventory targets inside the brackets.

	Product			
Due date	**A**	**B**	**C**	**D**
1-Jan-17	0 (6.2)	0 (0)	0 (0)	0 (22)
1-Feb-17	0 (6.2)	0 (0)	0 (4.9)	5.5 (27.5)
1-Mar-17	3.1 (9.3)	0 (0)	0 (9.8)	5.5 (27.5)
1-Apr-17	0 (9.3)	0 (0)	0 (9.8)	0 (27.5)
1-May-17	0 (12.4)	0 (0)	0 (9.8)	5.5 (27.5)
1-Jun-17	3.1 (12.4)	0 (0)	0 (9.8)	5.5 (33)
1-Jul-17	0 (15.5)	0 (0)	4.9 (19.6)	5.5 (33)
1-Aug-17	3.1 (21.7)	0 (0)	4.9 (19.6)	5.5 (27.5)
1-Sep-17	3.1 (21.7)	0 (0)	0 (14.7)	5.5 (27.5)
1-Oct-17	3.1 (24.8)	0 (0)	0 (19.6)	0 (27.5)
1-Nov-17	0 (21.7)	0 (0)	0 (19.6)	11 (38.5)
1-Dec-17	6.2 (24.8)	0 (0)	9.8 (19.6)	5.5 (33)
1-Jan-18	6.2 (27.9)	0 (0)	4.9 (14.7)	0 (33)
1-Feb-18	3.1 (21.7)	0 (0)	0 (19.6)	5.5 (33)
1-Mar-18	6.2 (24.8)	0 (0)	4.9 (19.6)	5.5 (33)
1-Apr-18	0 (24.8)	0 (0)	0 (14.7)	11 (33)
1-May-18	3.1 (24.8)	0 (0)	0 (14.7)	5.5 (27.5)
1-Jun-18	9.3 (27.9)	0 (6.2)	4.9 (19.6)	5.5 (33)
1-Jul-18	0 (27.9)	0 (6.2)	9.8 (19.6)	0 (33)
1-Aug-18	6.2 (27.9)	0 (6.2)	0 (9.8)	5.5 (33)
1-Sep-18	6.2 (31)	0 (6.2)	0 (19.6)	5.5 (38.5)
1-Oct-18	0 (31)	0 (6.2)	0 (19.6)	5.5 (33)
1-Nov-18	6.2 (34.1)	6.2 (6.2)	4.9 (19.6)	11 (38.5)
1-Dec-18	9.3 (34.1)	0 (6.2)	4.9 (19.6)	5.5 (33)
1-Jan-19	0 (27.9)	0 (6.2)	0 (24.5)	0 (33)
1-Feb-19	9.3 (27.9)	0 (6.2)	9.8 (34.3)	11 (33)
1-Mar-19	6.2 (27.9)	0 (6.2)	0 (24.5)	0 (33)
1-Apr-19	3.1 (27.9)	0 (6.2)	0 (29.4)	11 (44)
1-May-19	6.2 (34.1)	6.2 (6.2)	4.9 (39.2)	5.5 (33)
1-Jun-19	3.1 (34.1)	0 (6.2)	9.8 (39.2)	5.5 (33)
1-Jul-19	0 (31)	0 (6.2)	9.8 (29.4)	0 (33)
1-Aug-19	9.3 (31)	0 (6.2)	0 (19.6)	11 (33)
1-Sep-19	6.2 (21.7)	0 (6.2)	4.9 (19.6)	11 (22)
1-Oct-19	9.3 (15.5)	0 (6.2)	9.8 (14.7)	0 (11)
1-Nov-19	6.2 (6.2)	0 (6.2)	4.9 (4.9)	5.5 (11)
1-Dec-19	0 (0)	6.2 (6.2)	0 (0)	5.5 (5.5)

5.3. Methods

Chapter 2 defined a list of key requirements for the GA-based scheduling optimisation tool including the flexibility and applicability to a wide range of biopharmaceutical

facility models. In order to accomplish this, it is important to continue developing such a framework that could be used to solve a variety of biopharmaceutical scheduling problems without having to make significant changes to it. Therefore, the work of this chapter re-uses a lot of the methods described in Chapter 4, i.e. variable-length chromosome structure, genetic operators, scheduling heuristic. The focus of this section is on the changes and the additional features added to the GA and the scheduling heuristic, e.g. the multi-objective optimisation, the handling of constraints, rolling product changeovers.

Similarly to Chapter 4, the GA-based DST was applied in this chapter to solve the industrially-relevant multi-objective scheduling problem of multi-product biopharmaceutical manufacture. The API developed in Python was used for data I/O and visualisation such as plotting of Gantt charts and Pareto fronts. The variable-length multi-objective GA and the continuous-time scheduling heuristic were both implemented in C++ programming language and compiled with a gcc-8 compiler. Appendix B discusses the technical details and demonstrates an example of the GA-based DST application using Python API. The scheduling problem of this chapter has been solved on an Intel i7-4770HQ based macOS 10.13.5 system 16GB of RAM.

5.3.1. Chromosome Structure

The biopharmaceutical facilities described in the scheduling problem examples in Chapter 4 had relatively complex topologies with multiple *USP* and *DSP* suites. Each variable-length chromosome consisted of genes encoding *USP* suite and product labels and the number of batches produced. According to the problem definition of this chapter, the biopharmaceutical facility has only 1 *USP* and 1 *DSP* (Figure 5.1). Hence, the amount of information that needs to be encoded by each gene can be reduced by removing the *USP* suite labels (see Figure 5.2.a).

In order to demonstrate the flexibility of the novel encoding strategy developed earlier, the core idea behind the variable-length chromosome structure is preserved in this chapter: a 1-D list of genes is used to encode a production schedule. Every gene in the list contains a product label *{ A, B, C, D }* and a number of batches. Figure 5.2.b displays an example of what a variable-length chromosome looks like at the start (GEN 0) and after 100 generations (GEN 100) of the GA have elapsed.

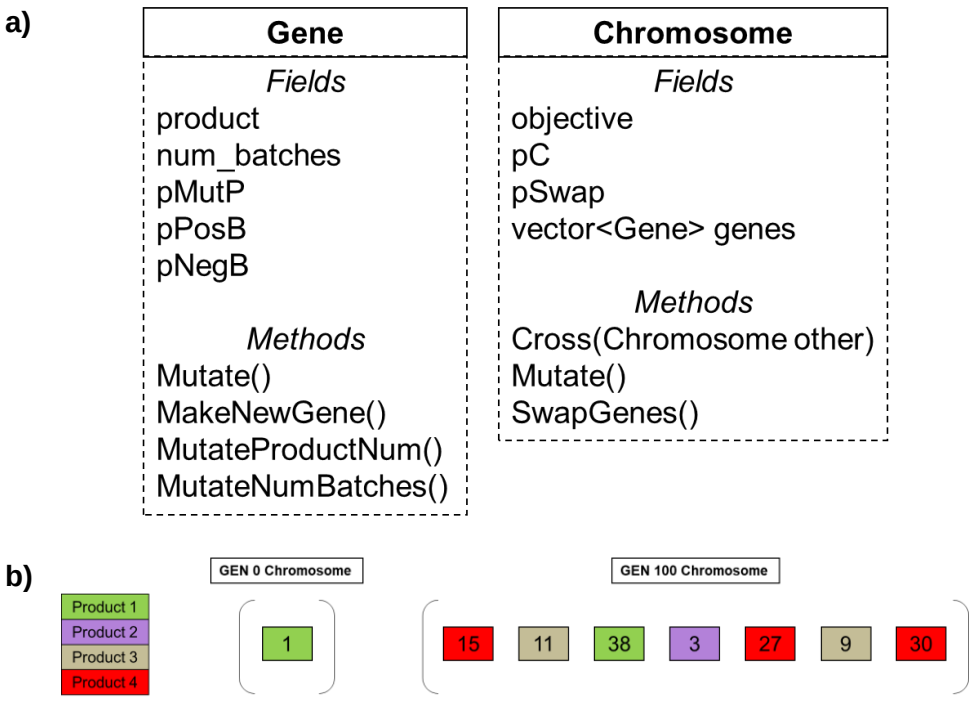


Figure 5.2. Variable-length chromosome:
(a) UML diagram representations of the gene and chromosome structures
(b) An example of a variable-length chromosome at the start (GEN 0) and end of the GA (GEN 100). The values in the boxes correspond to the number of batches produced. The product label is denoted by the color.

The order of the genes (from left to right) defines the timing of each manufacturing campaign. e.g. the second gene in the chromosome encodes the second manufacturing campaign in the production schedule. The initial population is created by generating a pool of random chromosomes containing a single gene. With the aid of special genetic operators described in Chapter 4, the chromosomes are enabled to

grow and shrink in length over the course of the GA. The impact of the starting number of genes on the algorithm's performance is assessed in detail in the results section.

5.3.2. Genetic Algorithm

With the exception of the *USP* suite mutation operator, all other genetic operators described in Chapter 4 and the aforementioned variable-length chromosome representation are integrated into a multi-objective GA that is based on NSGA-II. NSGA-II is well-know for its effectiveness at solving a wide variety of multi-objective problems, e.g. see Raisanen and Whitaker (2005) and Hamdy et al. (2016). The multi-objective variable-length GA employs a generational reproduction scheme using two populations (*parents* and *offspring*) with a fixed number of chromosomes. Parent population is used to keep track of the best solutions found, i.e. provides elitism, while the offspring population is a result of crossover, mutation, and selection operators. Figure 5.3 displays a high-level schematic of the key steps of the multi-objective GA developed in this chapter. After the initial population of single-gene chromosomes is created and evaluated, the steps are performed continuously until the maximum number of generations is reached. For completeness, the descriptions of the genetic operators described in Chapter 4 have also been included in the schematic.

The scheduling problem of this chapter is a constrained multi-objective optimisation problem. An area of the objective space where the corresponding solutions do not meet the constraint requirements is known as *infeasible region*. Production schedules that are not able to meet all product demands on time and/or result in a certain amount of product waste (either due to expired shelf-life or exceeded storage limits) would belong to the *infeasible region*. Constraint handling and representation in heuristic-based optimisation is a difficult issue (Harjunkoski et al., 2014). Simpler constraints such as the fact that a valid schedule has to be a permutation of jobs or product

NSGA-II Ranking

```
1   procedure Rank(parents, offspring)
2     R = combine parents and offspring population
3     F = using non-dominated sort, get a list of all non-dominated fronts from R
4     N = |parents|
5     i = 1
4     parents = ∅
5     while |parents| + |F_i| ≤ N
6       Assign crowding distance to the chromosomes in F_i
7       Add F_i to parents
8       i += 1
9     end while
10    Sort F_i in descending order according to the crowding distance
11    Insert the first N – |parents| elements of F_i into parents
12  end procedure
```

⇩

Binary Tournament Selection based on Constrained-Domination

- If both chromosomes violate constraints, select the one with the smaller violation
- Otherwise, select the one that Pareto dominates the other or has a greater crowding distance

⇩

Crossover

- Sort the chromosomes based on their length
- Apply uniform crossover on each pair of parent chromosomes with a rate of *pC:*
  - If parent chromosomes have different lengths, exchange the genes until the end of a shorter chromosome is reached
  - Conditionally copy the genes from the longer chromosome to the shorter one

⇩

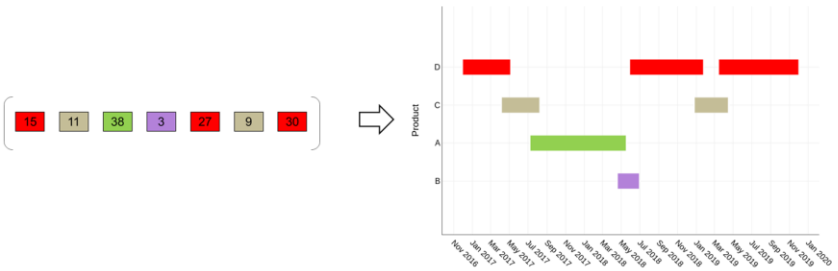
Mutation

- For each gene in the offspring chromosome:
  - Mutate the product label with a rate of *pMutP*
  - Increase the number of batches with a rate of *pPosB*
  - Decrease the number of batches with a rate of *pNegB*
- Add a new random gene
- Swap the positions of two genes within the chromosome once with a rate of *pSwap*

⇩

Scheduling Heuristic

- Using product sequence-dependent changeovers and process data, decode the offspring chromosome into a schedule



- Calculate the total production throughput, total inventory deficit, total backlog, and total product waste levels

Figure 5.3. Schematic of the core steps of the multi-objective GA developed in Chapter 5. Assuming the initial population has been created and evaluated, the steps are looped through until the maximum number of generations is reached.

demands can be mapped into the problem representation and into the choice of genetic operators. However, such implicit representation becomes harder with the increasing number and complexity of constraint.

The most basic methods of constraint handling are to discard all infeasible solutions or to apply a penalty function. More sophisticated methods include the use of repair mechanisms to convert infeasible solutions into feasible ones during the search process or the handling of only some of the degrees of freedom by the meta-heuristic search strategy and fixing the remaining ones during the evaluation of the solution (Harjunkoski et al., 2014), e.g. by using local priority rules (Piana & Engell, 2010).

In this work, repairing infeasible schedules was deemed to be too computationally expensive. The penalty-based constraint handling was rejected to avoid introducing additional parameters into the model. Moreover, according to Sand et al. (2008), incorrectly applied penalty, e.g. too large, may prevent the heuristic from traversing infeasible sub-regions in disjoint search spaces.

There have been several other constraint-handling approaches for the multi-objective problems reported in the literature, e.g. Fonseca and Fleming (1998) and Ray et al. (2001). For its simplicity and computational efficiency, a constraint-handling approach proposed by Deb et al. (2002) is used together with a binary tournament selection to choose more optimal, non-dominated solutions. The pseudocode for this procedure is listed in Algorithm 5.1. Using this approach, the solutions which do not satisfy the constraints of the problem, i.e. with a total amount of backog and/or product waste greater than 0 kg, will not be selected, i.e. will be ranked lower by the NSGA-II ranking algorithm, even if the values of the objectives are better than those of the solutions which fully satisfy the constraints. Therefore, the GA initially selects the chromosomes based on the extent of constraint statisfaction.

Algorithm 5.1. Procedure for binary tournament multi-objective selection based on constrained-domination (Deb et al., 2002). DetermineDominance procedure returns an integer flag of 1 if solution q dominates p, -1 if p dominates q, and 0 if both solutions are non-dominated.

```
 1 procedure Select(q, p)
 2   flag = DetermineDominance(q, p)
 3   if flag == 1
 4     return q
 5   else if flag == -1
 6     return p
 7   end if
 8   if q.d > p.d ▻ if both q and p are non-dominated select the solution with a larger crowding distance
 9     return q
10   else if p.d > q.d
11     return p
12   end if
13   Randomly select between q and p if both solutions have the same crowding distance
14 end procedure
15
16 procedure DetermineDominance(q, p)
17   if q.constraints != p.constraints   ▻ constraints variable is equal to the sum of all constraint violations
18     if q.constraints < p.constraints
19       return 1
20     return -1
21   end if
22   q_dominates = false
23   p_dominates = false
24   for each objective                                  ▻ all objectives are assumed to be minimised
25     if q.objective < p.objective
26       q_dominates = true
27     else if p.objective < q.objective
28       p_dominates = true
29     end if
30   end for
31   if q_dominates == true and p_dominates == false
32     return 1
33   else if p_dominates == true and q_dominates == false
34     return -1
35   end  if
36   return 0
37 end procedure
```

5.3.3. Continuous-Time Scheduling Heuristic

The variable-length chromosomes are decoded into production schedules using a continuous-time scheduling heuristic adapted from Chapter 4. In this chapter, the scheduling heuristic describes the biopharmaceutical manufacturing model of a multi-product biopharmaceutical facility with 1 *USP* and 1 *DSP* suite operating in a fed-

batch mode with staggered bio-reactors and rolling product sequence-dependent changeovers. For completeness, Figure 5.4 provides UML diagrams of the key objects used by the continuous-time scheduling heuristic. Algorithm 5.2 lists a brief pseudocode explaining the schedule construction logic.

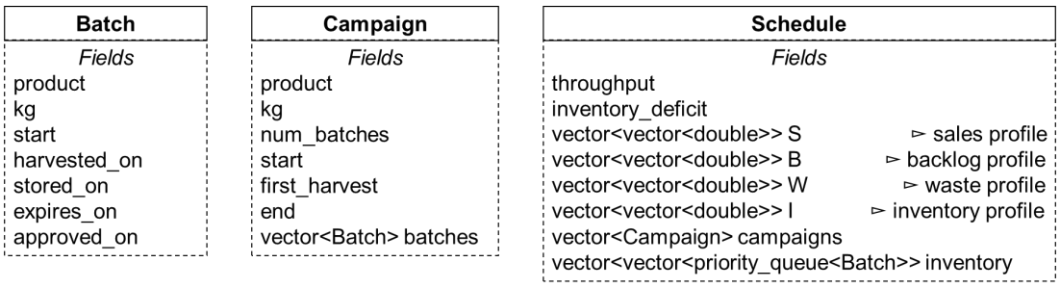


Figure 5.4. UML diagrams of the key objects used in the scheduling heuristic of this chapter to construct a schedule from a variable-length chromosome.

Algorithm 5.2. Pseudocode of the continuous-time scheduling heuristic part that builds a schedule in this chapter.

```
 1 procedure CreateSchedule(chromosome, schedule_start_date)
 2   Create a new schedule object
 3   if AddFirstCampaign(first gene in chromosome, schedule, schedule_start_date) == true
 4     for each remaining gene in chromosome
 5       if prev_gene.product != gene.product
 6         if AddNewCampaign(gene, schedule) == false                ▻ product changeover
 7           break
 8         end if
 9       else
10         if ContinuePreviousCampaign(gene, schedule) == false
11           break
12         end if
13       end if
14     end for
15   end if
16   return schedule
17 end procedure
18
19 procedure AddFirstCampaign(gene, schedule, schedule_start_date)
20   Create a new campaign object
21   campaign.product = gene.product
22   campaign.start = schedule_start_date
23   campaign.first_harvest = campaign.start + USP duration of campaign.product
24   if AddFirstBatch(campaign) == false
25     return false
26   else
27     AddRemainingBatches(gene, campaign)
28   end if
29   Add campaign to schedule.campaigns list
```

Algorithm 5.2. (continued) Pseudocode of the continuous-time scheduling heuristic part that builds a schedule in this chapter.

```
30    return true          ▻ will signal to CreateSchedule procedure to continue building the schedule
31 end procedure
32
33 procedure AddNewCampaign(gene, schedule)
34    Create a new campaign object
35    prev_campaign = last most recent campaign in schedule.campaigns list
36    campaign.product = gene.product
37    campaign.first_harvest = prev_campaign.end + changeover duration          ▻ see Figure 5.6.c
38    campaign.start = campaign.first_harvest – USP time of campaign.product
39    if AddFirstBatch(campaign) == false
40       return false
41    else
42       AddRemainingBatches(gene.num_batches – 1, campaign)
43    end if
44    Add campaign to schedule.campaigns
45    return true
46 end procedure
47
48 procedure ContinuePreviousCampaign(gene, schedule)
49    prev_campaign = last most recent campaign in schedule.campaigns list
50    return AddRemainingBatches(gene.num_batches, prev_campaign)
51 end procedure
52
53 procedure AddFirstBatch(campaign)
54    Create a new batch object
55    batch.product = campaign.product
56    batch.harvested_on = campaign.first_harvest
57    batch.stored_on = batch.first_harvest + DSP duration of batch.product
58    if batch.stored_on > planning horizon
59       return false          ▻ this will send a signal to CreateSchedule procedure to stop
60    end if
61    batch.kg = manufacturing yield of batch.product
62    batch.start = campaign.start
63    batch.approved_on = batch.stored_on + QC/QA approval time of batch.product
64    Add batch to campaign.batches list
65    Add batch to schedule.inventory for the appropriate batch.product demand due date
66    campaign.kg += batch.kg
67    return true
68 end procedure
69
70 procedure AddRemainingBatches(num_batches, campaign)
71    Ensure num_batches is within the minimum and maximum batch throughput bounds
72    Ensure num_batches is a multiple of the given number for gene.product
73    while num_batches > 1
74       Create a new batch object
75       prev_batch = last most recent batch in campaign.batches list
76       batch.product = campaign.product
77       batch.harvested_on = previous_batch.stored_on
78       batch.stored_on = batch.harvested_on + DSP time of batch.product
79       if batch.stored_on > planning horizon
80          return false          ▻ this will send a signal to CreateSchedule procedure to stop
81       end if
82       batch.kg = manufacturing yield of batch.product
83       batch.start = batch.harvested_on – USP duration of batch.product
```

Algorithm 5.2. (continued) Pseudocode of the continuous-time scheduling heuristic part that builds a schedule in this chapter.

```
84      batch.approved_on = batch.stored_on + QC/QA approval time of batch.product
85      Add batch to campaign.batches list
86      Add batch to schedule.inventory for the appropriate batch.product demand due date
87      campaign.kg += campaign.kg + batch.kg
88      num_batches = num_batches – 1
89    end while
90    last_batch = last most recent batch in campaign.batches list
91    campaign.end = last_batch.stored_on
92 end procedure
```

Figure 5.5 explains the concept of rolling product changeovers with a simple illustrative example of how a two-gene chromosome is decoded into a production schedule of two manufacturing campaigns. In Figure 5.5.a, the chromosome contains two genes: one represents a manufacturing campaign of one batch of product A and another – a manufacturing campaign of one batch of product C. The length of each production campaign is determined based on the number of batches within each gene and the number of *USP* and *DSP* days for the corresponding product. For example, it takes 52 days in total (45 for *USP* and 7 for *DSP*) to produce 1 batch of product A. The order of the genes within the variable-length chromosome determines the timings of the manufacturing campaigns. Hence, the campaigns are scheduled in sequence one after another. At the first glance, it might seem that the two manufacturing campaigns in Figure 5.5.b overlap with each other. However, it only looks so because of the aforementioned rolling product sequence-dependent changeovers. Figure 5.5.c illustrates how the rolling changeovers are implemented. For example, once the Inoculation stage of product A is complete, a changeover process can begin to prepare the stage for product C while product A is in Seed stage. The rolling product changeovers have the obvious benefit of making the utilisation of the available production time more efficient. However, not every biopharmaceutical facility design can allow this especially if the individual manufacturing stages do not take place in separate rooms.

The product sequence-dependent changeover time is used to determine the start date of the new campaign. This is illustrated by the black and white striped box which separates the *DSP* stages of products A and C in Figure 5.5.c (see also Lines 37 and 38 in Algorithm 5.2). The manufacturing campaign of product C is scheduled in such a way that its production stage ends 16 days, i.e. the number of changeover days (see Table 5.1), after the end of the manufacturing campaign of product A.

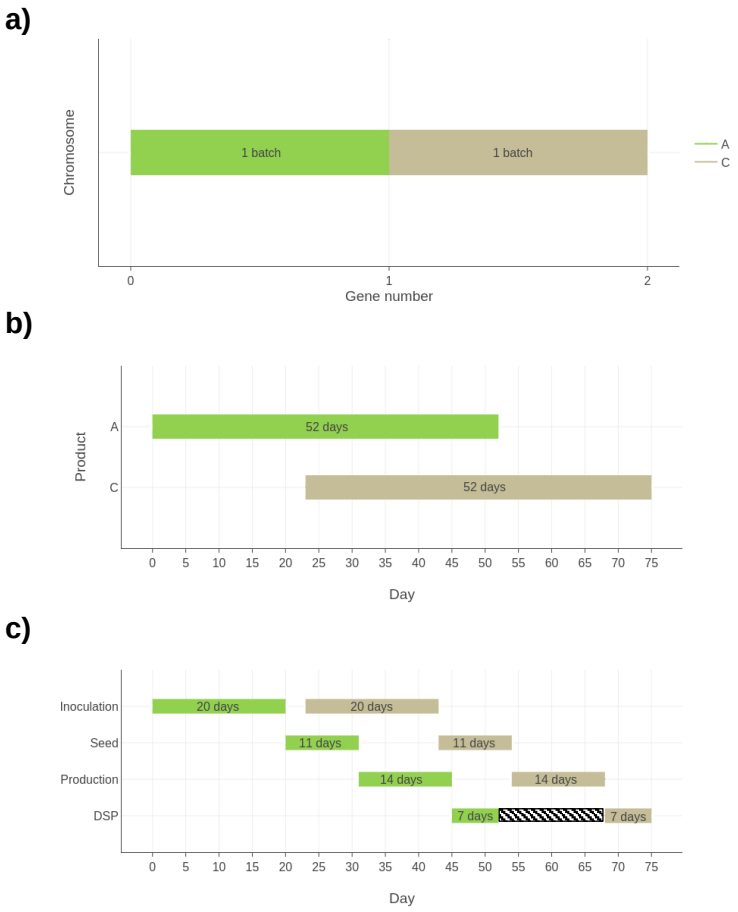


Figure 5.5. An example of the relationship between (a) the genes (b), the decoded production schedule displayed at a product campaign level, and (c) at a manufacturing stage level.

Every finished batch of each product is added to an inventory that is also implemented using a *priority queue* which ensures that the oldest batches are delivered first. Similarly to the continuous-time scheduling heuristic described in the preceding chapter, every product is assigned an individual *priority queue* for each due date. Additional check is introduced to ensure that every batch of product has been

approved by the QC/QA before it can be delivered. The scheduling heuristic maintains that each chromosome encodes a production schedule which starts and ends within the set planning horizon. Genes encoding production campaigns beyond the planning horizon are removed from the chromosome.

The crossover and mutation operators can sometimes cause multiple, consecutive genes encode manufacturing campaigns of the same product. After the schedule has been constructed, the heuristic combines the consecutive genes encoding the campaigns of the same product into one. Figure 5.6. illustrates an example of this.

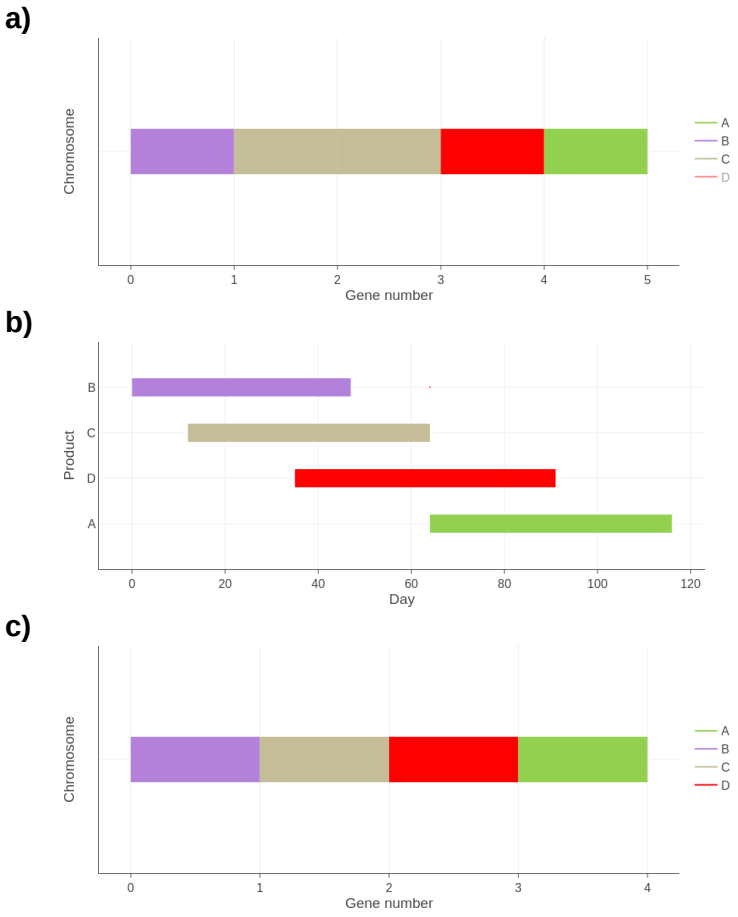


Figure 5.6. Correction of the mapping of genes to the production campaigns. In (a), the genes 2 and 3 correspond to the same product. The continuous-time scheduling heuristic combines them into (b) one contiguous manufacturing campaign and re-maps it to (c) a single a gene.

5.4. Results

In this section, the novel multi-objective variable-length GA developed earlier in this chapter is used to generate 3-year production schedules for a multi-product biopharmaceutical facility. The objectives and constraints of the capacity planning and scheduling problem are to maximise the total kilogram throughput, minimise the total kilogram inventory deficit whilst avoiding product waste and meeting all product demands on time. The multi-objective results are discussed in **Section 5.4.5** by comparing the trade-offs between the best non-dominated solutions. **Sections 5.4.2**-**5.4.4** study the relationship between the GA, its genetic operators, and their parameter values by varying them one at a time, keeping all the others unchanged, i.e. by performing ablation studies. It is acknowledged this is not the most optimal way because it does not account for the interactions between the operators (Eiben et al., 1999). Nevertheless, this approach can given some useful insights about the relative importance of each parameter and genetic operator. The following experiments are performed:

- In **Section 5.4.2**, the impact of the number of chromosomes on the GA's performance is investigated while keeping the number of generations constant and vice versa.
- **Section 5.4.3** assesses the importance of each genetic operator by comparing the performance of the GA when the corresponding rate value is set to 0.
- **Section 5.4.4** evaluates the effect of the starting number of genes on the GA's performance. Moreover, it investigates how the length of the chromosomes in the best Pareto front changes as a function of the number of generations.

Every experiment is performed for 50 independent GA runs. The top Pareto fronts from each individual run are combined and sorted again using the non-dominated sort

method (Deb et al., 2002) to obtain the *best Pareto front*. The performance of the multi-objective GA is evaluated on the basis of the maximum and mean hypervolume achieved after 50 runs. The hypervolume indicator measures the size of the area between a reference point (worst possible objective functions values) and the Pareto front. In this work, the maximum hypervolume is equal to the area between a reference point and the best Pareto front whereas the mean hypervolume correspond to the mean area size between a reference point and a Pareto front from a individual GA run.

Using unary performance indicators to assess the performance of multi-objective algorithms can be problematic (Zitzler et al., 2003). Nevertheless, the hypervolume indicator is often used for assessing the performance of many multi-objective evolutionary algorithms (Knowles et al., 2003; Zitzler & Künzli, 2004; Fonseca et al., 2006). In this work, an improved dimension-sweep algorithm proposed by (Fonseca et al., 2006) and provided by the DEAP framework (Fortin et al., 2012) is used to estimate the hypervolume indicator.

5.4.1. Objective Space

In order to set a benchmark for the multi-objective GA and get a better understanding of what the objective space looks like, the scheduling problem was first solved as a single-objective optimisation problem. A single-objective GA with 1000 chromosomes was run for 1000 generations 50 times (50 independent runs). In other words, a total of 50M objective function evaluations were performed to find the best value of each objective subject to the constraints of the scheduling problem (the total amount of backlog and product waste must be equal to 0 kg).

The worst possible values of the objectives (when the total production throughput is 0 kg and when the inventory deficit is equal to the sum of all strategic product inventory target values, i.e. 2651.7 kg) were used as a *reference point* for estimating the hypervolume indicator. The best values of the objectives (total production throughput of 630.4 kg and total inventory deficity of 184.8 kg) obtained with a single-objective GA were combined to create an *ideal point* which together with a *reference point* were used to make an assumption about the boundaries of the objective space for the problem of this chapter. The total area of the objective space was also used to normalise the hypervolume indicator to lie in the 0.0-1.0 range. Figure 5.7 displays the *reference* and *ideal points*, the single-objective solutions, and the objective space of the scheduling problem of this chapter. The results and statistics of the single-objective optimisation are also provided in Table 5.4.

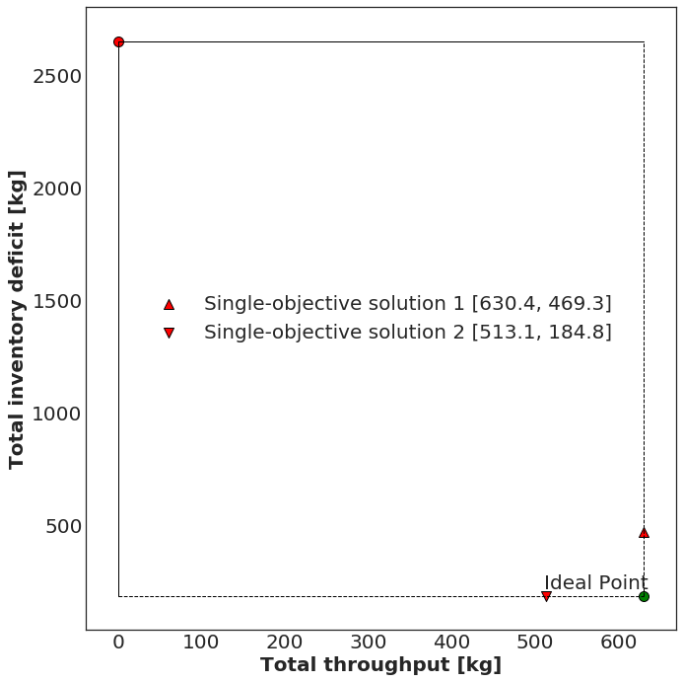


Figure 5.7. The objective space (dashed line) of the scheduling problem described in this chapter. The objectives are to maximise the total production throughput and to minimise the total inventory deficit subject to the sum of total backlog and product waste being equal to 0 kg. The single-objective solutions were obtained with a single-objective GA after 50 independent runs of 1000 generations with 1000 chromosomes.

Table 5.4. The best individual objective values (bold) obtained with a single-objective GA.

	Single-objective solution	
	1. Maximise total throughput	**2. Minimise total inventory deficit**
Total throughput [kg]	**630.4**	513.1
Total inventory deficit [kg]	469.3	**184.8**
Total backlog [kg]	0	0
Total waste [kg]	0	0
Starting length[1]	1	
No. runs	50	
No. generations	1000	
No. chromosomes	1000	
pC	0.108	
pMutP	0.041	
pPosB	0.608	
pNegB	0.766	
pSwap	0.471	
Run time[2]	12.6 s	13.7 s

[1] The starting number of genes per chromosome in the initial population.
[2] Mean run time of a single GA run.

5.4.2. The Impact of The Number of Chromosomes and The Number of Generations

This section assesses the sensitivity of the multi-objective GA to the increasing number of chromosomes while the number of generations is set to a sufficiently large number and vice versa. The parameter values of genetic operators and the starting number of genes used during the single-objective optimisation (Table 5.4) are also applied here to the multi-objective GA.

Figures 5.8.a and 5.8.b illustrate how the maximum and mean hypervolume values as well as the mean time of a single GA run are affected by the number of chromosomes and generations, respectively. Tables 5.5 and 5.6. contain a more detailed summary of the results and statistics of the experiments such as the objective function values of the boundary solutions X and Y from the best Pareto front and the number of unique non-dominated solutions in the best Pareto front. The best

attainment surfaces together with all non-dominated solutions collected from every GA run using different parameter combinations are displayed in Figures 5.9 and 5.10.

Overall, the performance of the GA, i.e. maximum and mean hypervolume values, improves with the increasing number of chromosomes and generations. Based on the comparison between Figures 5.8.a and 5.8.b, it is apparent that the number of chromosomes has a greater impact on the maximum and mean hypervolume than the number of generations. For example, after 50 runs of 1000 generations with 100 chromosomes, the values of maximum and mean hypervolume are 0.992 and 0.982 ± 0.011 respectively, whereas, when the number of generations is set to 100 and the number of chromosomes is set to 1000, the maximum and mean values increase to 0.994 and 0.991 ± 0.005, respectively.

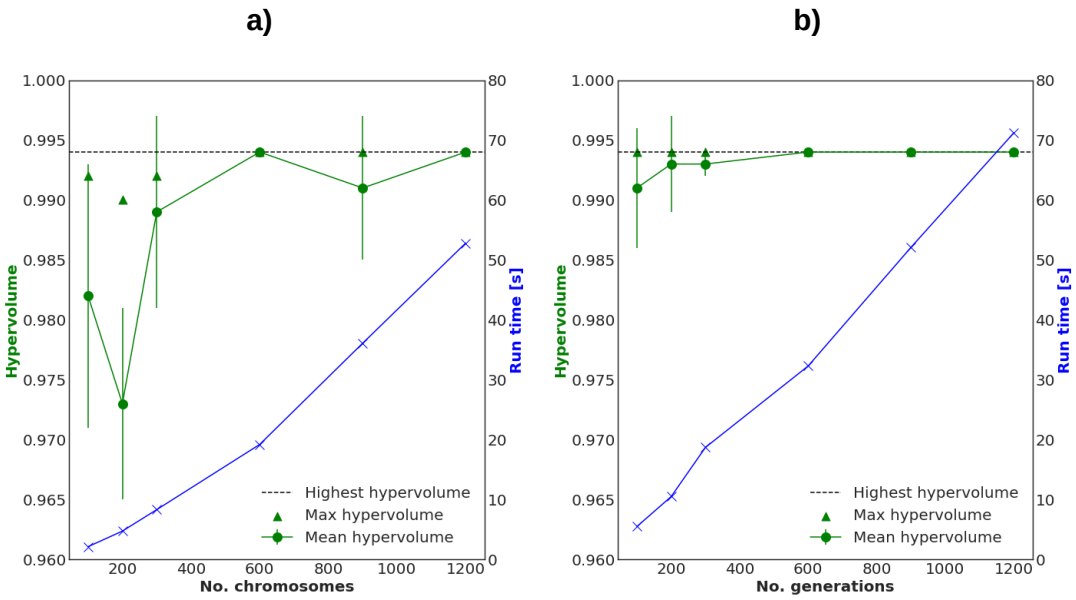


Figure 5.8. The impact of the number of (a) chromosomes and (b) generations on the performance of the multi-objective variable-length GA. In (a), the number of generations was fixed at 1000 whereas in (b) the number of chromosomes was set to 1000. The vertical lines denote the standard deviation of mean hypervolume. The black dashed line marks the highest maximum hypervolume achieved.

Table 5.5. The impact of the number of chromosomes on the performance of the multi-objective variable-length GA.

	No. chromosomes					
	100	**200**	**300**	**600**	**900**	**1200**
Max hypervolume	0.992	0.990	0.992	0.994	0.994	0.994
Mean hypervolume[1]	0.982 ± 0.011	0.973 ± 0.008	0.989 ± 0.008	0.994 ± 0.000	0.991 ± 0.006	0.994 ± 0.000
No. solutions[2]	35	31	35	35	37	35
Solution X[3]	[583.0, 193.4]	[583.0, 193.4]	[583.0, 193.4]	[583.0, 193.4]	[583.0, 193.4]	[583.0, 193.4]
Solution Y[3]	[630.4, 469.3]	[630.4, 469.3]	[630.4, 469.3]	[630.4, 469.3]	[630.4, 469.3]	[630.4, 469.3]
Run time[4] [s]	2.14	4.74	8.32	19.20	36.10	52.8
No. runs	50					
No. generations	1000					
Starting length[5]	1					

Table 5.6. The impact of the number of generations on the performance of the multi-objective variable-length GA.

	No. generations					
	100	**200**	**300**	**600**	**900**	**1200**
Max hypervolume	0.994	0.994	0.994	0.994	0.994	0.994
Mean hypervolume[1]	0.991 ± 0.005	0.993 ± 0.004	0.993 ± 0.001	0.994 ± 0.000	0.994 ± 0.000	0.994 ± 0.000
No. solutions[2]	35	35	36	35	36	37
Solution X[3]	[583.0, 193.4]	[583.0, 193.4]	[583.0, 193.4]	[583.0, 193.4]	[583.0, 193.4]	[583.0, 193.4]
Solution Y[3]	[630.4, 469.3]	[630.4, 469.3]	[630.4, 469.3]	[630.4, 469.3]	[630.4, 469.3]	[630.4, 469.3]
Run time[4] [s]	5.5	10.5	18.7	32.3	52.1	71.2
No. runs	50					
No. chromosomes	1000					
Starting length[5]	1					

[1] Mean ± 1 standard deviation.
[2] The number of solutions in the best Pareto front.
[3] The boundary solutions of the best Pareto front.
[4] Average time elapsed for each of the 50 runs.
[5] The starting number of genes per chromosome in the initial population.

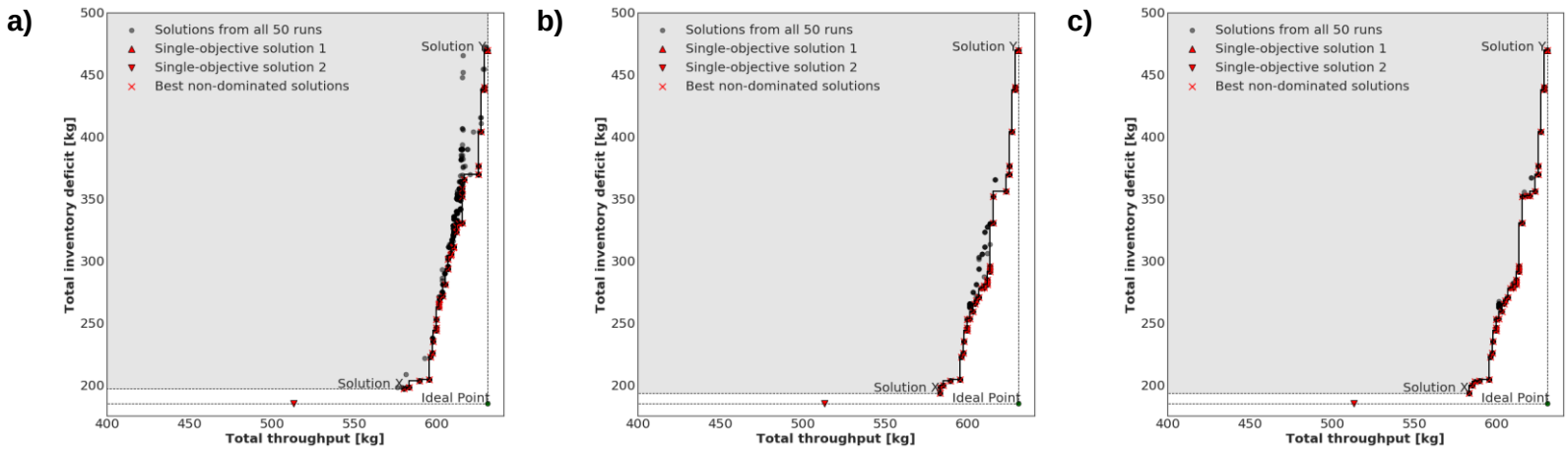


Figure 5.9. All non-dominated solutions (black circles) and the best Pareto front (red crosses) with (a) 100, (b) 600, and (c) 1200 chromosomes.

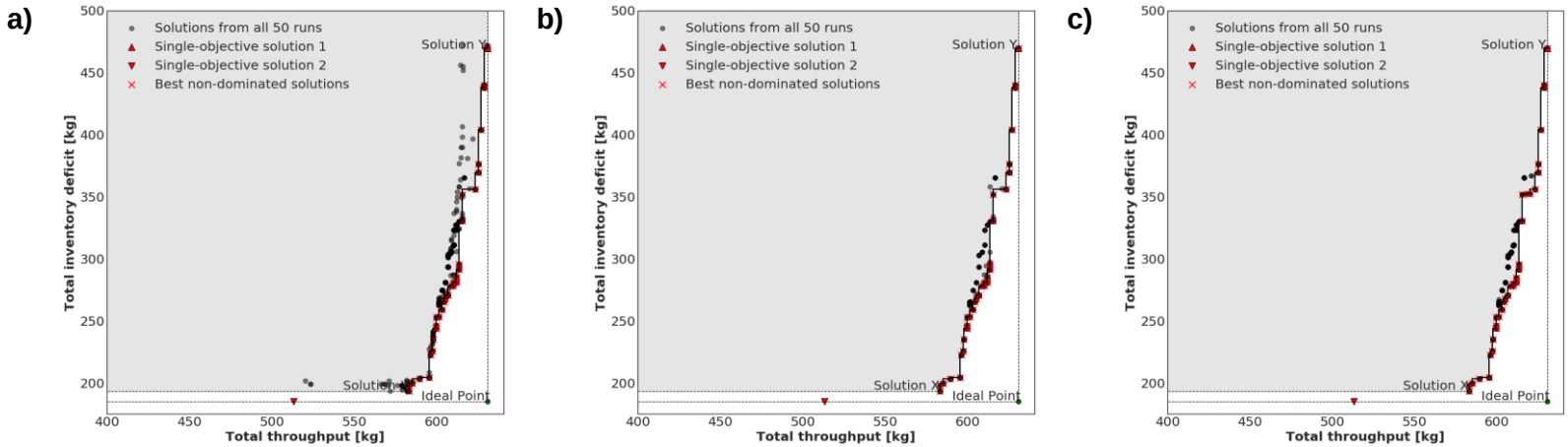


Figure 5.10. All non-dominated solutions and the best Pareto front (red crosses) after (a) 100, (b) 600, and (c) 1200 generations.

However, the relationship between the GA's performance and the number of chromosomes is not perfectly linear, e.g. the maximum and mean hypervolume values are actually higher when the number of chromosomes is 100 rather than 200 (Figure 5.8.a). The performance trend appears to be much more consistent with the number of generations, i.e. increasing this number leads to an improvement. In both cases, the maximum hypervolume stops improving once the total number of objective function evaluations ≥ 30M (50 runs of 1000 generations with 600 chromosomes or 50 runs of 600 generations with 1000 chromosomes).

The mean time of single GA run increases linearly with both the number of chromosomes and the number of generations. Nevertheless, the computational performance of the multi-objective variable-length GA developed in this chapter is more affected by the number of chromosomes rather than generations. It takes longer to run a GA with 1000 chromosomes for 100 generations than the other way around. The reason for this is because the evaluation of the chromosomes is parallelised.

5.4.3. The Importance of Genetic Operators

In the previous chapter, a set of new genetic operators was introduced to give the variable-length GA the means to search for the optimal number and permutation of production campaigns manufacturing the right amounts of the product. In this section, a series of ablation experiments is performed with a purpose of evaluating the relative importance of the following genetic operators:

- Modified uniform crossover which takes place with a rate of *pC*.
- Product label mutation that affects each gene individually with a rate of *pMutP*.
- Positive (+1) and negative (-1) mutations of the number of batches encoded in each gene with the rates of *pPosB* and *pNegB* respectively.

- Swap mutation – two genes are made to swap their positions once per chromosome with a rate of *pSwap.*

First, a benchmark was established by performing 50 runs of the multi-objective GA with 1000 chromosomes for 600 generations with all genetic operator parameter values set to 0.5 (see Table 5.7.a). The number of chromosomes and the number of generations were selected based on the findings from the previous section: out of all the combinations studied, this one gave the best tradeoff between the maximum hypervolume, the consistency of top non-dominated solutions from run to run, and the computational performance. The impact of each genetic operator on the GA's performance was evaluated by setting the corresponing rate to 0. The results and statistics of the experiments are provided in Table 5.7. The impact of disabling each operator is also illustrated by displaying the best Pareto fronts and all non-dominated solutions collected from the individual GA runs in Figure 5.11.

With the exception of product label mutation, individually disabling all other genetic operators had a negative impact on the mean hypervolume. Assuming the importance of each genetic operator can be quantified by the increase/decrease in mean hypvervolume when it is disabled, then, according to the results of ablation experiments, the operators can be ranked in the following order (from the most to the least important):

1. *Negative mutation of the number of batches*. Disabling this operator reduced the base case mean hypervolume from 0.930 ± 0.009 to 0.844 ± 0.061. Moreover, the consistency of the GA's performance was signficantly reduced. In Figure 5.11.e, the non-dominated solutions collected from the individual runs are a lot more widely scattered compared to the base case (Figure 5.11.a).

2. *Positive mutation of the number of batches*. Without this operator, the base case mean hypervolume dropped from 0.930 to 0.912 whereas the standard deviation increased from 0.009 to 0.016. Compared to the negative mutation of the number of batches, the impact was not as severe because the GA had other means of increasing the number of batches. For example, several consecutive genes can sometimes end up encoding the same product label because of the crossover, swap mutation or the addition of a new gene. The continuous-time scheduling heuristic combines the consecutive genes with the same product label summing up the number of batches from each gene.
3. *Swap mutation*. Compared to disabling the uniform crossover, the impact of disabling the swap mutation on the mean hypervolume was only slighty more negative. However, the variability in non-dominated solutions from run to run was nearly four times larger.
4. *Modified uniform crossover.* Disabling this genetic operator had negligible impact on the maximum and mean hypervolume values.
5. *Product label mutation*. The GA is capable of varying the product labels through crossover, swap mutation, and the addition of a new gene. Therefore, disabling this operator likely made the overall search process more directed which is also relfected by the improved maximum and mean hypervolume values.

Therefore, the recommendation for selecting the starting parameter values for solving biopharmaceutical scheduling problems similar to the one of this chapter would be to set *pNegB, pPosB,* and *pSwap* high with *pNegB* > *pPosB* > *pSwap* > *0.5* and set *pC* and *pMutP* low (0.0-0.1 range) or disable altogether.

Table 5.7. The impact of disabling (b) crossover, (c) product mutation, (d) positive and (e) negative mutations of the number of batches , and (f) swap mutation on the performance of the multi-objective variable-length GA compared to (a) the base case when the parameters of all genetic operators are set to 0.5.

	a)	**b)**	**c)**	**d)**	**e)**	**f)**
Maximum hypervolume	0.957	0.959	0.994	0.958	0.960	0.946
Mean hypervolume[1]	0.930 ± 0.009	0.923 ± 0.007	0.993 ± 0.003	0.912 ± 0.016	0.844 ± 0.061	0.920 ± 0.026
No. solutions[2]	17	13	37	22	12	18
Run time[3] [s]	29.0	28.9	29.1	29.1	28.9	29.2
No. runs	50					
No. generations	600					
No. chromosomes	1000					
Starting length[4]	1					
pC	0.5	**0**				
pMutP	0.5		**0**			
pPosB	0.5			**0**		
pNegB	0.5				**0**	
pSwap	0.5					**0**

[1] Mean ± 1 standard deviation.
[2] The number of solutions in the best Pareto front.
[3] Average time elapsed for each of the 50 runs.
[4] The starting number of genes per chromosome in the initial population.

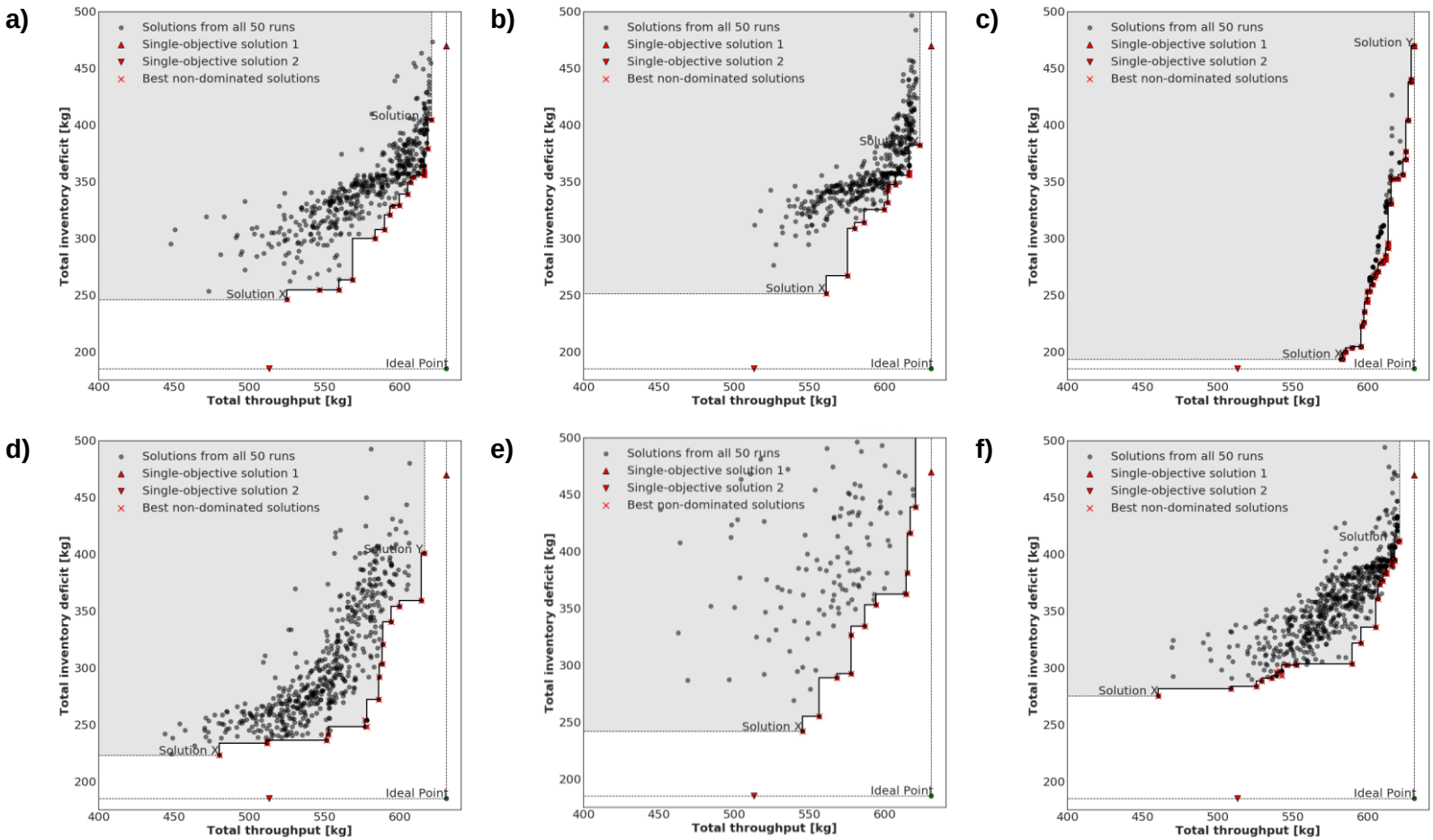


Figure 5.11. The impact of disabling (b) crossover, (c) product mutation, (d) positive and (e) negative mutations of the number of batches , and (f) swap mutation on the performance of the multi-objective variable-length GA compared to (a) the base case when the parameters of all genetic operators are set to 0.5.

5.4.4. The Impact of The Starting Number of Genes

This section evaluates how the starting number of genes in the variable-length chromosome affects the performance of the multi-objective variable-length GA. A total of 5 experiments were performed: when the starting number of genes is 1 (a base case), 3, 6, 9, and 12. Each experiment was performed by running the GA with 1000 chromosomes for 600 generations and 50 runs with all genetic operator parameter values set to 0.5.

According to the results displayed in Figure 5.12 and listed Table 5.8, increasing the starting number of genes does not have a significant positive or negative impact on the maximum and mean hypervolume achieved with the multi-objective variable-length GA. Nevertheless, there was a slight improvement in the performance when the starting number of genes was increased from 1 to 3 (the base case maximum and mean hypervolume increased from 0.957 and 0.930 ± 0.009 to 0.966 and 0.933 ± 0.009) and from 1 to 6 (the base case maximum and mean hypervolume increased from 0.957 and 0.930 ± 0.009 to 0.963 and 0.933 ± 0.009). However, increasing the starting number of genes beyond 6 decreased the performance slightly.

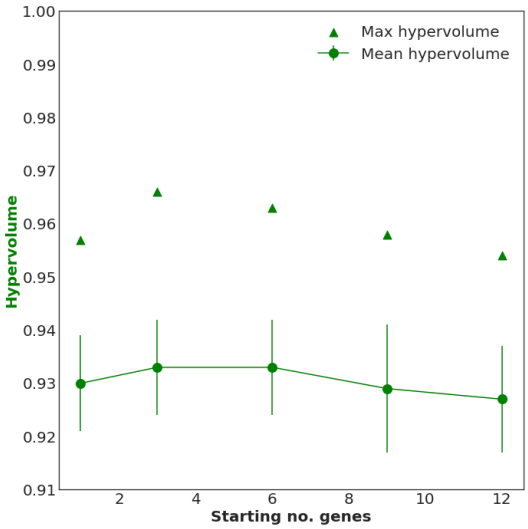


Figure 5.12. The impact of the starting number of genes on the maximum and mean hypervolume. Vertical lines denote the standard deviation of mean hypervolume.

Table 5.8. The impact of the starting number of genes on the maximum and mean hypervolume.

	a)	**b)**	**c)**	**d)**	**e)**
Maximum hypervolume	0.957	0.966	0.963	0.958	0.954
Mean hypervolume[1]	0.930 ± 0.009	0.933 ± 0.009	0.933 ± 0.009	0.929 ± 0.012	0.927 ± 0.01
No. solutions[2]	17	18	18	21	18
Run time[3] [s]	29.9	29.8	29.9	29.9	30.0
No. runs			50		
No. generations			600		
No. chromosomes			1000		
Starting length[4]	1	3	6	9	12
pC			0.5		
pMutP			0.5		
pPosB			0.5		
pNegB			0.5		
pSwap			0.5		

[1] Mean ± 1 standard deviation.
[2] The number of solutions in the best Pareto front.
[3] Average time elapsed for each of the 50 runs.
[4] The starting number of genes per chromosome in the initial population.

5.4.5. Multi-Objective GA Results

This section highlights the advantages of the multi-objective approach for optimising the production schedules of a multi-product biopharmaceutical facility by comparing the boundary solutions *X* and *Y* of the best Pareto front with one another and with the single-objective solutions. The reason for selecting the solutions for comparison from the extreme ends of the Pareto front was to illustrate the trade-off between the two objectives more clearly. As it was discussed earlier, the best Pareto front is generated by re-sorting combined Pareto fronts collected from individual GA runs.

Figure 5.13 displays the total objective space that was determined with a single-objective GA and the best Pareto front generated with a multi-objective GA side-by-side. Table 5.9 provides the details about the Pareto front boundary solutions X and Y.

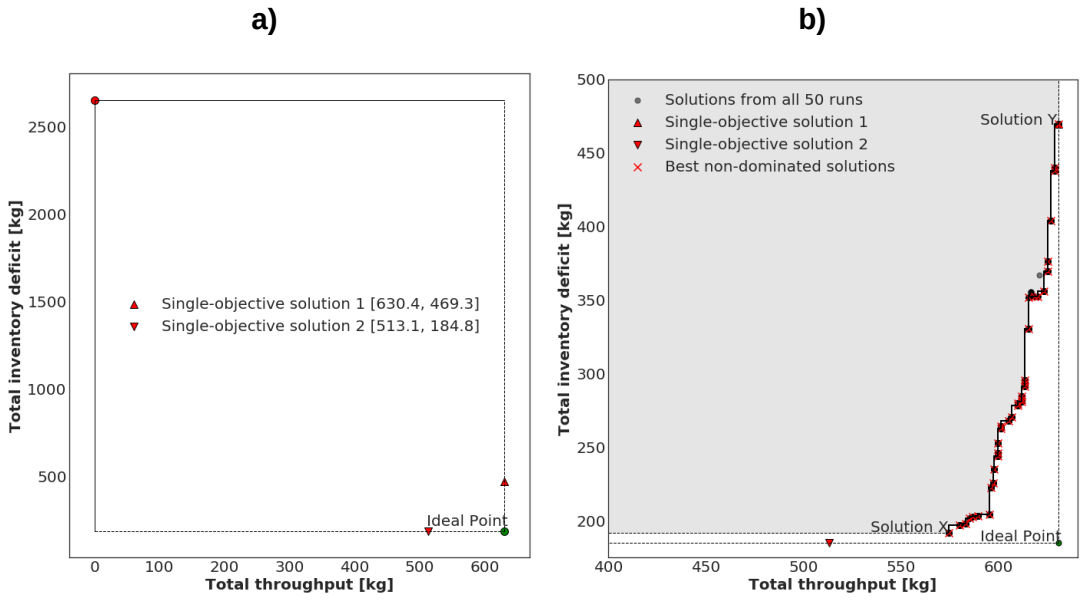


Figure 5.13. Multi-objective optimisation results:
(a) Objective space determined with a single-objective GA.
(b) The best Pareto front (red crosses) and all non-dominated solutions (black circles) collected from individual runs of the multi-objective variable-length GA (maximum and mean hypervolume of 0.994 and 0.944 ± 0.000).

Table 5.9. The boundary solutions X and Y of the best Pareto front generated with the multi-objective variable-length GA.

	Pareto front boundary solution	
	X	**Y**
Total throughput [kg]	574.4	**630.4**
Total inventory deficit [kg]	**191.4**	469.4
Total backlog [kg]	0	0
Total waste [kg]	0	0
Starting length[1]	1	
No. runs	50	
No. generations	600	
No. chromosomes	1000	
pC	0.108	
pMutP	0.000	
pPosB	0.608	
pNegB	0.766	
pSwap	0.471	
Run time[2]	20.0 s	20.0 s

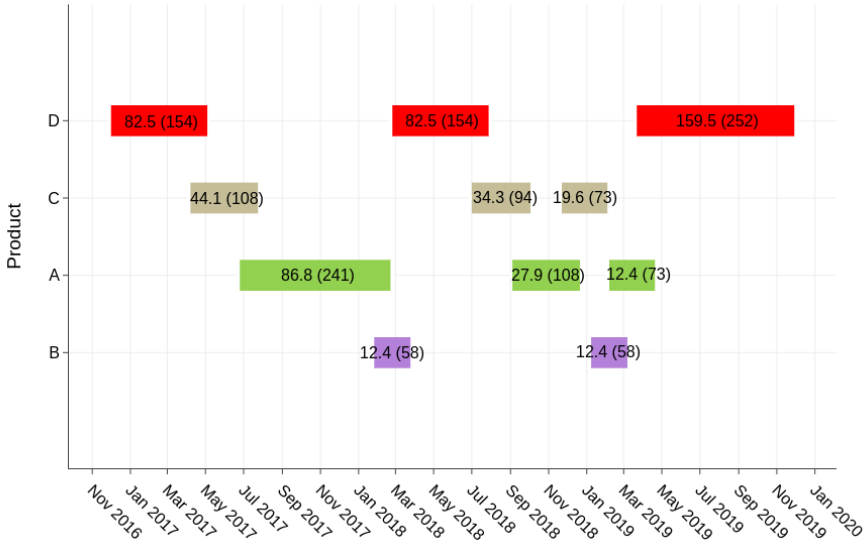
[1] The starting number of genes per chromosome in the initial population.

[2] Mean run time of a single GA run.

According to Figure 5.13 and Table 5.9, the multi-objective variable-length GA is capable of finding solutions which meet all product demands on time and avoid product waste and, at the very least, non-dominate the single-objective solutions. For example, the total inventory deficit of solution X is only slightly larger than that of the single-objective solution 2 (191.3 vs 184.8 kg) but it also has a larger total production throughput (513.1 kg vs 574.4 kg). On the other hand, solution Y matches the single-objective solution 1, i.e. the total production throughput and total inventory deficit are the same for both (630.3 kg and 469.4 kg respectively). The key advantage of the multi-objective GA over the single-objective one is that it provides more options. A total of 36 unique non-dominated solutions were generated. Every production schedule in the best Pareto front offers close-to-optimal (if not optimal) trade-off between maximising the manufacturing capacity of a facility and maintaining a balanced product inventory. A single production schedule can be selected from the non-dominated solutions using, for example, a weighted sum method, Euclidean distance (finding a production schedule that is closest to the ideal point in the objective

space) or using a more sophisticated Monte Carlo simulation-based sensitivity analysis to evaluate the robustness of the schedule to the variations in product demand.

a)



b)

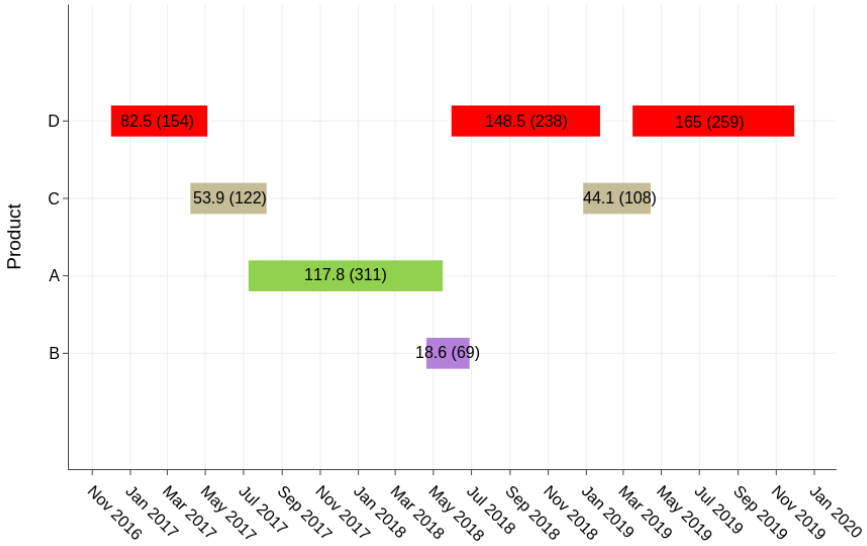


Figure 5.14. Production schedules of (a) solution X and (b) solution Y from the best Pareto front. The numbers in the boxes show how many kilograms are being manufactured, followed by the production time (days).

Figure 5.14 compares the production schedule of solution X with that of solution Y. Every campaign in both solutions has a batch throughput that is within the minimum

and maximum bounds of the corresponding product. The special requirement of product *D* to be produced in multiples of 3 batches has also been met. All campaigns of product *D* in either solution has a number of batches that is evenly divisible by 3. The production schedule of solution X has 11 manufacturing campaigns which are 124-day long and produce 11 batches or 52 kg on average. In contrast, the production schedule of solution Y comprises 7 manufacturing campaigns with an average duration of 180 days, and average throughput of 19 batches and 90 kg. This difference follows the overall pattern of the non-dominated solutions: shorter but more frequent campaigns scheduled appropriately will lead to better balanced product inventory, i.e. lower inventory deficit, but at the cost of lower total production throughput due to more changeovers taking place. In Figure 5.16, the gaps between the strategic inventory targets and the product inventory levels of solution Y are wider and more frequent than those displayed in Figure 5.15 for solution X. The product inventory levels profile of solution X has a more balanced, sawtooth-like pattern, i.e. the inventory tends increase and decrease at a more even rate , compared to those of solution Y. For example, the monthly mean inventory level of product D for solution X is 52.2 ± 20.7 kg; in contrast, the monthly mean inventory level of product D for solution Y is 61.9 ± 41.2 kg.

This comparison illustrates the value of the variable-length chromosome structure for multi-objective scheduling problems. First, it enables the scheduling optimisation to take place in continuous-time. Second, it allows the GA to evolve a set of non-dominated solutions with varying total numbers of production campaigns.

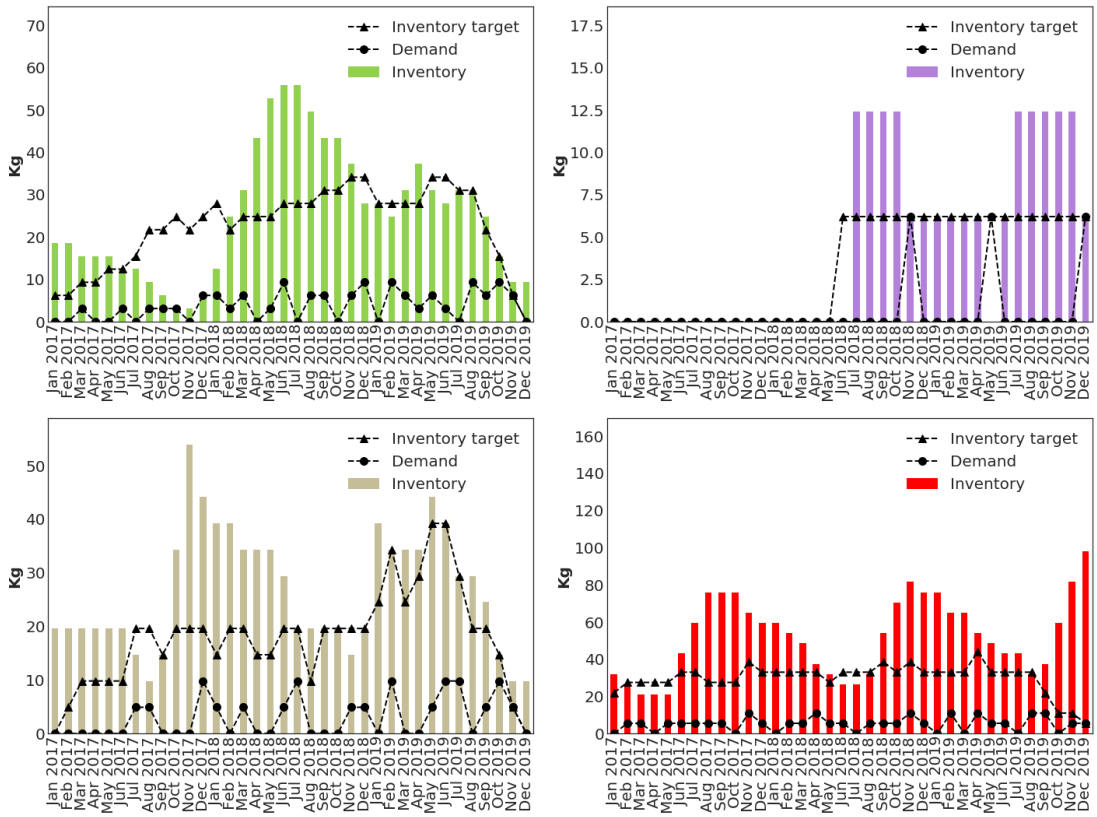


Figure 5.15. Product (A B C D) inventory levels of solution X.

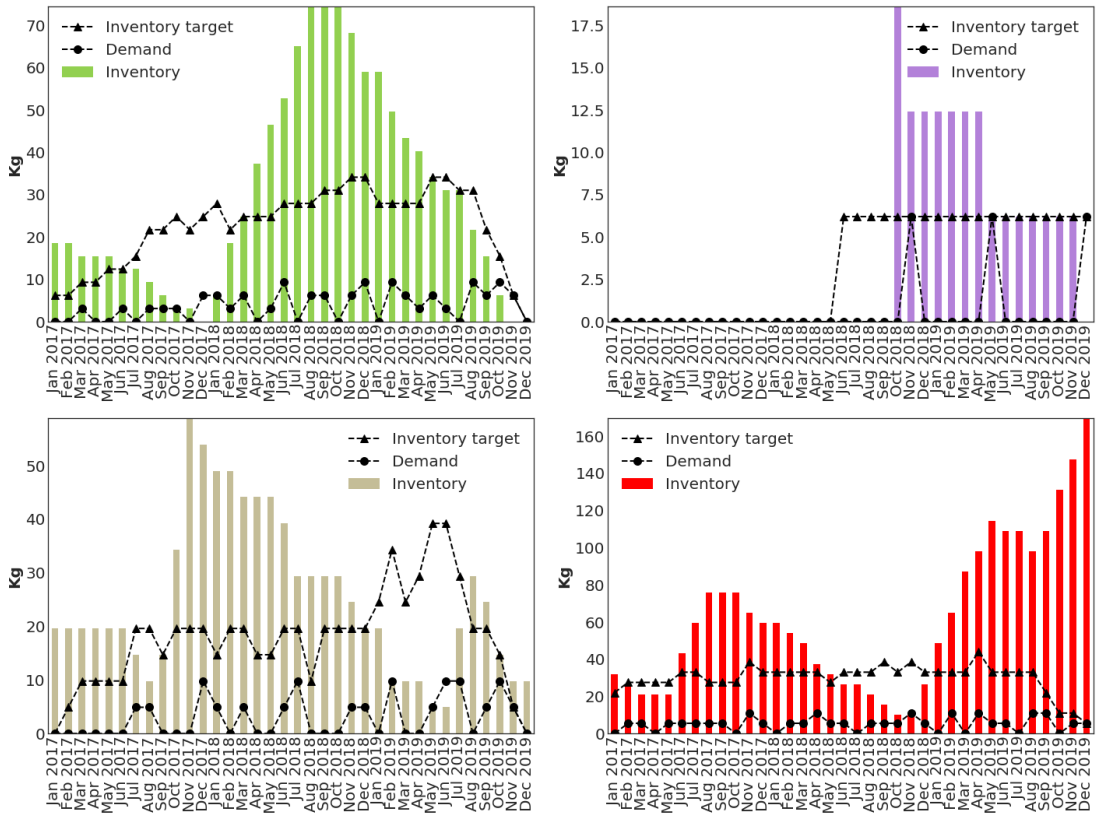


Figure 5.16. Product (A B C D) inventory levels of solution Y.

5.5. Summary

This chapter considered a real-life capacity planning and scheduling problem of multi-product biopharmaceutical manufacture featuring multiple objectives and constraints, product-dependent changeovers, QC/QA checks, and storage and shelf-life limits. An adaptable, variable-length multi-objective GA and a continuous-time scheduling heuristic were adapted from Chapter 4 to tackle the aformentioned scheduling problem. The problem was first solved using a single objective GA to determine the objective space and set a benchmark for the multi-objective optimisation. The variable-length multi-objective GA achieved on average 99.4% of the total objective space hypervolume and generated a *Pareto* front that, at the very least, non-dominated the solutions obtained with a single-objective GA. Furthermore, all solutions met the constraints of the planning problem including the special manufacturing requirements. In the next chapter, the proposed approach will be extended to generate production plans under uncertainty of the biopharmaceutical environment.

6. Multi-Objective Biopharmaceutical Capacity Planning Under Uncertainty

6.1. Introduction

In the previous chapter, the single-objective variable-length GA developed in Chapter 4 was extended with a multi-objective component for continuous-time optimisation of total production throughput and monthly inventory levels of a multi-product biopharmaceutical facility given a 3-year long product demand profile with multiple intermediate due dates. Adding the ability to optimise several objectives simultaneously was shown to be advantageous compared to the single-objective GA-based approach. The multi-objective variable-length GA was used to generate a set of production schedules that not only met all product demands on time without exceeding storage and shelf-life limits but also provided a trade-off between maximising the utilisation of the biopharmaceutical facility's capacity and having a more balanced product inventory. Nevertheless, the presented approach did so deterministically without the consideration for an inherent feature of biopharmaceutical manufacture which is the uncertainty of conditions in this environment.

Meeting product demand in the biopharmaceutical industry is a highly sensitive issue owing to the high value and importance of the products. However, the market demand is often not known in advance and must be estimated. In case the demand uncertainty is neglected during the planning process, the obtained production schedules may be costly or even infeasible. For example, in the 1990s, Wyeth and Immunex (now Amgen) developed Enbrel for the treatment of rheumatoid arthritis. When Enbrel was finally launched in 1998, the demand was higher than what it was anticipated. Even

after increasing volume with their existing Enbrel CMO, both Wyeth and Immunex were unable to satisfy the higher than expected market demand (Kamarck, 2006). Malik et al. (2002) estimated that the lack of manufacturing capacity for the highly successful arthritis drug, Enbrel, cost the company more than $200M in lost revenue in 2001. Therefore, the biopharmaceutical companies must ensure an adequate supply of the product.

Production plans created based on the assumption that the average product demand scenarios will occur can be flawed. Savage (2002) called this phenomenon *The Flaw of Averages* stating that whenever an average is used to represent an uncertain quantity it ends up distorting the results as it neglects the impact of the inevitable fluctuations. A decision to produce the amount equal to an average product demand will lead to the profit that will be on average less than the profit associated with average demand. Lower-than-average demand will lead to higher inventory costs and increased chance of product waste while greater demand will exceed the capacity of the facility and result in late deliveries. A better way to make plans under demand uncertainty is by utilising Monte Carlo simulation which can be used to generate hundreds of demand scenarios based on the whole range of possible values and their likelihood of occurring.

The term Monte Carlo simulation (or method) was coined by Metropolis and Ulam (1949) in reference to games of chance, a popular attraction in Monte Carlo, Monaco. It was a codename for the simulations performed during the 1930s and 1940s to estimate the probability that the chain reaction needed for an atom bomb to detonate would be successful. The key idea behind Monte Carlo simulation is to use randomness by generating draws from a probability distribution. Monte Carlo simulation performs risk analysis by building models of possible results by substituting a probability distribution for any factor that has inherent uncertainty. It then calculates

results over and over, each time using a different set of random values from the probability functions. Depending upon the number of uncertainties and the ranges specified for them, a Monte Carlo simulation could involve thousands or tens of thousands of recalculations before it is complete. Monte Carlo simulation produces distributions of possible outcome values.

In this chapter, the multi-objective variable-length GA from the previous chapter is extended with a Monte Carlo simulation component to generate medium-term production schedules that are robust to the variations in product demand. For the sake of brevity, the integrated Monte Carlo simulation and multi-objective GA approach will be referred to as the *stochastic GA* while the multi-objective GA without Monte Carlo simulation will be referred to as the *deterministic GA*. The advantages of the stochastic GA over the deterministic one will be demonstrated by comparing the production schedules generated when the uncertainty in demand is ignored by using only the most likely demand values and when it is accounted for by characterising it with a probability distribution.

The chapter is organised as follows: **Section 6.2** contains the input data and the definition of the biopharmaceutical scheduling problem with uncertain product demand. **Section 6.3** describes how Monte Carlo simulation is integrated with the multi-objective variable-length GA presented in the previous chapter and how the combined approach is used to generate production schedules under the product demand uncertainty. Additionally, the section explains how the stochastic GA is made more efficient by accelerating the computationally expensive Monte Carlo simulations using GPU resources. The results and discussion are given in **Section 6.4.** Similarly to Section 5.4.1, **Section 6.4.1** first defines the stochastic objective space and then presents the best Pareto front generated using a stochastic multi-objective GA. The trade-offs between the boundary solutions X and Y of the best Pareto front are

explored in **Section 6.4.2**. **Section 6.4.3** shows the impact of neglecting the uncertainty in product demand by comparing the production schedules generated using the stochastic GA (GA with Monte Carlo simulation embedded in the optimisation) and deterministic GA. Deterministic GA outcomes were tested with Monte Carlo simulation post-optimisation.

6.2. Problem Definition

The scheduling problem from the previous chapter has been adapted to demonstrate the features of the integrated multi-objective variable-length GA and Monte Carlo simulation approach. For completeness, the problem statement is as follows:

- *Given:*
  - A start date (1-Dec-2016) and a planning horizon of 3 years
  - A set of biopharmaceutical products *{ A, B, C, D }*
  - *USP* and *DSP* processing times
  - Product-dependent manufacturing yields
  - Product sequence-dependent changeovers
  - Varying amounts of product stock available at the beginning of the schedule
  - Desired minimum and maximum number of batches per individual product campaign
  - Unique manufacturing requirements to produce the batches in multiples of a specified number
  - QC/QA approval times
  - 3-year profile of strategic product inventory targets
  - 3-year profile of uncertain monthly product demand
- *Determine:*
  - A set of production schedules
  - The number and length of manufacturing campaigns
  - Production quantities along with inventory and late delivery profiles
- *So as to (constrained stochastic multi-objective problem):*
  - Maximise the total production throughput

  - Minimise the median total inventory deficit, i.e. cumulative differences between the monthly product inventory levels and the strategic inventory targets
- *Subject to:*
  - The median total backlog being no greater than 0 kg

It is assumed that the biopharmaceutical facility is available during the entire 3-year (1096-day) period. The product demand is assumed to be due on the first day of each month. The products must undergo a 90-day QC/QA process before they can be delivered which must be taken into consideration when meeting the product demand. Product sequence-dependent changeover time (Table 6.1) is incurred only when there is a switch between different product campaigns. Each product has a different manufacturing yield which determines how many kilograms are produced in a single batch. Due to the QC/QA approval process, there is a certain amount of product stock made available at the beginning of the schedule to meet the product demand during the first 90 days. The complete process data for the industrial case study is provided in Table 6.2. The strategic product inventory monthly targets are listed in Table 6.3.

In the last chapter, one of the objectives was to minimise the total inventory deficit which was defined as the cumulative sum of the differences between the product inventory levels and the corresponding strategic monthly targets whenever the latter were greater than the former. In this chapter, the product demand is characterised by a triangular probability distribution based on the specifications of minimum, maximum, and most likely amounts for each due date (see Table 6.4 and Figure 6.1). Therefore, the total inventory deficit and total backlog will have a corresponding distribution of different values depending on the randomly generated product demand scenarios during Monte Carlo simulation.

The goal of the stochastic GA is to generate a set of schedules that are the most robust to the variations in product demand, e.g. with a high probability of meeting all product demands on time. This is accomplished by maximising the total production throughput and minimising the median total inventory deficit subject to the median total backlog being no greater than 0 kg. The objective of the total production throughput maximisation remains unchanged from the previous chapter as the throughput from each individual manufacturing campaign is the same regardless of the product demand scenario.

Table 6.1. Product sequence-dependent changeovers [days].

		To product			
		A	**B**	**C**	**D**
From product	**A**	0	10	16	20
	B	16	0	16	20
	C	16	10	0	20
	D	18	10	18	0

Table 6.2. Process data for the industrial case study.

	Product			
	A	**B**	**C**	**D**
USP duration [days]	45	36	45	49
DSP duration [days]	7	11	7	7
QC/QA duration	90	90	90	90
Yield per batch [kg]	3.1	6.2	4.9	5.5
Opening stock [kg]	18.6	0	19.6	110
Minimum batch throughput per campaign	2	2	2	3
Maximum batch throughput per campaign	50	50	50	30
Produce batches per campaign in multiples of	1	1	1	3

Due to the skewness of product demand distributions (Figure 6.1) and the expected non-symmetrical distributions of the stochastic multi-objective optimisation outcomes, median was chosen as a measure of central tendency. Later in this chapter, non-parametric statistical tests are applied to analyse the stochastic optimisation results and compare them with the results from the deterministic optimisation.

Table 6.3. Strategic inventory targets.

	Product			
Due date	**A**	**B**	**C**	**D**
1-Jan-17	6.2	0	0	22
1-Feb-17	6.2	0	4.9	27.5
1-Mar-17	9.3	0	9.8	27.5
1-Apr-17	9.3	0	9.8	27.5
1-May-17	12.4	0	9.8	27.5
1-Jun-17	12.4	0	9.8	33
1-Jul-17	15.5	0	19.6	33
1-Aug-17	21.7	0	19.6	27.5
1-Sep-17	21.7	0	14.7	27.5
1-Oct-17	24.8	0	19.6	27.5
1-Nov-17	21.7	0	19.6	38.5
1-Dec-17	24.8	0	19.6	33
1-Jan-18	27.9	0	14.7	33
1-Feb-18	21.7	0	19.6	33
1-Mar-18	24.8	0	19.6	33
1-Apr-18	24.8	0	14.7	33
1-May-18	24.8	0	14.7	27.5
1-Jun-18	27.9	6.2	19.6	33
1-Jul-18	27.9	6.2	19.6	33
1-Aug-18	27.9	6.2	9.8	33
1-Sep-18	31	6.2	19.6	38.5
1-Oct-18	31	6.2	19.6	33
1-Nov-18	34.1	6.2	19.6	38.5
1-Dec-18	34.1	6.2	19.6	33
1-Jan-19	27.9	6.2	24.5	33
1-Feb-19	27.9	6.2	34.3	33
1-Mar-19	27.9	6.2	24.5	33
1-Apr-19	27.9	6.2	29.4	44
1-May-19	34.1	6.2	39.2	33
1-Jun-19	34.1	6.2	39.2	33
1-Jul-19	31	6.2	29.4	33
1-Aug-19	31	6.2	19.6	33
1-Sep-19	21.7	6.2	19.6	22
1-Oct-19	15.5	6.2	14.7	11
1-Nov-19	6.2	6.2	4.9	11
1-Dec-19	0	6.2	0	5.5

Table 6.4. Product demand uncertainty for a 3-year period.

	Product			
Due date	**A**	**B**	**C**	**D**
1-Jan-17	0	0	0	0
1-Feb-17	0	0	0	Tr(4.5, 5.5, 8.25)
1-Mar-17	Tr(2.1, 3.1, 4.65)	0	0	Tr(4.5, 5.5, 8.25)
1-Apr-17	0	0	0	0
1-May-17	0	0	0	Tr(4.5, 5.5, 8.25)
1-Jun-17	Tr(2.1, 3.1, 4.65)	0	0	Tr(4.5, 5.5, 8.25)
1-Jul-17	0	0	Tr(3.9, 4.9, 7.35)	Tr(4.5, 5.5, 8.25)
1-Aug-17	Tr(2.1, 3.1, 4.65)	0	Tr(3.9, 4.9, 7.35)	Tr(4.5, 5.5, 8.25)
1-Sep-17	Tr(2.1, 3.1, 4.65)	0	0	Tr(4.5, 5.5, 8.25)
1-Oct-17	Tr(2.1, 3.1, 4.65)	0	0	0
1-Nov-17	0	0	0	Tr(10, 11, 16.5)
1-Dec-17	Tr(5.2, 6.2, 9.3)	0	Tr(8.8, 9.8, 14.7)	Tr(4.5, 5.5, 8.25)
1-Jan-18	Tr(5.2, 6.2, 9.3)	0	Tr(3.9, 4.9, 7.35)	0
1-Feb-18	Tr(2.1, 3.1, 4.65)	0	0	Tr(4.5, 5.5, 8.25)
1-Mar-18	Tr(5.2, 6.2, 9.3)	0	Tr(3.9, 4.9, 7.35)	Tr(4.5, 5.5, 8.25)
1-Apr-18	0	0	0	Tr(10, 11, 16.5)
1-May-18	Tr(2.1, 3.1, 4.65)	0	0	Tr(4.5, 5.5, 8.25)
1-Jun-18	Tr(8.3, 9.3, 13.95)	0	Tr(3.9, 4.9, 7.35)	Tr(4.5, 5.5, 8.25)
1-Jul-18	0	0	Tr(8.8, 9.8, 14.7)	0
1-Aug-18	Tr(5.2, 6.2, 9.3)	0	0	Tr(4.5, 5.5, 8.25)
1-Sep-18	Tr(5.2, 6.2, 9.3)	0	0	Tr(4.5, 5.5, 8.25)
1-Oct-18	0	0	0	Tr(4.5, 5.5, 8.25)
1-Nov-18	Tr(5.2, 6.2, 9.3)	Tr(5.2, 6.2, 9.3)	Tr(3.9, 4.9, 7.35)	Tr(10, 11, 16.5)
1-Dec-18	Tr(8.3, 9.3, 13.95)	0	Tr(3.9, 4.9, 7.35)	Tr(4.5, 5.5, 8.25)
1-Jan-19	0	0	0	0
1-Feb-19	Tr(8.3, 9.3, 13.95)	0	Tr(8.8, 9.8, 14.7)	Tr(10, 11, 16.5)
1-Mar-19	Tr(5.2, 6.2, 9.3)	0	0	0
1-Apr-19	Tr(2.1, 3.1, 4.65)	0	0	Tr(10, 11, 16.5)
1-May-19	Tr(5.2, 6.2, 9.3)	Tr(5.2, 6.2, 9.3)	Tr(3.9, 4.9, 7.35)	Tr(4.5, 5.5, 8.25)
1-Jun-19	Tr(2.1, 3.1, 4.65)	0	Tr(8.8, 9.8, 14.7)	Tr(4.5, 5.5, 8.25)
1-Jul-19	0	0	Tr(8.8, 9.8, 14.7)	0
1-Aug-19	Tr(8.3, 9.3, 13.95)	0	0	Tr(10, 11, 16.5)
1-Sep-19	Tr(5.2, 6.2, 9.3)	0	Tr(3.9, 4.9, 7.35)	Tr(10, 11, 16.5)
1-Oct-19	Tr(8.3, 9.3, 13.95)	0	Tr(8.8, 9.8, 14.7)	0
1-Nov-19	Tr(5.2, 6.2, 9.3)	0	Tr(3.9, 4.9, 7.35)	Tr(4.5, 5.5, 8.25)
1-Dec-19	0	Tr(5.2, 6.2, 9.3)	0	Tr(4.5, 5.5, 8.25)

Note: Tr(x, y, z) denotes a triangular distribution where x, y, and z are the minimum, mode (most likely), and maximum values.

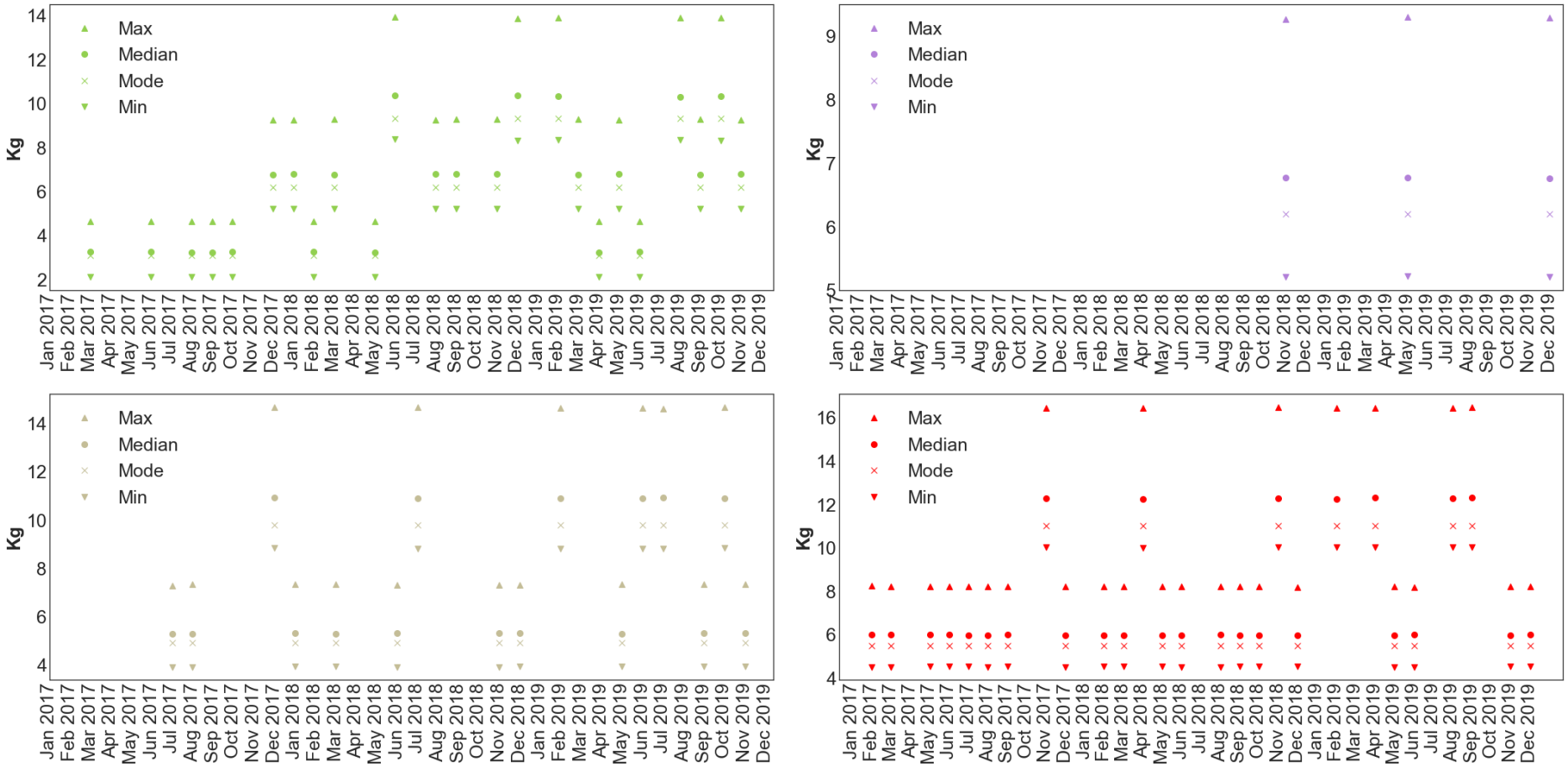


Figure 6.1. Min, median, and max product (A B C D) demand values for each due date after 1,000 Monte Carlo simulation trials using the corresponding triangular distribution from Table 6.4 as an input.

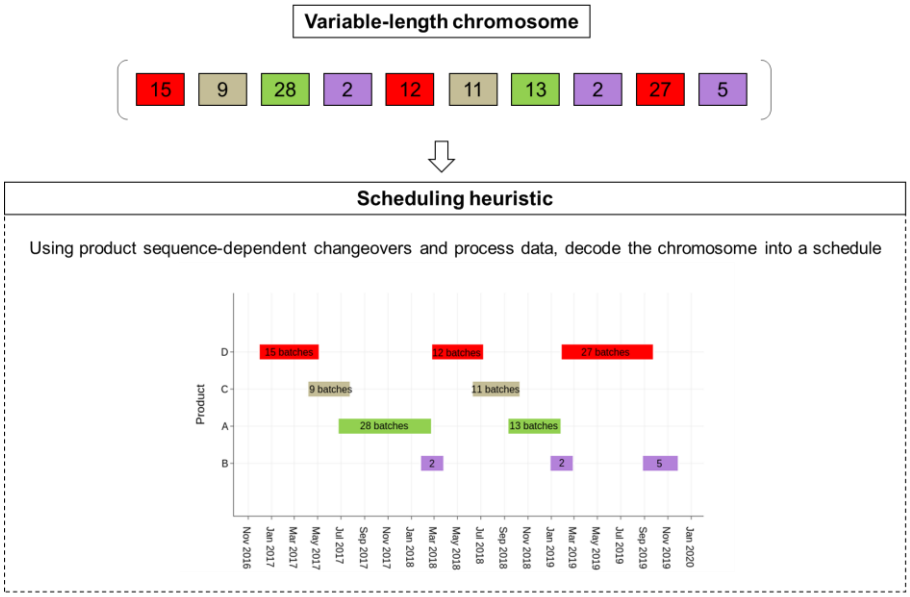
6.3. Methods

The multi-objective variable-length GA and the scheduling heuristic have been implemented in C++ programming language and compiled with MSVC14 compiler to run on a CPU. The Monte Carlo simulation component was developed using C++ and CUDA 8.0 API and compiled with NVCC v8.0 compiler to run on a GPU. The industrially-relevant capacity planning and scheduling problem of medium-term multi-product biopharmaceutical manufacture under uncertainty has been solved on Intel i5-6500 (CPU) and NVIDIA GTX-1060 (GPU) based Windows 10 system with 16GB of RAM and 6 GB of VRAM.

The chromosome encoding strategy, genetic operators, NSGA-II based multi-objective optimisation, constraint handling, and the scheduling heuristic remain largely unchanged from the previous chapter. Therefore, for the sake of brevity, the focus of this section is placed on the implementation details of Monte Carlo simulation integration with the multi-objective GA and steps taken to improve the performance of the stochastic multi-objective GA-based framework.

Figure 6.2 provides a flowchart illustrating of how Monte Carlo simulation fits into the GA-based scheduling optimisation framework from a high-level. First, a continuous-time scheduling heuristic is applied to decode the variable-length chromosomes into production schedules (the heuristic logic has been discussed in Chapters 4 and 5). This is accomplished using the product sequence-dependent changeovers and process data just the same way as it was described in the previous chapter. After the schedule has been constructed, its robustness to the variations in product demand is then tested by conducting Monte Carlo simulation trials. Hundreds of demand scenarios are generated for each individual production schedule based on the provided triangular probability distributions for each product and its every demand due

date. The performance of production schedule, e.g. total inventory deficit, total amount of backlog, is evaluated on each randomly generated demand scenario. For each Monte Carlo simulation trial *t*, the calculated values of total inventory deficit and total amount of backlog are stored in | *t* |-dimensional arrays (see Lines 9 and 10 in Figure 6.2). After the simulation trials are completed, the medians of the total inventory deficit and total backlog distributions are assigned to the corresponding chromosome as the objective and constraints values.



Objective function evaluation

```
1  procedure EvaluateSchedule(chromosome, schedule)
2      chromosome.objective1 = schedule.total_production_throughput
3      deficit_distribution, backlog_distribution = MonteCarloSimulationKernel(schedule)   ▻ executed on a GPU
4      chromosome.objective2 = Median(deficit_distribution)
5      chromosome.constraints = Median(backlog_distribution)
6  end procedure
7
8  procedure MonteCarloSimulationKernel(schedule)
9      deficit_distribution ∈ ℝ^|t|                    ▻ used to store total inventory deficit value after each trial
10     backlog_distribution ∈ ℝ^|t|                    ▻ used to store total backlog value after each trial
11     for each simulation trial t in parallel
12         D = CreateDemandScenario(Tr)
13         Build inventory and backlog profiles for D
14         deficit_distribution_t = calculate total inventory deficit
15         backlog_distribution_t = calculate total backlog
16     end for
17     return deficit_distribution, backlog_distribution
18 end procedure
19
20 procedure CreateDemandScenario(Tr)
21     D ∈ ℝ^(|p|×|d|)                                 ▻ create an empty demand array to store randomly sampled values
22     for each product p
23         for each demand due date d
24             D_pd = random number drawn from the corresponding probability distribution Tr_pd
25         end for
26     end for
27     return D
28 end procedure
```

Figure 6.2. Objective function evaluation of the chromosome using the continuous-time scheduling heuristic and Monte Carlo simulation.

One the major drawbacks of Monte Carlo simulation is the associated computational overhead. In this chapter, 1000 Monte Carlo simulation trials were applied for the evaluation of each chromosome and the impact of this on the execution performance can be seen in Figure 6.3.a. The average time elapsed for a single run of stochastic GA with Monte Carlo simulation embedded into the optimisation was approximately 100-fold longer than that of a deterministic GA without Monte Carlo simulation. Reducing the number of Monte Carlo simulation trials to improve the performance is not ideal as the error of the simulation estimates is inversely proportional to the number of trials. The larger the number of trials is, the more confident the estimates are. Hence, it was necessary to find a way to improve the performance, i.e. execution speed, without sacrificing the accuracy and confidence of the results.

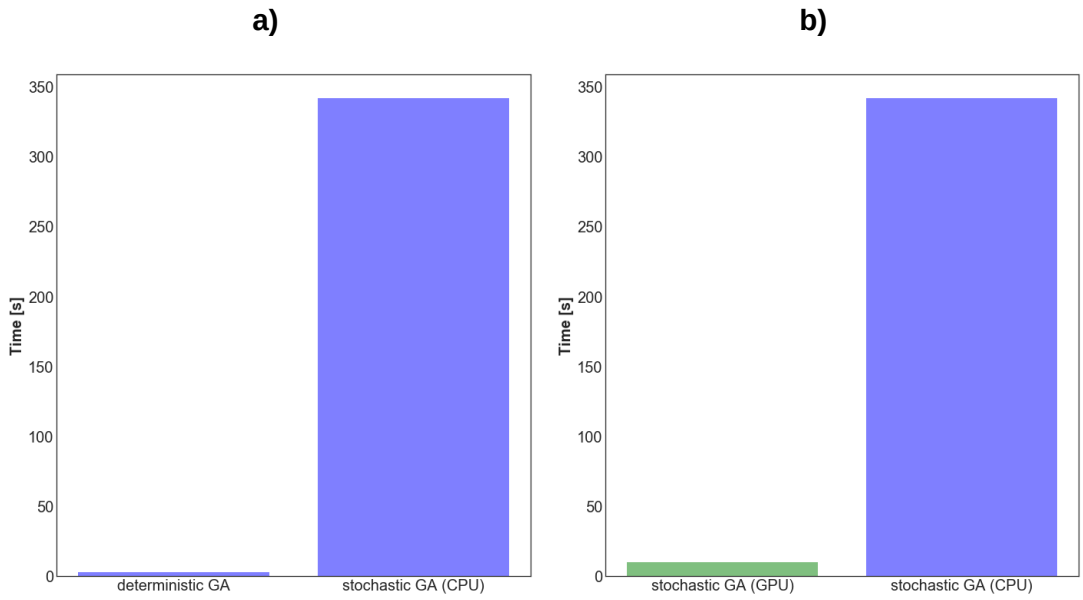


Figure 6.3. Average elapsed time for each of the 50 GA runs with 100 chromosomes for 1000 generations:
(a) deterministic GA vs. CPU-only stochastic GA
(b) Stochastic GA with Monte Carlo simulation performed on a GPU vs. CPU-only stochastic GA
Note: fitness evaluations deterministic and CPU-only stochastic GAs were performed in parallel

Since the individual Monte Carlo simulation trials are independent form each other in this study, the overall simulation process can be made more efficient through the use

of Single Instruction Multiple Data (SIMD)-based architectures. Modern GPUs are optimised for SIMD type processing with massive parallelism. For example, compared to an average consumer-grade Central Processing Unit (CPU) which typically has from 4 to 8 cores, a single GPU can have over 2000 cores (Vanek et al., 2017). Figure 6.4 illustrates the difference between the high-level architectures of GPU and CPU. Each individual Monte Carlo trial can be assigned to a single core on a GPU thus enabling hundreds of trials to be performed in parallel with substantial savings in computational power and time.

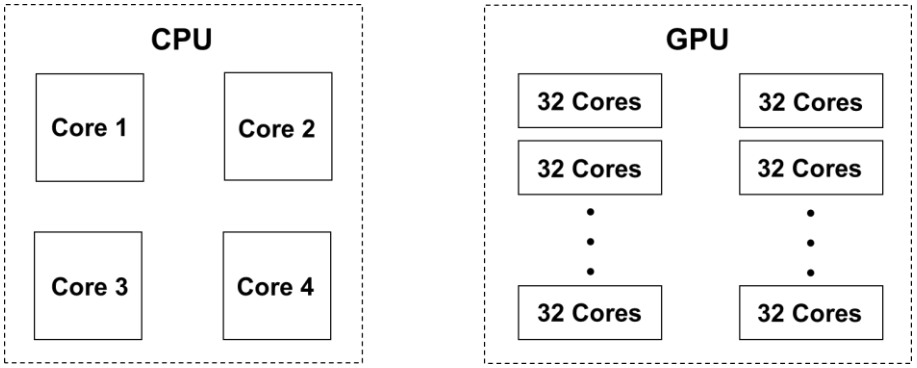


Figure 6.4. Comparison of a high-level architecture between a Central Processing Unit (CPU) and a Graphics Processing Unit (GPU).

In this work, only the Monte Carlo simulation component from the stochastic multi-objective GA-based framework was made to run on a GPU since it was found to be the biggest performance bottleneck compared to other components. The execution of the program was transferred from CPU to GPU every time *MonteCarloSimulationKernel* (see Lines 3, 8-18 in Figure 6.2) was invoked during the objective function evaluation. Once the simulation finished, the execution of the program was transferred back to CPU to continue running the GA.

Accelerating Monte Carlo simulation with a GPU reduced the mean running time of a single stochastic GA run by approximately 30 times (see Figure 6.3.b). In other words, in the time it takes to complete a single run of a CPU-only stochastic GA, 30 runs of

a GPU-accelerated stochastic GA could be completed. As it was outlined in the requirements section of Chapter 2, the ability to achieve solutions in a timely manner is very valuable as it would enable the production schedulers to test more scenarios and perform more case studies with different inputs in less amount of time.

6.4. Results

In this section, the validity of *stochastic* multi-objective GA outlined earlier is demonstrated on an industrially-relevant case study of multi-objective biopharmaceutical capacity planning and scheduling. The problem requires to produce a set of optimal 3-year schedules for a multi-product biopharmaceutical facility manufacturing 4 products under uncertain monthly demand. The objectives of the capacity planning and scheduling problem are to maximise the total kilogram throughput, minimise the median total kilogram inventory deficit. The optimisation problem is subject to the constraint of 0 kg median total kilogram backlog.

First, **Section 6.4.1** defines the objective space of the stochastic optimisation problem using a single-objective GA with integrated Monte Carlo simulation The results obtained using a stochastic, multi-objective GA with Monte Carlo simulation embedded into the optimisation are discussed in **Section 6.4.2** by comparing the trade-offs between two non-dominated solutions selected from the extreme ends of the best Pareto front. **Section 6.4.3** strengthens the argument for stochastic optimisation with a comparison of the production schedules generated using the stochastic and deterministic GAs. The schedules generated with the deterministic GA are tested using Monte Carlo simulation post-optimisation to show the impact of optimisation using only the most likely values, ignoring the uncertainty in product demand.

6.4.1. Stochastic Objective Space

In the preceding chapter, a single-objective GA was applied to obtain an ideal point (a combination of the best objective function values) which together with a reference point (a combination of the worst possible objective function values) was used to assume the boundaries of the objective space. Knowing the total hypervolume of the objective space, made it more convenient to gauge the performance of the multi-objective GA using a hypervolume indicator normalised to 0.0-1.0 range (the higher, the better). Moreover, this also made it easier to interpret and compare the different Pareto fronts to one another.

Table 6.5. The best values of each objective (bold) obtained with the stochastic single-objective GA.

	Stochastic single-objective solution	
	1. Maximise total throughput	**2. Minimise median total inventory deficit**
Total throughput [kg]	**602.1**	514.3
Median total inventory deficit [kg]	555.2	**423.1**
Median total backlog [kg]	0.0	0.0
No. Monte Carlo simulation trials[1]	1000	
No. runs	50	
No. generations	1000	
No. chromosomes	100	
Starting length[2]	1	
pC	0.108	
pMutP	0.041	
pPosB	0.608	
pNegB	0.766	
pSwap	0.471	
Run time[3] [s]	8.94	9.13

[1] Number of Monte Carlo simulations for each chromosome evaluation.
[2] The starting number of genes per chromosome in the initial population.
[3] Mean run time of a single GA run

The same methodology is also applied in this chapter. The scheduling problem with uncertain demand is first solved using a single-objective GA that also has Monte Carlo simulation embedded into the objective function evaluation. The highest total production throughput and the lowest median total inventory deficit values are used

to create an ideal point (see Table 6.5) to define the limits of the stochastic objective space for the scheduling problem of this chapter (see Figure 6.5.a).

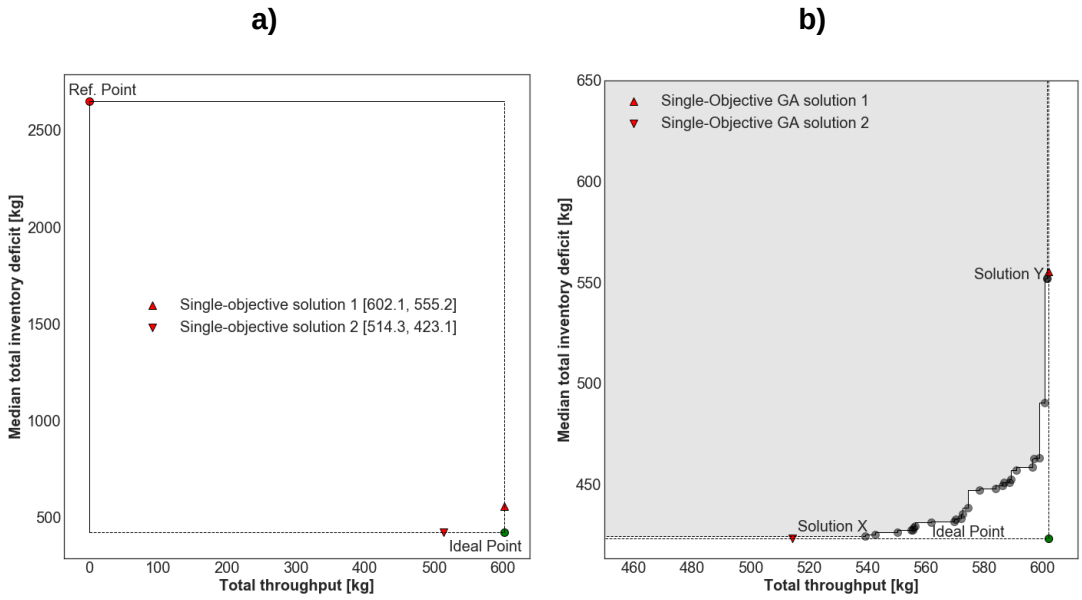


Figure 6.5. (a) Stochastic objective space and (b) the best Pareto front generated using the stochastic multi-objective GA (hypervolume of 0.997). The gray shaded area is used for illustrative purposes to show the area of the objective space that is dominated by the Pareto front solutions.

Table 6.6. Boundary solutions X and Y of the best Pareto front generated using the stochastic multi-objective GA (hypervolume of 0.997).

	Stochastic Pareto solution	
	X	**Y**
Total throughput [kg]	539.3	**601.5**
Median total inventory deficit [kg]	**424.4**	551.7
Median total backlog [kg]	0	0
No. Monte Carlo simulation trials[1]	1000	
No. runs	50	
No. generations	1000	
No. chromosomes	100	
Starting length[2]	1	
pC	0.108	
pMutP	0.041	
pPosB	0.608	
pNegB	0.766	
pSwap	0.471	
Run time[3] [s]	10.81	

[1] Number of Monte Carlo simulations for each chromosome evaluation.
[2] The starting number of genes per chromosome in the initial population.
[3] Mean run time of a single GA run

The goal of the GA in this chapter is to generate a Pareto front of unique non-dominated solutions with the maximum hypervolume value, i.e. as optimal as possible in terms of the specified objectives and constraints. However, as it was described earlier, the GA is an optimisation technique that is not guaranteed to converge on the same solution(s) every time. Therefore, the top Pareto front is saved at the end of each individual GA run. After all 50 runs are completed, the fronts are combined and sorted again using the non-dominated sorting algorithm described by Deb et al. (2002) to create the best Pareto front containing a set of top non-dominated solutions. Such front of non-dominated solution with a hypervolume of 0.997 is displayed in Figure 6.5.b alongside the single-objective solutions and the ideal point.

6.4.2. Stochastic Multi-Objective GA Results

This section compares the solutions X and Y selected from the extreme end of the best Pareto front generated after 50 stochastic GA runs with a population size of 100 for 1000 generations (see Figure 6.5.b and Table 6.6).

The production schedule of solution X (Figure 6.6.a) has a greater number of manufacturing campaigns than solution Y (Figure 6.6.b). The average production time per campaign of solution X is 117 days compared to 161 days for solution Y. Similar to the previous chapter, the model predicted that more frequent but shorter manufacturing campaigns scheduled according to a recurring pattern would lead to better optimised product inventory levels but at the cost of lower total production throughput because of the lost production time due to more frequent product changeovers.

According to the Mann-Whitney U test, the difference between the total inventory deficit distributions of the solutions X and Y (Figure 6.7.a) was found to be statistically

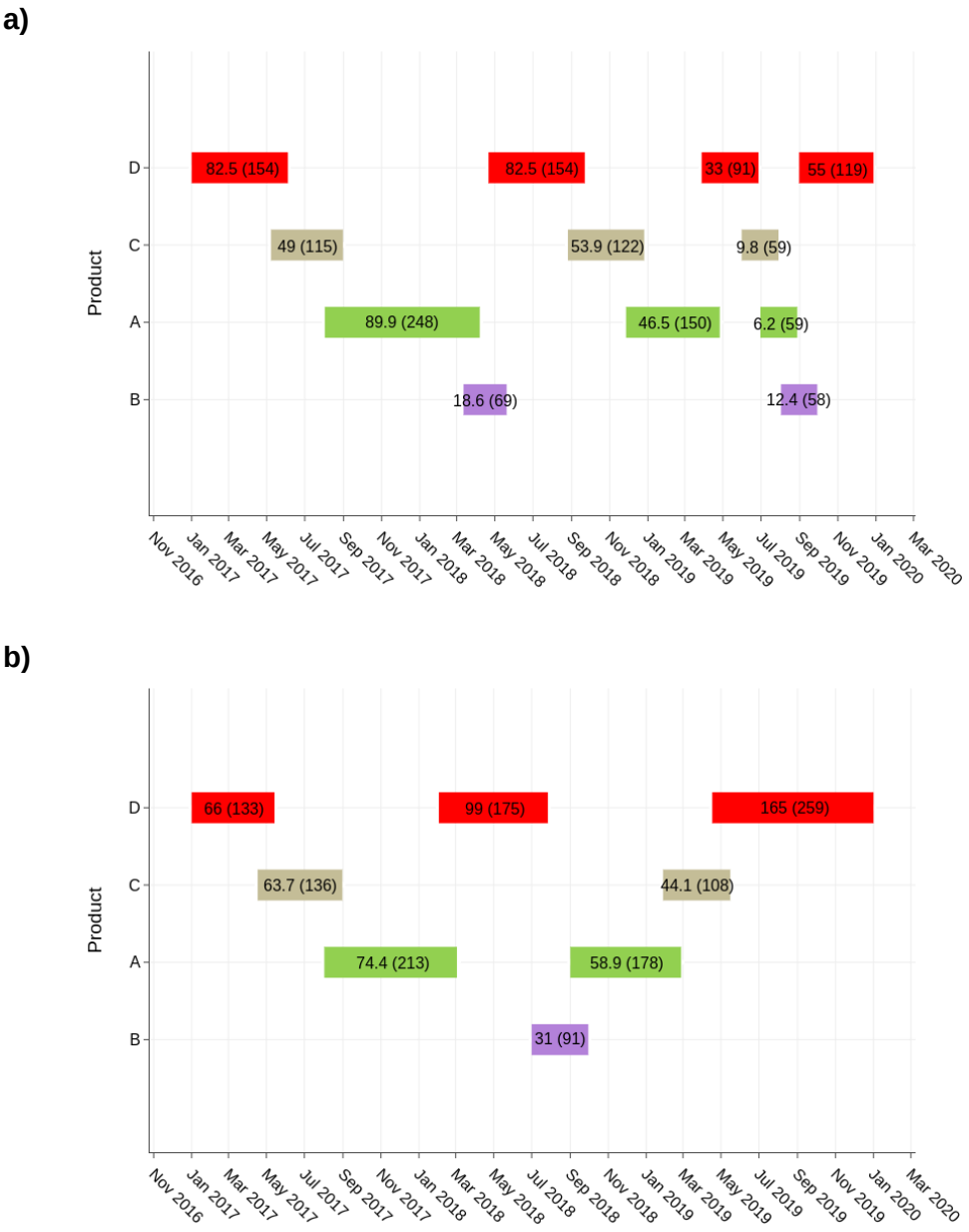


Figure 6.6. Production schedules of (a) solution X and (b) Y from the best Pareto front after 50 runs generated using the stochastic GA. The numbers in the boxes show how many kilograms are being manufactured, followed by the production time (days).

significant with a two-tailed *p* value of 0. Mann-Whitney U test is a non-parametric alternative to independent samples *t-test.* The null hypothesis H_0 of the test is that the probability of a random observation from distribution X exceeding a random observation from distribution Y is the same, i.e. $P(X > Y) == P(Y > X)$. While Mann-Whitney U test helps to evaluate the probability of the effect, it does not reveal any details about its size. However, this can be evaluated by calculating the point estimate

of the Hodges-Lehmann's median difference *Δ* which is equal to the median of all pairwise differences between the two distributions (Hodges Jr & Lehmann, 1963). The value of *Δ* between the total inventory deficit distributions of the solutions X and Y is equal to 126 kg. This value can also be interpreted in the following way: the total inventory deficit of the solution Y is 126 kg higher on average than that of the solution X.

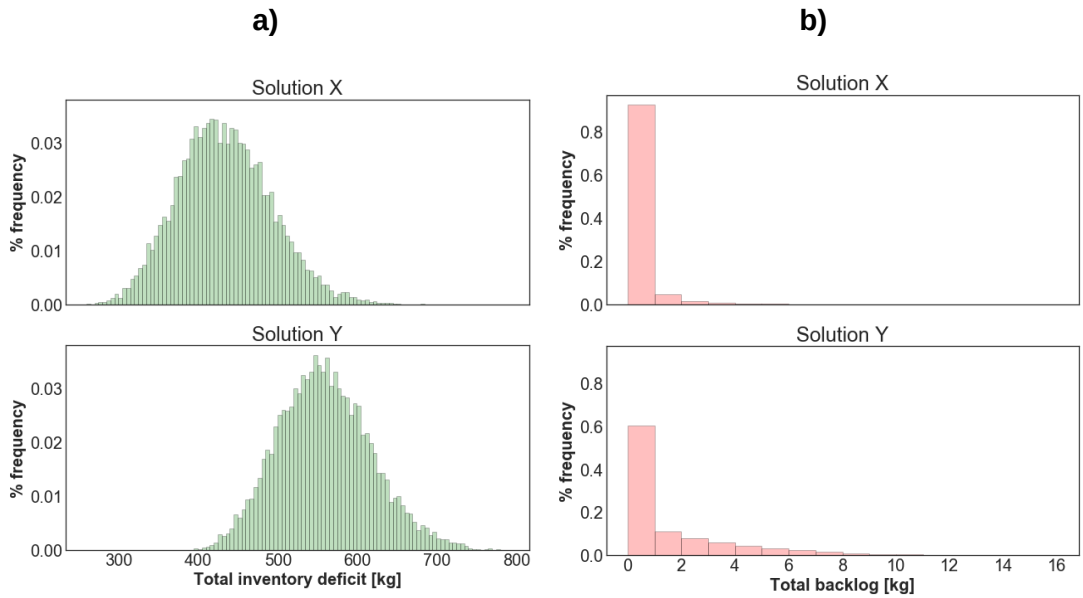


Figure 6.7. Comparison of (a) the total inventory deficit and (b) total backlog distributions between the solutions X and Y from the best Pareto front generated using the stochastic GA.

The median total backlog is equal to 0 kg for both solutions (Table 6.7). Nevertheless, solution X has a greater probability of meeting the product demand compared to the solution Y (0.82 vs 0. 0.50). Using Mann-Whitney U test, the difference between the total backlog distributions in Figure 6.7.b is also found to be statistically significant (two-tailed *p* value of 0) with *Δ* of 0.1 kg.

The results of the stochastic multi-objective GA show that depending on the chosen objectives and constraints, there can be multiple alternative solutions to a scheduling problem even in the presence of uncertainty. The stochastic GA generates a set of equally good alternative production schedules. Depending on the business strategy,

decision-makers can decide whether it is more acceptable to choose a production schedule that would result in higher total throughput but also a higher risk of not being able to meet the inventory targets and product demands on time or vice versa.

Table 6.7. Comparison of the solutions X and Y from the best Pareto front generated using the stochastic GA.

	Stochastic Pareto front solution	
	X	**Y**
Total throughput [kg]	539.3	601.5
Max total backlog [kg]	8.2	16.0
Mean total backlog [kg]	0.2 ± 0.6	7.1 ± 4.3
Median total backlog [kg]	0	0
Min total backlog [kg]	0	0
P(total backlog ≤ 0 kg)	0.82	0.50
Max total inventory deficit [kg]	683.4	786.0
Mean total inventory deficit [kg]	432.6 ± 58.6	558.6 ± 59.0
Median total inventory deficit [kg]	424.4	551.7
Min total inventory deficit [kg]	259.4	355.6

6.4.3. Comparison with the Deterministic GA

This section of the results will discuss the merits of integrating Monte Carlo simulation into the multi-objective variable-length GA for creating production schedules under demand uncertainty. As mentioned earlier, the advantages will be illustrated by comparing the stochastic optimisation results with a deterministic GA-based approach.

The scheduling problem presented in this chapter was solved again but the uncertainty in product demand was ignored and instead of the probability distributions (Table 6.4) only the most likely product demand values were used as an input. First, the objective space was defined using a deterministic single-objective GA without Monte Carlo simulation. Then, a multi-objective GA, also without Monte Carlo

simulation, was used to generate the best Pareto front of deterministic solutions in a similar way that was described in earlier in the last paragraph of Section 6.4.1.

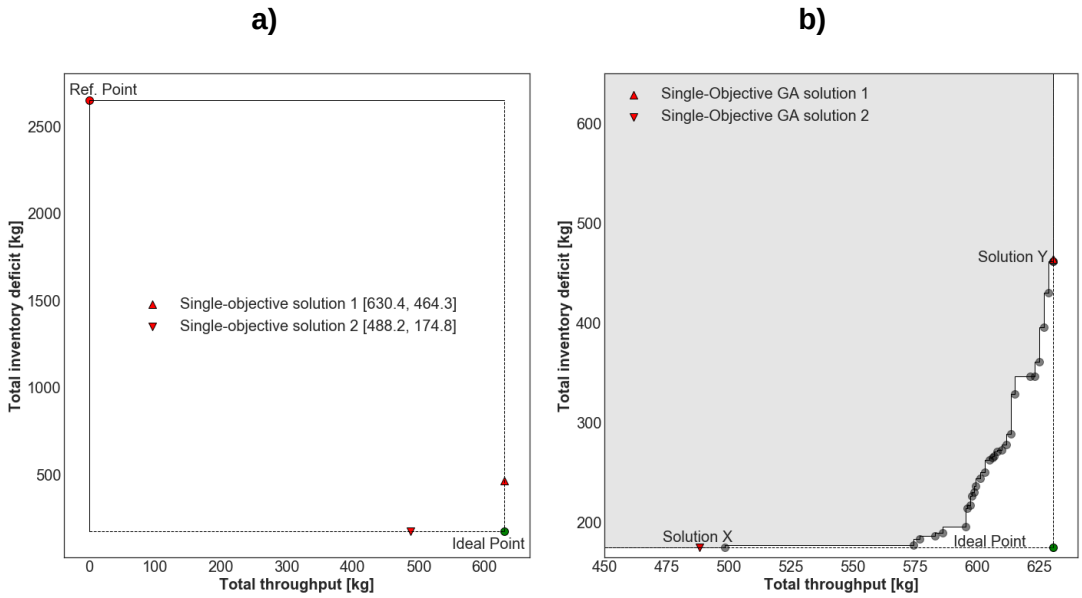


Figure 6.8. (a) Deterministic objective space and (b) the best Pareto front generated using the deterministic multi-objective GA (hypervolume of 0.996).

Table 6.8. The best values of each objective (bold) obtained with a deterministic single objective GA.

	Deterministic single-objective solution	
	1. Maximise total throughput	**2. Minimise total inventory deficit**
Total throughput [kg]	**630.4**	488.2
Total inventory deficit [kg]	464.3	**174.8**
Total backlog [kg]	0	0
No. runs	50	
No. generations	1000	
No. chromosomes	100	
Starting length[1]	1	
pC	0.108	
pMutP	0.041	
pPosB	0.608	
pNegB	0.766	
pSwap	0.471	
Run time[2] [s]	0.78	0.86

[1] The starting number of genes per chromosome in the initial population.

[2] Mean run time of a single GA run

Table 6.8 lists the best single-objective values obtained with a deterministic single-objective GA (without Monte Carlo simulation) whereas Figure 6.8.a shows the

boundaries of the deterministic objective space defined by the reference and ideal points. The solution X (refer to Figure 6.8.b and Table 6.9) from the best Pareto front generated with the *deterministic* multi-objective GA, i.e. without Monte Carlo simulation, is compared to the solution X (see Figure 6.5.b and Table 6.6) from the best Pareto front generated using the *stochastic* GA. For convenience, the two solutions will be referred to as *deterministic* and *stochastic solution* respectively. To be able to compare the deterministic solution with the stochastic one, Monte Carlo simulation is used to conduct a stochastic analysis to assess its robustness to the variability of product demand.

Table 6.9. The boundary solutions X and Y of the best Pareto front generated using the deterministic multi-objective GA without the embedded Monte Carlo simulation-based optimisation.

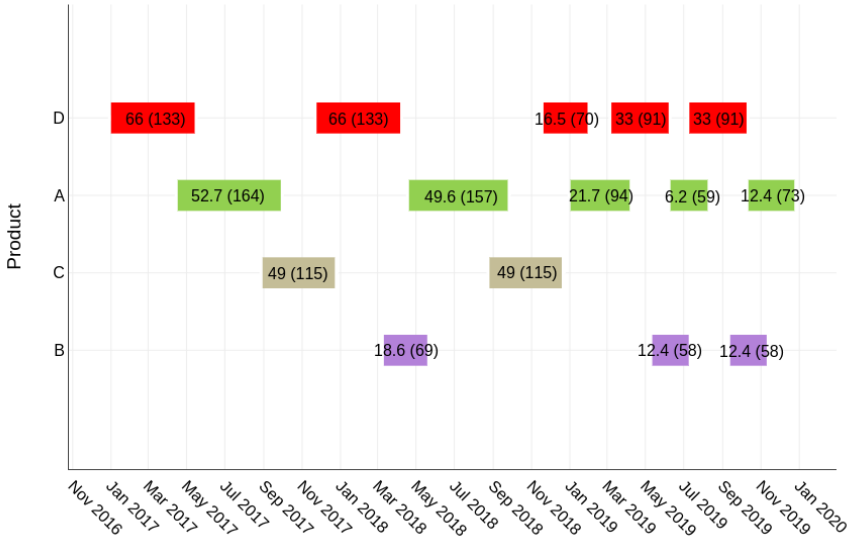
	Deterministic Pareto front solution	
	X	**Y**
Total throughput [kg]	498.5	**630.4**
Total inventory deficit [kg]	**175.4**	461.4
Total backlog [kg]	0	0
No. runs	50	
No. generations	1000	
No. chromosomes	100	
Starting length[1]	1	
pC	0.108	
pMutP	0.041	
pPosB	0.608	
pNegB	0.766	
pSwap	0.471	
Run time[2] [s]	3.07	

[1] The starting number of genes per chromosome in the initial population.
[2] Mean run time of a single GA run

Using only the most likely demand values, the deterministic solution (solution X from Figure 6.8.b and Table 6.9) achieved the total throughput and total inventory deficit values of 498.5 kg and 175.4 kg respectively. The production schedules of the deterministic (Figure 6.9.a) and stochastic solution (Figure 6.9.b) are very similar: both contain short but frequent recurring manufacturing campaigns. Therefore, at the first

glance, it might seem like the production schedule generated deterministically would perform similarly to the production schedule generated using a stochastic GA under uncertain product demand, i.e. have a similar median total inventory deficit and a median total backlog equal to 0 kg.

a)



b)

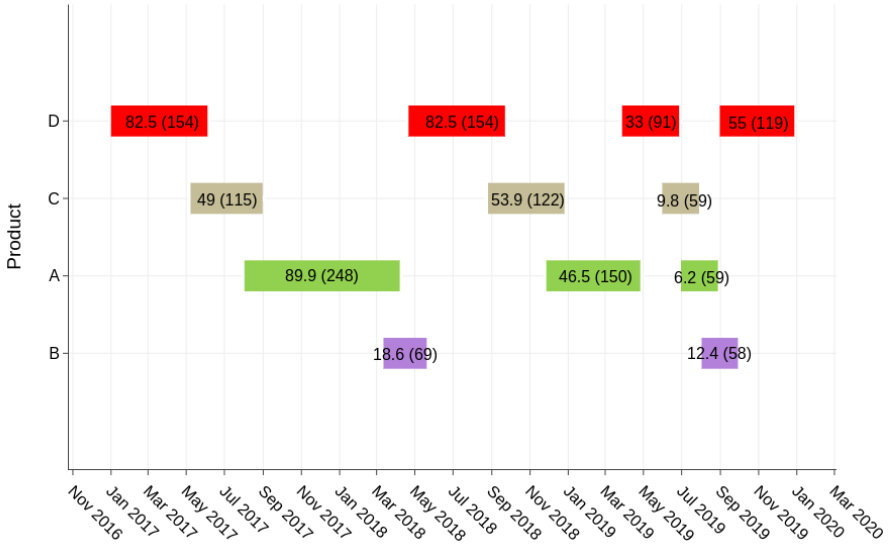


Figure 6.9. Production schedules of (a) the deterministic solution X and (b) stochastic solution X from the respective best Pareto fronts. The numbers in the boxes show how many kilograms are being manufactured, followed by the production time (days).

However, after Monte Carlo simulation was applied post-optimisation to evaluate the robustness of the deterministic solution to the variability of demand, it was found that the corresponding production schedule had a significantly lower probability of meeting product demands on time.

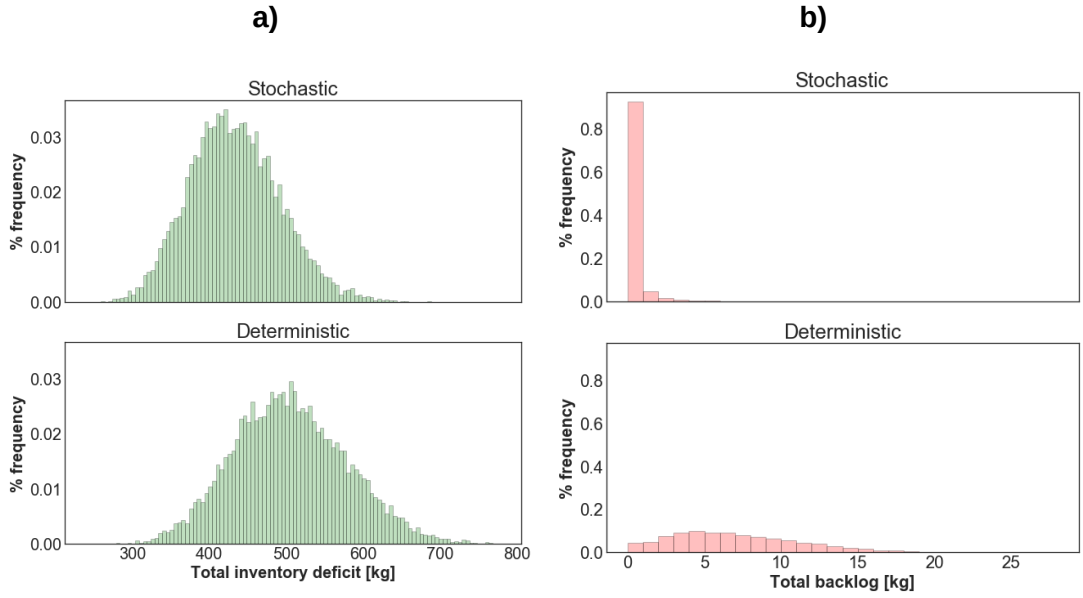


Figure 6.10. A comparison of (a) the total inventory deficit (a) and (b) total backlog distributions between the stochastic and deterministic solutions. after the stochastic analysis with Monte Carlo simulation

Table 6.8. A comparison between the stochastic and the deterministic solutions.

	Solution	
	Stochastic	**Deterministic**
Total throughput [kg]	539.3	498.5
Max total backlog [kg]	8.2	27.1
Mean total backlog [kg]	0.2 ± 0.6	7.1 ± 4.3
Median total backlog [kg]	0	6
Min total backlog [kg]	0	0
P(total backlog ≤ 0 kg)	0.82	0.01
Max total inventory deficit [kg]	683.4	776.5
Mean total inventory deficit [kg]	432.6 ± 58.6	504.8 ± 74.1
Median total inventory deficit [kg]	424.4	501
Min total inventory deficit [kg]	259.4	238.4

Note: the stochastic solution was generated using a multi-objective GA with Monte Carlo simulation embedded into the optimisation of the objectives. The deterministic solution was obtained using a multi-objective GA without the integrated Monte Carlo simulation. Instead, Monte Carlo simulation was used to perform a post-optimisation sensitivity analysis of the solution was performed using it.

Figure 6.10 and Table 6.8 illustrate and list the results of the Monte Carlo simulation-based sensitivity analysis of the deterministic solution and the comparison with the stochastic solution generated using the multi-objective GA with Monte Carlo simulation embedded into the objective function evaluation. The production schedule generated using the deterministic multi-objective GA was capable of meeting all product demands in only 14 randomly generated product demand scenarios out of 1000 in total (1.4%). Moreover, the median total inventory deficit level was also higher (501 kg vs 424.4 kg) than that of the production schedule generated using the stochastic GA with integrated Monte Carlo simulation.

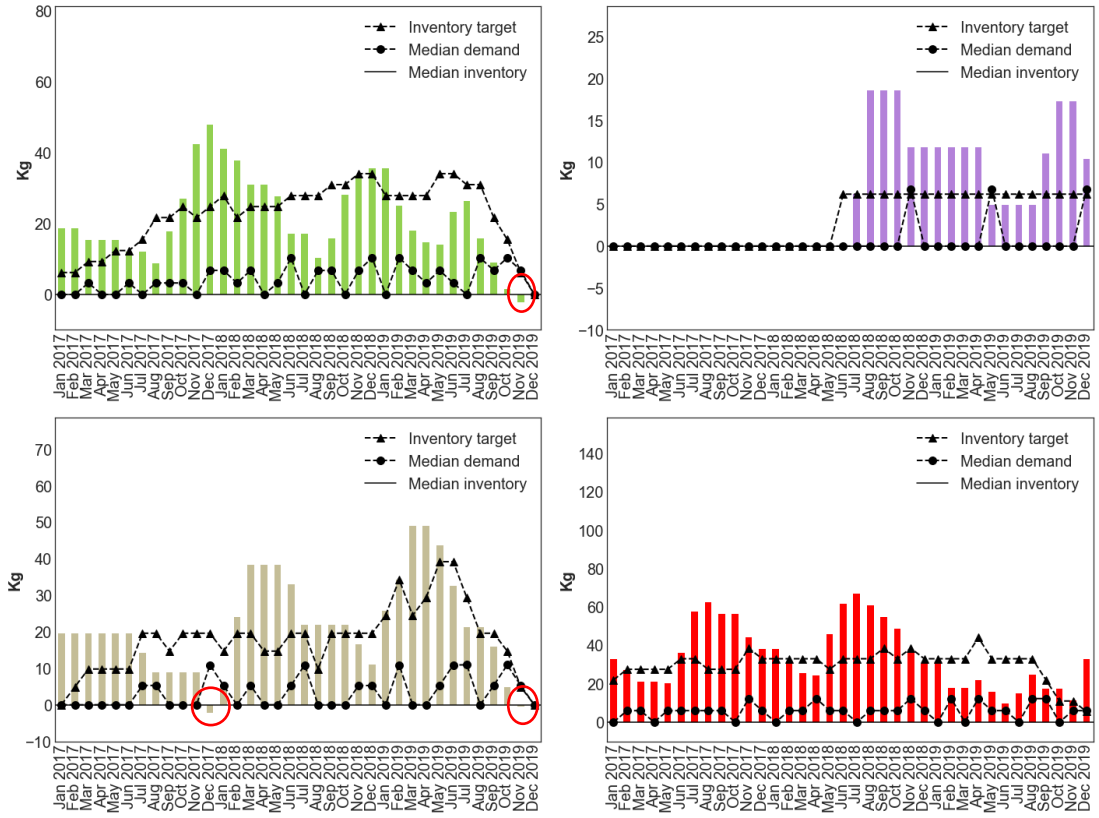


Figure 6.11. Individual product (A B C D) inventory profiles of the deterministic solution after the stochastic analysis with Monte Carlo simulation. The negative inventory levels highlighted by the red ovals indicate the median amount of unmet product demand.

Using the Mann-Whitney U-test, the difference between the total inventory distributions of the deterministic and stochastic solutions is statistically significant

(two-tailed *p* value of 0) with *Δ* of 71.4 kg. The difference between the total backlog distributions is also statistically significant (two-tailed *p* value of 0) with *Δ* of 6.4 kg. More importantly, the deterministic solution has only 0.01 probability of meeting all product demands on time compared to 0.82 probability of the stochastic solution. According to Figure 6.11, the deterministic production schedule is expected to be unable to meet the demand on time for products A and C on 6 separate due dates. In comparison, the stochastic solution meets all product demands on time on average (Figure 6.12).

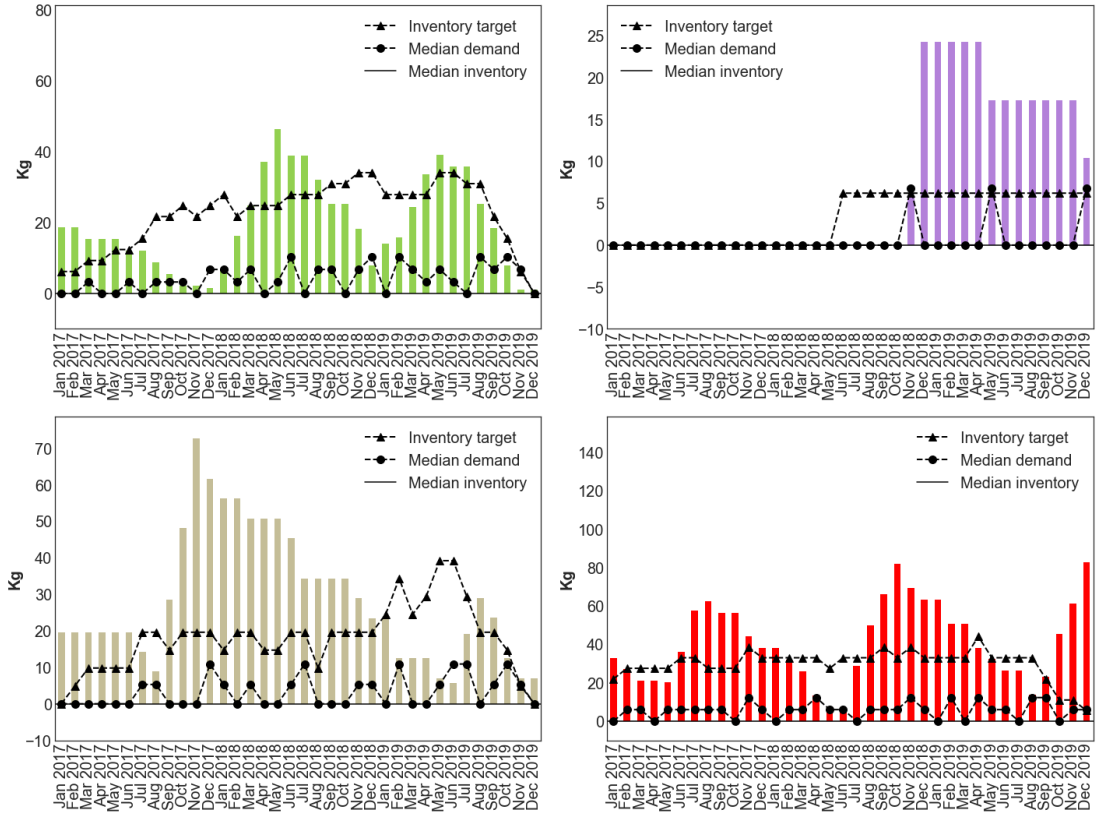


Figure 6.12. Individual product (A B C D) inventory profiles of the stochastic solution.

Based on the comparison between the deterministic and stochastic solutions, the advantages of the stochastic GA are evident. The difference between the total inventory deficit and total backlog distributions of the two solutions is statistically significant. The stochastic solution has a much greater chance of meeting all product

demands on time despite the variations. Moreover, the monthly product inventory levels of the stochastic solution are expected to be closer to the set strategic targets.

6.5. Summary

In this chapter, the novel continuous-time GA-based scheduling optimisation approach described in Chapters 4 and 5 was extended with Monte Carlo simulation to address an inherent and very important feature of biopharmaceutical industry – uncertainty in product demand. The monthly demand for each product was characterised with a triangular distribution defined by the minimum, most likely, and maximum quantities. Integrating Monte Carlo simulation into the multi-objective GA permitted the identification of more robust production schedules better suited to handle product demand fluctuations. The benefits of an integrated GA and Monte Carlo simulation approach were demonstrated by comparing it with a deterministic approach. The production schedules generated with a deterministic GA were based on the most likely demand values and did not account for the variability in product demand. Hence, in scenarios where the product demand was higher than expected the solution was shown as not able to meet all product demands on time on average.

7. Commercialisation

7.1. Introduction

This chapter outlines a plan for potential commercialisation of the work generated during this PhD. A minimum viable product (MVP) has been developed to demonstrate the viability of the commercialisation plan. The following areas are discussed:

- Delivery model – best way to deliver the software so that benefits both the clients and the software developers.
- Architecture – the overall design of the software including database schema, user experience (UX), and user interface (UI).
- Revenue model – strategies for pricing the software based on its features.

7.2. Delivery Model

Traditionally, businesses would buy software, install it and maintain it on their own machines. That software delivery model is giving way to a modern one known as Software as a Service (SaaS).

The concept of SaaS is relatively simple: just like e-mails or social media applications, business applications can also be accessed with a Web browser over the Internet. Instead of buying a license and installing the software on individual machines, a business buys a subscription to use the application and services hosted in a cloud environment. *Cloud computing* has become so popular that it was introduced as a new word in the English language in 2012 (Dutt et al., 2017). According to the forecast by the International Data Corporation (IDC, 2018), worldwide public cloud services

spending is expected to reach $203.4B by 2020 and SaaS solutions are estimated to account for about 60% of this spending.

SaaS model offers multiple benefits to software developers and buyers alike. For buyers, the advantages include easier and more frequent upgrades, lower cost of ownership, and better support from vendors as they have to be more responsive to customers or risk losing subscription revenues (Dubey & Wagle, 2007). Moreover, the investment in SaaS product development tends to be higher which also results in higher software quality compared to perpetual licensing (Choudhary, 2007). For software developers, the benefits of a SaaS approach include reduced deployment time, streamlined software building and testing cycles using, for example, continuous integration (CI) systems, and the ability to monitor software usage which can be used to enhance it. Countering the advantages of SaaS are the risks of reliability and security of the service. For example, some of the biopharmaceutical companies might have concerns about data privacy.

SaaS model is especially attractive for deploying the GA-based DST developed during this PhD. Using high-performance cloud computing environments, it is possible to make the computationally demanding features, e.g. GPU-accelerated integrated Monte Carlo simulation for stochastic optimisation, accessible to all users on virtually any platform (including tablets, mobile phones, different Operating Systems – anything that can run a Web browser). In contrast, using the traditional delivery model, clients would be most likely restricted to one platform and would not be able to access such complex features unless they invest in building the IT infrastructure and buying specialist hardware with high-performance CPUs and GPUs.

Therefore, this chapter proposes to build and deliver the GA-based DST as a SaaS application.

7.3. Architecture

This section explains the proposed architecture of the SaaS solution for capacity and scheduling of biopharmaceutical facilities using snapshots of an actual MVP.

7.3.1. Overview

The GA-based DST has been developed according to most of the requirements and specifications outlined in Chapter 2. The tool comprises the following three core parts:

- High-performance, multi-threaded C++ implementations of the work that was presented in the previous chapters, i.e. GA-based scheduling optimisation algorithms and biopharmaceutical scheduling models.
- Python API which wraps up the C++ components, provides methods for data input and output, results reporting and visualisation, Gantt charts generation, and also makes the integration with other libraries or applications much more streamlined.
- Django (Django, 2018) open source web framework which provides a user-friendly platform to view and manage input data and scheduling optimisation results. There are a number of different web frameworks available, but Django is the most popular Python web framework that encourages rapid development, clean, pragmatic design, and offers a very wide range of features available out-of-the-box, including but not limited to a web server, extensible authentication system, and an object relational mapping (ORM) tool for storing and retrieving data from a relational database.

Figure 7.1 displays a high-level architecture of a SaaS application (GA-based DST) that is hosted in a cloud environment and accessible over the Internet using a Web browser. Django web framework is responsible for handling client's requests using Models, Views, Templates, and a Uniform Resource Locator (URL) Dispatcher,

Models. A URL Dispatcher maps the requested URL to a View function and calls it. For example, if a client requested to enter new or edit the existing data, a URL Dispatcher would map that request to a corresponding View function which would perform one or several Create, Read, Update, and Delete (CRUD) operations in the relational database.

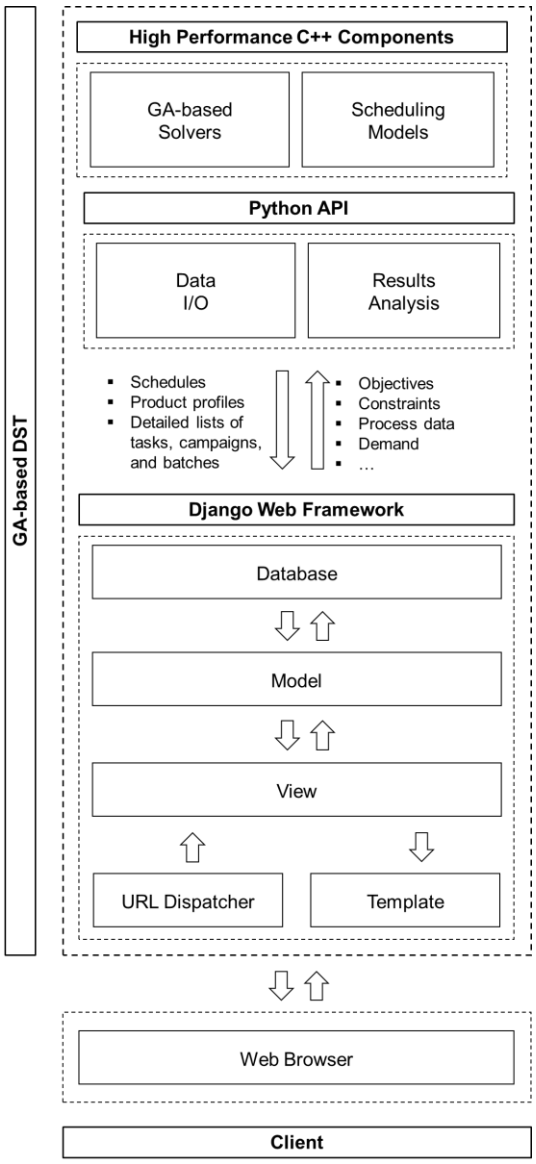


Figure 7.1. High-level architecture of the GA-based Decision Support Tool implemented as a SaaS application.

In an example scenario, if a client requests to create production schedules for a specific facility, the URL Dispatcher would call a matching View function to retrieve

the input data comprised of *Facility*, *Product*, *Changeover*, *Demand*, and *InventoryTarget* tables (Figure 7.2) from the database using *user_id* and *facility_id*. The input data is passed to Python API (see Figure 7.1) which connects the web framework and the high-performance C++ implementations of the variable-length GAs and scheduling models described in the earlier chapters. Python API is responsible not only for transferring the data back and forth from the C++ components but also for formatting, analytics, and reporting the results back to the web framework. After the request is complete, the View function would save the output data in the *Schedule, Campaign, Batch,* and *Inventory* tables (Figure 7.2), create a Template – an HTTP object with tables, figures, and Gantt charts from the scheduling optimisation – and render it using a Web browser.

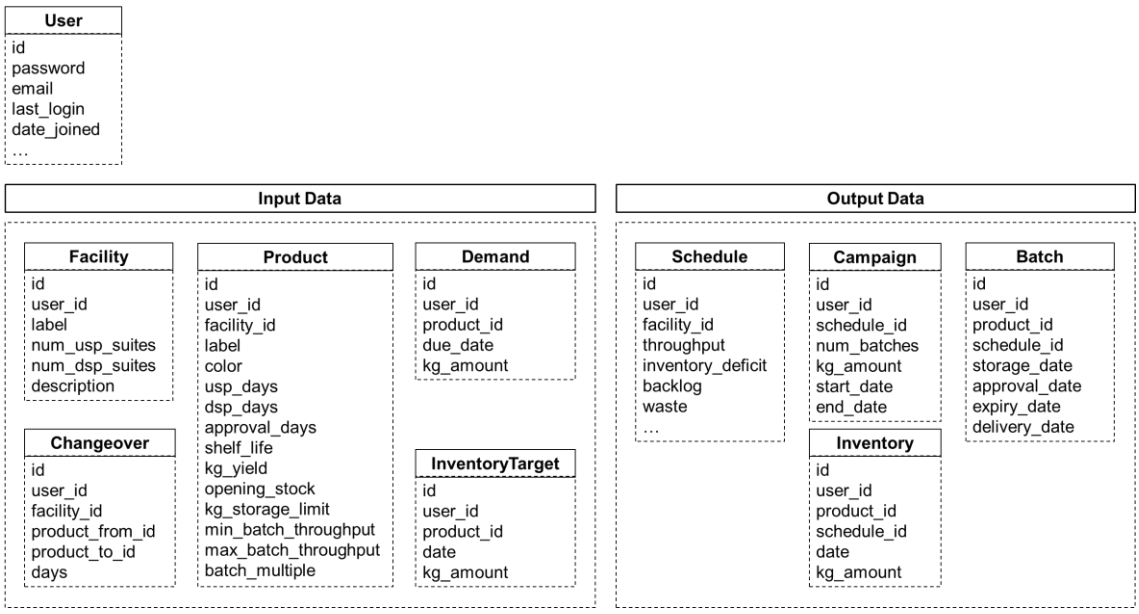


Figure 7.2. Database schema utilised by the GA-based Decision Support Tool.

7.3.2. Input Data Setup

Before the SaaS application can be accessed, a new client would need to create a user account first (Figure 7.3). A user account allows the client to write and save data that is protected with a password. Moreover, the user account type determines how

much the client will be charged for using the service (pricing is discussed in Section 7.4). Once a user account is created, a client can start entering the input data for the scheduling problem.

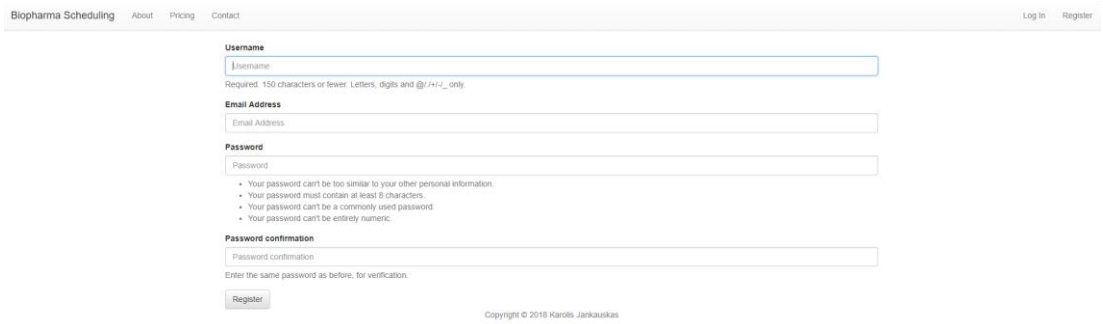
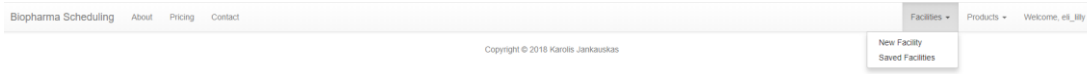


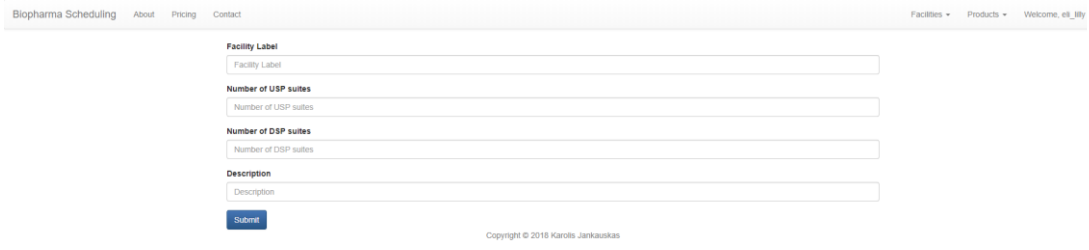
Figure 7.3. Sign up (Register) page view.

a)



b)

c)



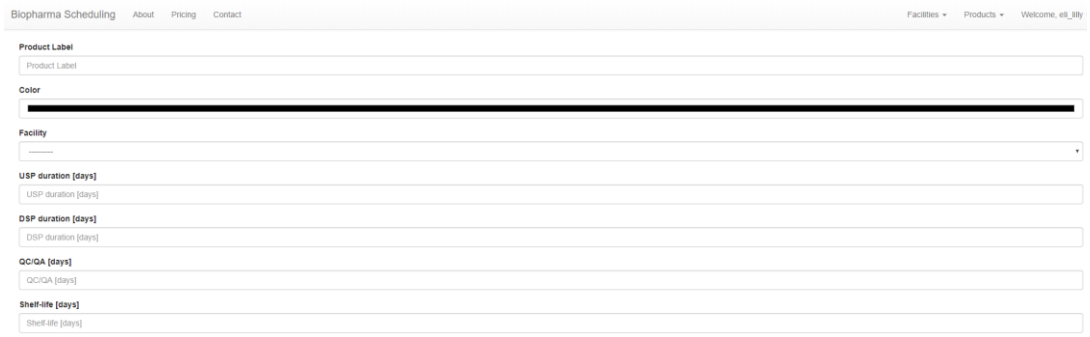
d)

Figure 7.4. Entering facility data into the application. (a) and (b) display the different ways of gaining access to (c) a facility form whereas (d) displays a facility data table.

The first step is to enter data about the biopharmaceutical facility that needs to be optimised. This can be accomplished by filling in the facility form (Figure 7.4.c) that is accessible from the *Facilities* > *New Facility* tab (Figure 7.4.a). The form can also be accessed by selecting the *Facilities* > *Saved Facilities* tab and then clicking the "+ *New Facility"* button (Figure 7.4.b).

a)

b)



c)

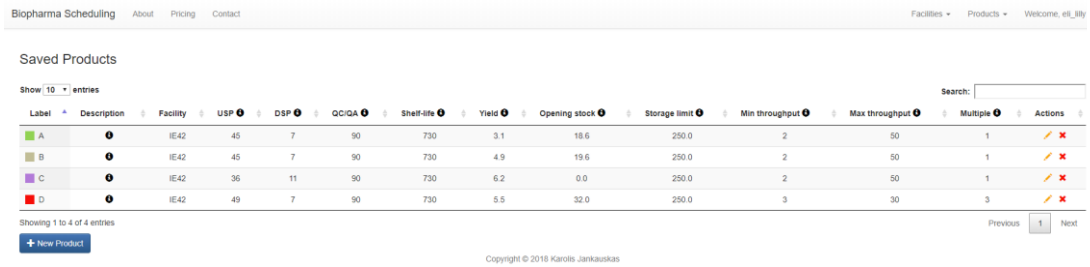


Figure 7.5. Entering product data into the application. (a) displays how to access (b) the form (only a portion of it is shown here) for entering data about an individual product. (c) displays a product data table.

Once the data about the facility is set up (Figure 7.4.d), a client can then begin assigning products to the facility. All forms for entering product-related data, including

process data (7.5.b), product sequence-dependent changeovers, and product demand, can be accessed from the *Products* > *New Product* tab (Figure 7.5.a).

a)

b)

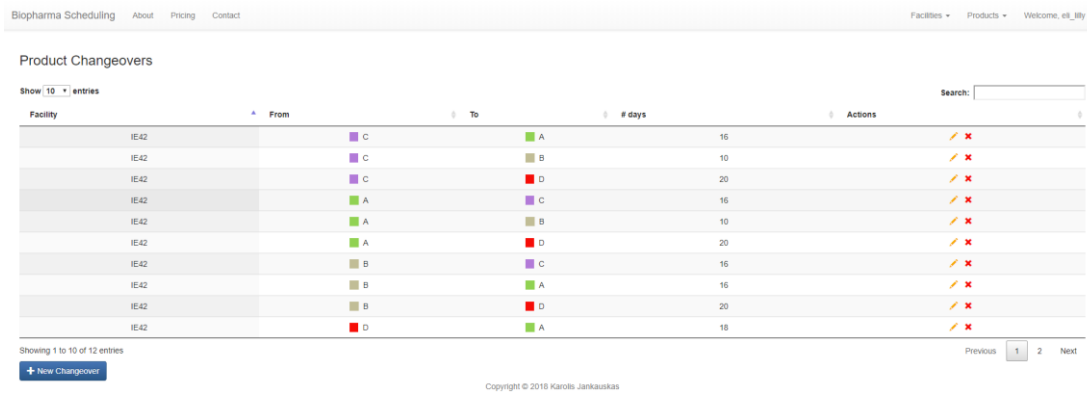


Figure 7.6. Entering product sequence-dependent changeover data into the application. (a) shows the form for entering the data whereas (b) displays the product changeover table with the data filled-in.

When all of the products are defined for a given facility (Figure 7.5.c), a client can enter data about the product sequence-dependent changeovers. This is accomplished by going over to the *Products* > *Product Changeovers* tab which gives access to a product changeovers form (7.6.a). The changeovers data is set up by adding individual rows specifying the product sequence (from-to), the duration (days), and the facility since the changeovers are inherently dependent not only on the product but also on the design and capabilities of a biopharmaceutical facility. The product changeover data table is displayed in Figure 7.6.b. Finally, the product demand and strategic inventory targets can be added to the database of the

application by uploading CSV files containing the data. When the right file is added and uploaded (Figure 7.7.a), a client is presented with a graphical output and a message informing of success (Figure 7.7.b).

a)

b)

Figure 7.7. Uploading product demand data into the application: (a) before and (b) after the upload.

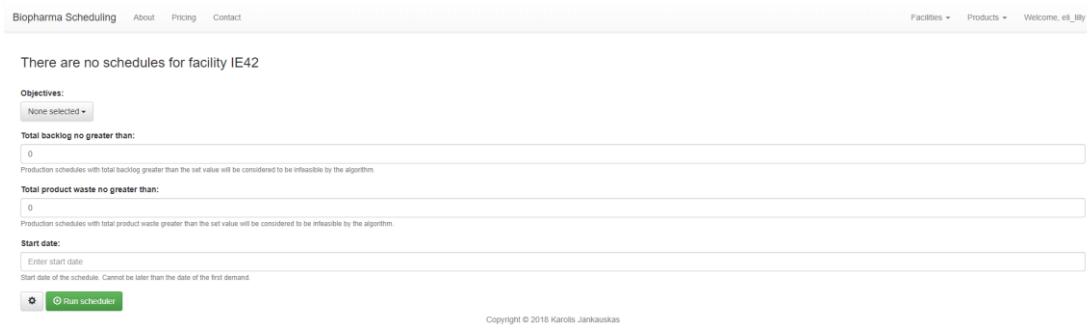
7.3.3. Optimisation Setup

Once all of the required input data is in the database, the scheduling optimisation dashboard for a specific facility can be accessed by clicking the *Plan* icon (Figure 7.8).

Figure 7.8. Accessing scheduling optimisation dashboard.

After having clicked the icon, a client is presented with a minimalistic dashboard that displays only the key problem settings such as objectives, constraints, and the start date of the schedule (Figure 7.9.a). The advanced settings such as the number of runs, generations, and chromosomes, the rates of crossover and mutation operators are also available and can be accessed by clicking the *Advanced Settings* icon (Figure 7.9.b).

a)



b)

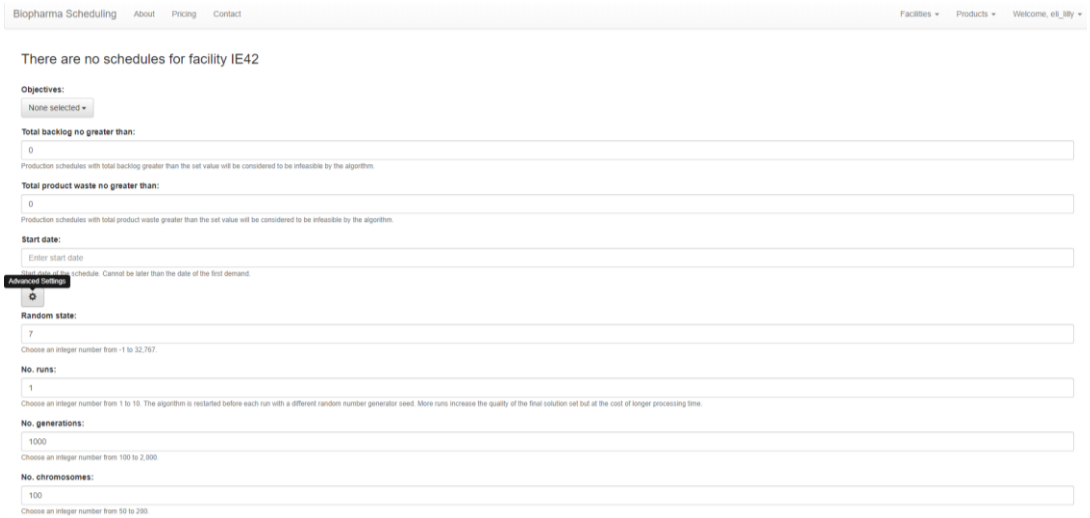


Figure 7.9. Scheduling optimisation setup in the application. (a) lists only the key scheduling optimisation settings whereas (b) displays an expanded list of advanced mostly GA-related parameters.

Once all the parameters are set, the scheduling optimisation can be initiated by clicking the *Run scheduler* button. When this button is pressed, the web framework instructs the scheduling optimisation process to start running in the background. This way the web page remains responsive and a client can continue interacting with it

while the optimisation is running. In certain cases, the optimisation process can take longer than a few seconds to complete. Therefore, a progress bar is used to inform of the application status (Figure 7.10).

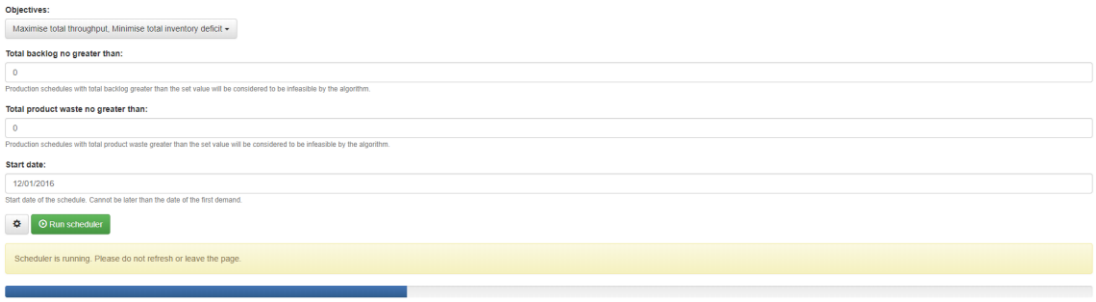


Figure 7.10. Scheduling optimisation in progress.

7.3.4. Visualisation of Results

When the scheduling optimisation is finished, the best non-dominated solutions generated with the GA are transferred to the web framework using Python API for display (Figure 7.11). First, a client will be presented with an interactive table of solutions and, if the scheduling optimisation problem has two objectives, an interactive graph of the best Pareto front (Figure 7.11.a).

a)

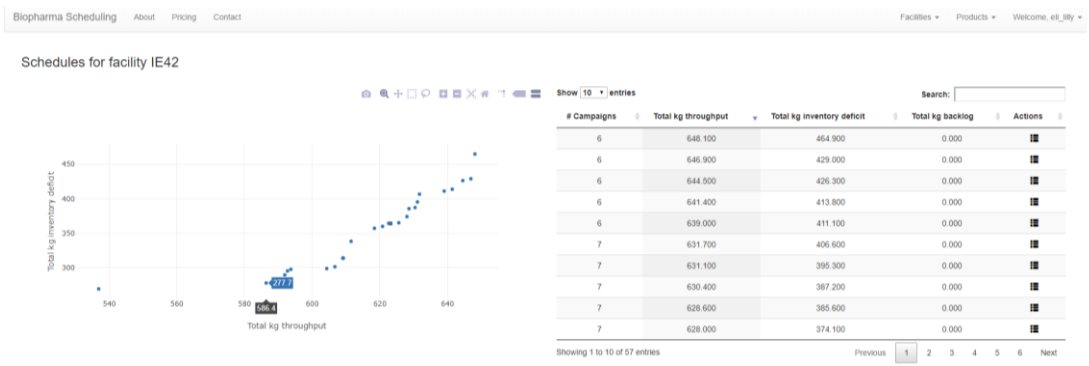


Figure 7.11. Scheduling optimisation results view. In (a), if the optimisation problem has two objectives, an interactive chart of the best Pareto will be displayed. In (b), every row in the table represents a unique schedule that can be inspected by clicking the corresponding View schedule icon in the Actions column.

b)

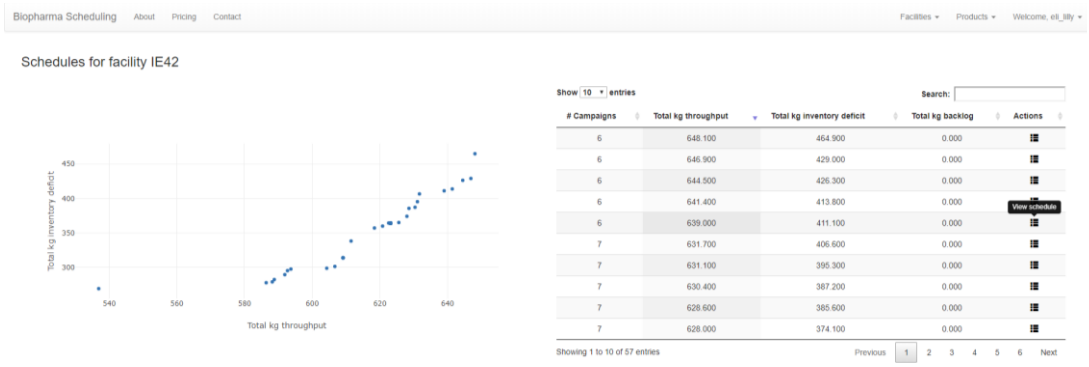


Figure 7.11. (continued) Scheduling optimisation results view. In (a), if the optimisation problem has two objectives, an interactive chart of the best Pareto will be displayed. In (b), every row in the table represents a unique schedule that can be inspected by clicking the corresponding View schedule icon in the Actions column.

Each row in the table corresponds to a unique production schedule which can be inspected by clicking the *View schedule* icon in the *Actions* column (7.11.b). Clicking this icon will bring up a detailed view of the schedule which includes an interactive Gantt chart and an interactive table listing the production throughput, and the start and end dates of each manufacturing campaign (Figure 7.12).

a)

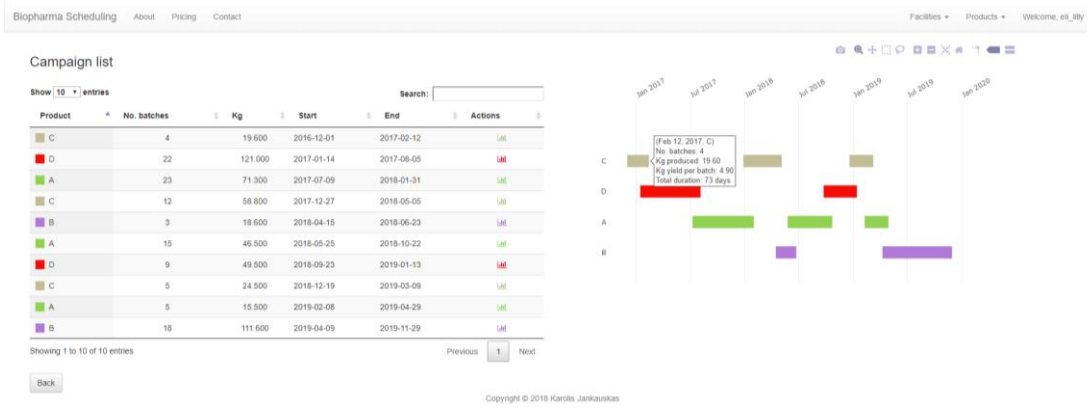


Figure 7.12. Detailed view of a selected production schedule. In (a), manufacturing campaigns can be inspected by hovering over them in the Gantt chart. In (b), individual product profiles can be viewed by selecting the View inventory icon in the Actions column of the production schedule table.

b)

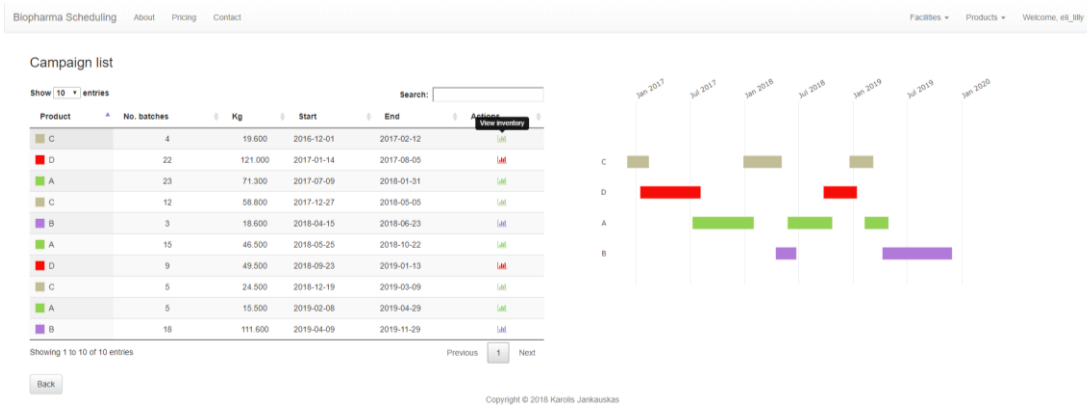


Figure 7.12. (continued) Detailed view of a selected production schedule. In (a), manufacturing campaigns can be inspected by hovering over them in the Gantt chart. In (b), individual product profiles can be viewed by selecting the View inventory icon in the Actions column of the production schedule table.

The inventory profile, strategic targets, and demand of each product (Figure 7.13) can be inspected by clicking the *View inventory* icon in the *Actions* column of the production schedule table (Figure 7.12.a). Clicking this icon would return a web page with interactive charts that display the quantities for each due date.

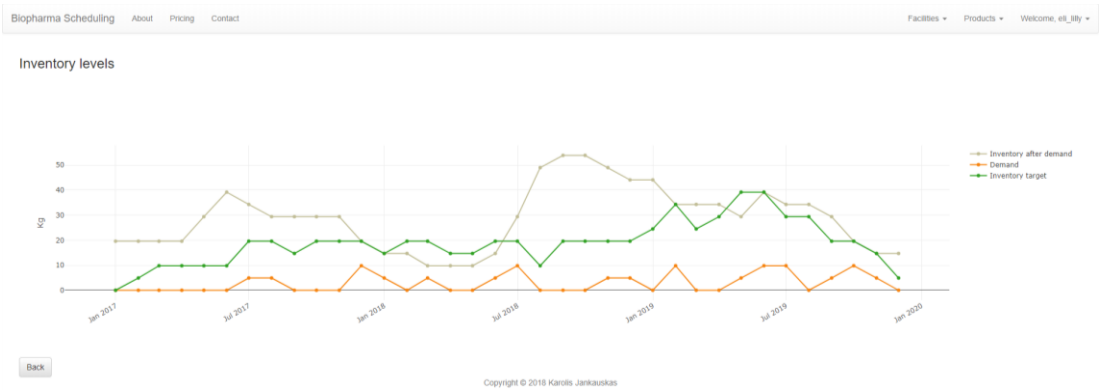


Figure 7.13. Product inventory profile.

7.4. Pricing

Unlike the traditional software development and delivery model, SaaS model has the potential of reaching a broader range of users and building a relationship between a producer and a customer that creates and captures value where the customer participates actively. The relationship with them shapes the service and determines which features are essential and which are not. Hence, it is important to let the customers know that they are buying a highly customizable service that can be improved with their feedback. The experience gained from a relationship with one customer will also help bring new ones onboard. Therefore, the pricing of a SaaS application can be quite complicated compared to on-premise software with perpetual licenses.

Fortunately, with a SaaS model it is possible to charge for usage. The modularity of the GA-based DST enables different pricing strategies. It is not necessary to pack all of the application's value into one single (and often large) number that can put off a lot of smaller scale clients. For starters, clients could select one of three different subscription plans priced proportionally to the number of features made available (see Figure 7.14). A *Free* subscription plan would allow the users to test the basic features of the platform at no cost which would provide some *organic* growth for the business. A free user account can be thought of as the first step towards building a relationship between a client and a SaaS provider. Free users can be later convinced to switch to a paying subscription by showing them the value of the platform. The biggest downside of *Free* subscription plan is its scalability. Too many free users can put a strain on resources, e. g. storage space and processing speed, with a marginal benefit to the service provider. A better alternative to a *Free* subscription plan would be a limited trial period.

Figure 7.14. Monthly subscriptions plans priced proportionally to the number of features provided by the service.

A client would have to pay for *Basic* and *Premium* subscription plans but also would get access to a broad range of useful features such as the ability to create schedules for a larger number of products and longer planning horizons, the ability to perform sensitivity analysis and stochastic optimisation, and have the SaaS provider create a scheduling model that is tailored to their biopharmaceutical facility. The main purpose of a *Basic* subscription plan is to provide a reliable but moderate stream of revenue from the customers who might not necessarily require customisation or complex features but would like to use the tool to estimate the available capacity of the facility or to evaluate the impact of shorter product changeovers or increased product demand. A *Premium* plan with a high-level of customisation would suit customers who operate uniquely designed facilities with distinct manufacturing capabilities and deliver biopharmaceuticals in unpredictable markets. They are likely to benefit from features such as multi-objective scheduling optimisation under uncertainty, e.g. production yields and product demand, the most.

While the pre-defined and readily-available subscription plans are a good starting point, it is better to understand the exact features that the customers value the most. In addition to asking for a customer's feedback directly, this can also be accomplished by collecting and analysing usage data (especially from free or trial users). This would

identify which group certain types of customers fall into and help determine the right subscription plans.

7.5. Summary

This chapter described an implementation plan of the GA-based DST generated during this PhD thesis. An MVP was developed to demonstrate the practical value of the tool and the viability of the plan. It was proposed to deliver it as a SaaS application because of the lower development and deployment costs, and a subscription-based pricing model which would facilitate building stronger, mutually beneficial relationships with the customers.

Once the SaaS application starts generating revenue, it could be expanded further to include not only the models and algorithms for scheduling optimisation but also other features such as process economics models (Farid, 2007), tools for assessing process robustness (Stonier et al., 2012) and algorithms for process design and optimisation (Allmendinger et al., 2012; Simaria et al., 2012). Additional features would increase the value of the application, help retain the existing and attract the new customers.

8. Conclusions and Future Work

8.1. Introduction

Suboptimal scheduling of biopharmaceutical manufacturing campaigns can have significant financial implications such as increased variable costs incurred by holding too much inventory or loss of profit due to insufficient or underutilised capacity to meet the product demand. In order to create a cost-effective production plan, biopharmaceutical companies need to consider a wide range of factors including but not limited to:

- Product-dependent yields and process durations of individual manufacturing stages.
- Manufacturing capabilities of the facility, e.g. the number of *USP* and *DSP* suites.
- Amount of time required for setup and clean-up between the manufacturing campaigns. The duration of a changeover can depend not only on the products but also on the sequence they are manufactured in.
- Product and process-dependent constraints, e.g. finite storage space and product shelf-life, minimum and maximum production throughput limits.
- Multiple concurrent and often conflicting objectives, e.g. maximisation of profit, minimisation of costs, maximisation of production throughput and capacity utilisation, meeting all product demands on time, avoiding product waste, minimisation of the differences between the monthly product inventory levels and the corresponding strategic targets etc.
- Lengthy QC/QA approval times. The manufacturing campaigns have to be scheduled in a such way that the product material is made available several weeks or even months in advance of the delivery due date.

- Uncertain variables inherent in the biopharmaceutical manufacturing environment such as product demand.

This thesis has addressed all of these features of biopharmaceutical capacity planning and scheduling. Moreover, a flexible Decision Support Tool (DST) (including not only the Application Programming Interface but also a more user friendly web-based interface) based on a Genetic Algorithm (GA) was developed to help the biopharmaceutical companies tackle the said challenges and make better scheduling decisions faster. The reasons for taking a GA-based approach were several:

- Most of the optimisation methods for biopharmaceutical capacity planning and scheduling reported utilise discrete-time modelling and Mixed Integer Linear Programming (MILP). There have been multiple MILP-model reported for single- and multi-objective optimisation of biopharmaceutical capacity plans using discrete- and continuous-time representations. In comparison, the number of scheduling optimisation methods based on a GA or other alternatives is much more limited. Therefore, there is significant potential for new research exploring alternative optimisation methods for biopharmaceutical capacity planning and scheduling.
- The tools for mathematical programming typically require high level of expertise (Mustafa et al., 2006). There is also reported lack of transparency associated with certain exact methods which often requires intervention from the specialists (Widmer et al., 2008). In contrast, the core features of a GA are relatively simple to implement (Deb, 2001) and combine with other methods such as Monte Carlo simulation.

Briefly, the GA-based DST brings the following benefits to both academia and industry:

- A variable-length chromosome structure for continuous-time scheduling which makes it possible to create accurate and realistic production schedules. The validity of the variable-length GA was first demonstrated in Chapter 4 on two industrial case studies adapted from the literature and compared with discrete- and continuous-time MILP models. The same variable-length chromosome structure was also applied in Chapters 5 and 6 to solve entirely new deterministic and stochastic multi-objective case studies.
- Flexible multi-objective optimisation. The advantages of a GA-based multi-objective optimisation were demonstrated in Chapter 5. Instead of a single solution, the multi-objective GA presents a biopharmaceutical strategist with a set of non-dominated solutions to choose from.
- Integrated Monte Carlo simulation, i.e. stochastic GA, for generating production schedules under product demand uncertainty. The benefits of stochastic GA were demonstrated in Chapter 6. It identified production schedules which had high probabilities of meeting all product demands on time despite the large variations. In comparison, the production schedules generated with a deterministic approach, i.e. GA without the integrated Monte Carlo simulation, had only < 2% probability of meeting the product demands on time according to the post-optimisation Monte Carlo simulation-based sensitivity analysis.

8.2. Contribution of This Thesis

The following **Sections 8.2.1**-**8.2.5** outline the contributions of this thesis and briefly describe the work undertaken to create a flexible and user-friendly DST for realistic medium-term scheduling of a multi-product biopharmaceutical facility that could tackle multiple conflicting objectives and the uncertainty of biopharmaceutical environment.

8.2.1. Discrete-Time Biopharmaceutical Capacity Planning and Scheduling

At the time of writing Chapter 3 there were not any relevant papers on GA-based methods for biopharmaceutical capacity planning and scheduling. Therefore, the goal was to fill in the knowledge gap by developing GA-based alternatives to the discrete-time MILP-based models reported in the literature. The GA-based scheduling optimisation was evaluated on two cases studies adapted from the literature. In the first case study, a medium-term capacity planning problem of a single-site, multi-suite biopharmaceutical facility was solved. The GA-based approach obtained the global optimum faster than a MILP model. In the second case study, a more complex, long-term capacity planning problem of a multi-site biopharmaceutical manufacture was solved. A rolling time horizon approach was implemented to improve the GA performance. Using this approach, the average optimality gap achieved with the GA was 1.1%.

A Particle Swarm Optimisation (PSO) algorithm was used to automatically tune the GA parameters. The objective of the PSO algorithm was to maximise the mean best GA objective function value achieved after a given number of independent runs using the parameters encoded in a particle's position.

Other key contributions of Chapter 3 include a chromosome encoding strategy, algorithmic adaptations that captured capacity planning objectives for multiple products across multiple suites and facilities, and a rolling time horizon approach to improve the GA performance.

8.2.2. Continuous-Time Biopharmaceutical Capacity Planning and Scheduling

The challenges and limitations encountered in Chapter 3 provided the motivation for the development of unique GA-based scheduling optimisation methods that leveraged the GA's flexibility instead of attempting to solve the scheduling problems by adapting the mathematical models.

Chapter 4 introduced a novel variable-length chromosome structure and an entirely new continuous-time scheduling heuristic for decoding the chromosomes into full-solutions, i.e. production schedules. The heuristic included a wide-range of biopharmaceutical manufacture features such as product-dependent changeovers, multiple intermediate demand due dates, backlogs, limited storage capacity, shelf-life, and waste disposal. The validity of the new approach was demonstrated on two literature-based examples of medium-term biopharmaceutical capacity planning and scheduling problems. In the first example, the variable-length GA was applied to a scheduling problem involving a single, multi-suite (2 *USP* and 2 *DSP* suites) biopharmaceutical facility manufacturing three products. The GA-based approach met all of the product demands on time and achieved a higher objective function value than the discrete- and continuous-time MILP-based solutions reported in the literature. In the second example, the variable-length GA was compared with a discrete-time MILP-based model on a problem involving a longer demand profile and a single, multi-suite (2 *USP* and 3 *DSP* suites) biopharmaceutical facility producing four products. The GA solution met all of the product demands on time and achieved an objective function value that was 33% greater than that of the globally optimal discrete-time MILP solution which met approximately 86% of all product demands on time.

8.2.3. Multi-Objective Biopharmaceutical Capacity Planning and Scheduling

Chapter 5 continued with the development of the variable-length GA for continuous-time biopharmaceutical capacity planning and scheduling by adding a multi-objective component. The variable-length chromosome structure and the new genetic operators introduced in Chapter 4 were integrated with NSGA-II. A completely new, real-life capacity planning and scheduling problem of multi-product biopharmaceutical manufacture was used to demonstrate the advantages of the multi-objective optimisation. The problem featured a single biopharmaceutical facility manufacturing four products to meet a demand profile based on realistic due dates, multiple objectives and constraints, rolling product sequence-dependent changeovers, QC/QA times, storage and shelf-life limits. The objectives of the problem were to maximise the total kilogram throughput and to minimise the sum of differences between the inventory level and the corresponding strategic targets. The production schedules that did not meet product demand on time or resulted in product waste were treated as infeasible. The scheduling problem was first solved using a single objective GA to determine the objective space and set a benchmark for the multi-objective optimisation. The variable-length multi-objective GA achieved on average 99.4% of the total objective space and generated a *Pareto* front that, at the very least, non-dominated the solutions obtained with a single-objective GA.

8.2.4. Multi-Objective Biopharmaceutical Capacity Planning and Scheduling Under Uncertainty

Chapter 6 expanded upon the ideas presented in Chapters 4 and 5 by integrating Monte Carlo simulation with a multi-objective variable-length GA to tackle production scheduling under uncertain product demand. The integrated Monte Carlo simulation

and GA approach, i.e. stochastic GA, was applied to the biopharmaceutical capacity planning and scheduling problem adapted from Chapter 5. The product demand uncertainty was quantified using a triangular distribution defined by the minimum, most likely, and maximum product demand quantities for each due date. The benefits of the stochastic GA were demonstrated by comparing it with a deterministic approach (a GA without Monte Carlo simulation). The stochastic GA permitted the identification of more robust production schedules with much higher probabilities of meeting all product demands on time and lower expected cumulative deviations from the strategic inventory targets. On the other hand, the production schedules generated with a deterministic GA ignored the variability in product demand thus in scenarios where the product demand was higher than expected the solutions were shown to have close to 0% probability of meeting product demands on time.

8.2.5. Commercialisation

Chapter 7 outlined a plan for commercialising the work generated during this PhD. A minimum viable product (MVP) was developed to demonstrate the viability of the plan and the various features of the GA-based DST. A Software-as-a-Service (SaaS)-based delivery model of the product was discussed highlighting many benefits to both the developer(s) as well as the users of the tool or clients. A subscription pricing model was proposed to charge the users or clients according to the usage of tool. Finally, it was suggested that the GA-based DST could be used as a platform to include other decisional frameworks reported in the literature, e.g. process economics models (Farid, 2007), tools for assessing process robustness (Stonier et al., 2012), and algorithms for process design and optimisation (Allmendinger et al., 2012; Simaria et al., 2012).

8.3. Future Work

This PhD thesis demonstrated how GA-based scheduling optimisation can be used to tackle continuous-time, multi-objective, deterministic and stochastic biopharmaceutical capacity planning problems. This section lists a number of ways the work conducted in this PhD thesis could be improved and expanded.

8.3.1. Additional Constraints and Features

The continuous-time scheduling heuristics presented in this thesis assumed that the biopharmaceutical facilities were available for the entire planning horizon. However, in reality, biopharmaceutical companies often have to regularly shut down their facilities for maintenance or inspection. The variable-length chromosome could include several special genes encoding a facility shut-down taking place. This way the GA could generate production schedules with optimised start and end dates of the facility shut-down(s) without compromising the objectives and constraints of the problem. Moreover, an integrated Monte Carlo simulation and GA approach could be applied to evaluate the impact of *unplanned* facility shut-downs which can occur, for example, due to contamination or equipment breakdown.

In Chapter 6, the stochastic GA was applied to create production schedules under demand uncertainty. The novel stochastic scheduling optimisation approach could be taken further to tackle other uncertainties inherent in the biopharmaceutical manufacturing process such as variable fermentation titres and process yields, contamination risks, QC/QA rejection rates, and clinical attrition rates. The integrated Monte Carlo simulation and multi-objective variable-length GA could be used to generate production schedules that have the highest probabilities of meeting the specified objectives and constraints under the aforementioned uncertainties and risks.

This PhD work considered objectives and constraints including total profit, total production throughput, maintaining strategic inventory targets, meeting all product demands on time, and avoiding product waste. It would be interesting to evaluate other objectives such as minimisation of product changeovers. Furthermore, it could also be of interest to test whether the biopharmaceutical facility has sufficient capacity to accommodate the production of an additional product by minimising the total manufacturing time whilst meeting all product demands on time. The GA would need to not only determine the timings and durations of the manufacturing campaigns but also when the facility can remain idle. In this work, each gene corresponded to a real manufacturing campaign. The start and end dates of every manufacturing campaign were inferred from the order of the genes in the variable-length chromosome and the product-dependent process durations and changeovers. A straightforward fix to enable the minimisation of manufacturing time would be to allow the genes to encode *dummy* campaigns that do not have a product label associated with them. The number of batches of a dummy campaign could be used to encode the idle time duration in the facility. The timings of the real and dummy manufacturing campaigns can be implicitly encoded by the order of genes within the chromosome.

Finally, the variable-length chromosome could be extended to address capacity planning and scheduling of multiple biopharmaceutical facilities. For example, the continuous-time scheduling described in Chapter 4 could be adapted for multi-site biopharmaceutical manufacture by changing the *USP* suites to biopharmaceutical facilities. Each gene could encode biopharmaceutical facility and product labels, and the number of batches to be manufactured.

8.3.2. Improved GA-based Optimisation

This PhD thesis focused on a generational single-objective GA and a multi-objective GA based on NSGA-II. It would be beneficial to investigate how the variable-length chromosome structure would perform with other GA types, e.g. compare the generational GA with a steady-state one.

One of the major limitations of NSGA-II is that its performance degrades with an increasing number of objectives and constraints (Ishibuchi et al., 2008). Therefore, for many-objective scheduling problems the variable-length chromosome structure could be integrated with more sophisticated MOEAs such as NSGA-III (Yuan et al., 2014) or Unified (U)-NSGA-III (Seada & Deb, 2015).

It would also be interesting to see whether the work developed in this thesis could be combined and enhanced with Reinforcement Learning, e.g. for tuning and controlling the GA, creating a hyper-heuristic, or for improving the scheduling models. Allmendinger (2012) provided a good review of Reinforcement Learning applications in Evolutionary Optimisation and the development of hyper-heuristics. Zhang (1996) studied Reinforcement Learning applications in short-term job-shop scheduling with the aim to learn a repair-based scheduler capable of repairing a set of temporal and resource constraint violations.

In summary, this PhD thesis presents novel GA-based methods that are relatively easy to implement and provides a strong foundation for future work developing advanced stochastic multi-objective capacity planning and scheduling optimisation methods for the biopharmaceutical industry.

References

Abara, J. (1989). Applying integer linear programming to the fleet assignment problem. Interfaces, 19, 20-28.

Allmendinger, R. (2012). Tuning evolutionary search for closed-loop optimization. The University of Manchester (United Kingdom).

Allmendinger, R., Simaria, A. S., & Farid, S. S. (2012). Efficient discovery of chromatography equipment sizing strategies for antibody purification processes using evolutionary computing. In International Conference on Parallel Problem Solving from Nature (pp. 468-477): Springer.

Allmendinger, R., Simaria, A. S., Turner, R., & Farid, S. S. (2014). Closed-loop optimization of chromatography column sizing strategies in biopharmaceutical manufacture. Journal of chemical technology and biotechnology, 89, 1481-1490.

Almada-Lobo, B., Klabjan, D., Antónia carravilla, M., & Oliveira, J. F. (2007). Single machine multi-product capacitated lot sizing with sequence-dependent setups. International Journal of Production Research, 45, 4873-4894.

Amodeo, L., Prins, C., & Sánchez, D. R. (2009). Comparison of metaheuristic approaches for multi-objective simulation-based optimization in supply chain inventory management. In Workshops on Applications of Evolutionary Computation (pp. 798-807): Springer.

Asenjo, J. A., Montagna, J. M., Vecchietti, A. R., Iribarren, O. A., & Pinto, J. M. (2000). Strategies for the simultaneous optimization of the structure and the process variables of a protein production plant. Computers & Chemical Engineering, 24, 2277-2290.

Back, T. (1993). Optimal mutation rates in genetic search. In Proceedings of the fifth international conference on genetic algorithms (pp. 2-8): Morgan Kaufmann, San Mateo, CA.

Beazley, D. (2010). Understanding the python gil. In PyCON Python Conference. Atlanta, Georgia.

Bitran, G. R., & Yanasse, H. H. (1982). Computational complexity of the capacitated lot size problem. Management science, 28, 1174-1186.

Blau, G. E., Pekny, J. F., Varma, V. A., & Bunch, P. R. (2004). Managing a portfolio of interdependent new product candidates in the pharmaceutical industry. Journal of Product Innovation Management, 21, 227-245.

Branke*, J., & Mattfeld, D. C. (2005). Anticipation and flexibility in dynamic scheduling. International Journal of Production Research, 43, 3103-3129.

Brastow, W., & Rice, C. (2003). Planning pharmaceutical manufacturing strategies in an uncertain world. BioProcess International, 1, 46-55.

Brie, A. H., & Morignot, P. (2005). Genetic Planning Using Variable Length Chromosomes. In ICAPS (pp. 320-329).

Brooke, A., Kendrick, D., Meeraus, A., & Raman, R. (1998). GAMS: the solver manuals. Washington, DC, GAMS Development Corporation.

Brunet, R., Guillén-Gosálbez, G., Pérez-Correa, J. R., Caballero, J. A., & Jiménez, L. (2012). Hybrid simulation-optimization based approach for the optimal design of single-product biotechnological processes. Computers & Chemical Engineering, 37, 125-135.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society, 64, 1695-1724.

Camilleri, M., Neri, F., & Papoutsidakis, M. (2014). An algorithmic approach to parameter selection in machine learning using meta-optimization techniques. WSEAS Transactions on systems, 13, 202-213.

Chen, M.-S., & Liao, F. H. (1998). Neural networks training using genetic algorithms. In Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on (Vol. 3, pp. 2436-2441): IEEE.

Chipperfield, A., & Fleming, P. (1995). The MATLAB genetic algorithm toolbox. In Applied Control Techniques Using MATLAB, IEE Colloquium on (pp. 10/11-10/14): IET.

Choudhary, V. (2007). Comparison of software quality under perpetual licensing and software as a service. Journal of Management Information Systems, 24, 141-165.

Copil, K., Wörbelauer, M., Meyr, H., & Tempelmeier, H. (2017). Simultaneous lotsizing and scheduling problems: a classification and review of models. OR spectrum, 39, 1-64.

Corne, D. W., Knowles, J. D., & Oates, M. J. (2000). The Pareto envelope-based selection algorithm for multiobjective optimization. In International conference on parallel problem solving from nature (pp. 839-848): Springer.

Cui, Y., Geng, Z., Zhu, Q., & Han, Y. (2017). Multi-objective optimization methods and application in energy saving. Energy, 125, 681-704.

Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. IEEE computational science and engineering, 5, 46-55.

Dantzig, G. B. (1951). Maximization of a linear function of variables subject to linear inequalities. New York.

De Jong, K. (1988). Learning with genetic algorithms: An overview. Machine learning, 3, 121-138.

Deb, K. (2001). Multi-objective optimization using evolutionary algorithms (Vol. 16): John Wiley & Sons.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on evolutionary computation, 6, 182-197.

DiMasi, J. A., Grabowski, H. G., & Hansen, R. W. (2016). Innovation in the pharmaceutical industry: new estimates of R&D costs. Journal of health economics, 47, 20-33.

DiMasi, J. A., Hansen, R. W., & Grabowski, H. G. (2003). The price of innovation: new estimates of drug development costs. Journal of health economics, 22, 151-185.

Ding, H., Benyoucef, L., & Xie, X. (2006). A simulation-based multi-objective genetic algorithm approach for networked enterprises optimization. Engineering Applications of Artificial Intelligence, 19, 609-623.

Django. (2018). Django overview | Django. In.

Dubey, A., & Wagle, D. (2007). Delivering software as a service. The McKinsey Quarterly, 6, 2007.

Dutt, A., Jain, H., & Kumar, S. (2017). Providing Software as a Service: a design decision (s) model. Information Systems and e-Business Management, 1-30.

Eberhart, R. C., & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In Evolutionary Computation, 2000. Proceedings of the 2000 Congress on (Vol. 1, pp. 84-88): IEEE.

Eiben, Á. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. IEEE Transactions on evolutionary computation, 3, 124-141.

Eskandari, H., Rabelo, L., & Mollaghasemi, M. (2005). Multiobjective simulation optimization using an enhanced genetic algorithm. In Proceedings of the 37th conference on Winter simulation (pp. 833-841): Winter Simulation Conference.

Farid. (2007). Process economics of industrial monoclonal antibody manufacture. Journal of Chromatography B, 848, 8-18.

Farid, S., Washbrook, J., & Titchener-Hooker, N. J. (2007). Modelling biopharmaceutical manufacture: Design and implementation of SimBiopharma. Computers & Chemical Engineering, 31, 1141-1158.

Farid, S. S., Novais, J. L., Karri, S., Washbrook, J., & Titchener-Hooker, N. J. (2000). A tool for modeling strategic decisions in cell culture manufacturing. Biotechnology Progress, 16, 829-836.

Farid, S. S., Washbrook, J., & Titchener-Hooker, N. (2001). Decision-support tool for risk analysis in biopharmaceutical manufacture. IFAC Proceedings Volumes, 34, 161-165.

Farid, S. S., Washbrook, J., & Titchener-Hooker, N. J. (2005). Decision-support tool for assessing biomanufacturing strategies under uncertainty: Stainless steel versus disposable equipment for clinical trial material preparation. Biotechnology Progress, 21, 486-497.

Fike, R. (2009). Nutrient Supplementation Strategies for Biopharmaceutical Production, Part 2. BioProcess Int, 7.

Floudas, C. A., & Lin, X. (2004). Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review. Computers & Chemical Engineering, 28, 2109-2129.

Fogel, D. B., & Atmar, J. W. (1990). Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. Biological Cybernetics, 63, 111-114.

Fonseca, C. M., & Fleming, P. J. (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 28, 26-37.

Fonseca, C. M., Paquete, L., & López-Ibánez, M. (2006). An improved dimension-sweep algorithm for the hypervolume indicator. In Evolutionary Computation, 2006. CEC 2006. IEEE Congress on (pp. 1157-1163): IEEE.

Fortin, F.-A., Rainville, F.-M. D., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research, 13, 2171-2175.

Friedman, L. M., Furberg, C., DeMets, D. L., Reboussin, D., & Granger, C. B. (2015). Fundamentals of clinical trials: Springer.

Gatica, G., Papageorgiou, L. G., & Shah, N. (2003). Capacity Planning Under Uncertainty for the Pharmaceutical Industry. Chemical Engineering Research and Design, 81, 665-678.

George, E. D., & Farid, S. S. (2008). Strategic Biopharmaceutical Portfolio Development: An Analysis of Constraint-Induced Implications. Biotechnology Progress, 24, 698-713.

Gicquel, C., Hege, L., Minoux, M., & Van Canneyt, W. (2012). A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints. Computers & Operations Research, 39, 629-636.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers & Operations Research, 13, 533-549.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. Foundations of genetic algorithms, 1, 69-93.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. IEEE Transactions on systems, man, and cybernetics, 16, 122-128.

Gutjahr, W. J., & Pichler, A. (2016). Stochastic multi-objective optimization: a survey on non-scalarizing methods. Annals of Operations Research, 236, 475-499.

Hamdy, M., Nguyen, A.-T., & Hensen, J. L. (2016). A performance comparison of multi-objective optimization algorithms for solving nearly-zero-energy-building design problems. Energy and Buildings, 121, 57-71.

Harjunkoski, I., Maravelias, C. T., Bongers, P., Castro, P. M., Engell, S., Grossmann, I. E., Hooker, J., Méndez, C., Sand, G., & Wassick, J. (2014). Scope for

industrial applications of production scheduling models and solution methods. Computers & Chemical Engineering, 62, 161-193.

Hartl, D. L. (1988). A primer of population genetics: Sinauer Associates, Inc.

Hassan, R., Cohanim, B., de Weck, O., & Venter, G. (2004). A copmarison of particle swarm optimization and the genetic algorithm. American Institute of Aeronautics and Astronautics.

Haupt, R. (1989). A survey of priority rule-based scheduling. Operations-Research-Spektrum, 11, 3-16.

Hertz, A., & Widmer, M. (2003). Guidelines for the use of meta-heuristics in combinatorial optimization. European Journal of Operational Research, 151, 247-252.

Ho, J. C., Chang, Y.-L., & Solis, A. O. (2006). Two modifications of the least cost per period heuristic for dynamic lot-sizing. Journal of the Operational Research Society, 57, 1005-1013.

Hodges Jr, J. L., & Lehmann, E. L. (1963). Estimates of location based on rank tests. The Annals of Mathematical Statistics, 598-611.

Holland, J., & Goldberg, D. (1989). Genetic algorithms in search, optimization and machine learning. Massachusetts: Addison-Wesley.

Holland, J. H. (1975). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence: U Michigan Press.

IDC. (2018). Worldwide Public Cloud Services Spending Forecast to Reach $122.5 Billion in 2017, According to IDC. In.

Ishibuchi, H., Tsukamoto, N., & Nojima, Y. (2008). Evolutionary many-objective optimization. In Genetic and Evolving Systems, 2008. GEFS 2008. 3rd International Workshop on (pp. 47-52): IEEE.

James, K., & Russell, E. (1995). Particle swarm optimization. In Proceedings of 1995 IEEE International Conference on Neural Networks (pp. 1942-1948).

James, R. J., & Almada-Lobo, B. (2011). Single and parallel machine capacitated lotsizing and scheduling: New iterative MIP-based neighborhood search heuristics. Computers & Operations Research, 38, 1816-1825.

Jankauskas, K., Long, A., Osborne, M., McCartney, G., Papageorgiou, L., & Farid, S. (2017). Multi-objective medium-term capacity planning for stainless steel and single-use multi-product biopharmaceutical facilities under uncertainty using a genetic algorithm. In ABSTRACTS OF PAPERS OF THE AMERICAN CHEMICAL SOCIETY (Vol. 253): AMER CHEMICAL SOC 1155 16TH ST, NW, WASHINGTON, DC 20036 USA.

Jankauskas, K., Papageorgiou, L. G., & Farid, S. S. (2017). Continuous-Time Heuristic Model for Medium-Term Capacity Planning of a Multi-Suite, Multi-Product Biopharmaceutical Facility. In Computer Aided Chemical Engineering (Vol. 40, pp. 1303-1308): Elsevier.

Jans, R., & Degraeve, Z. (2008). Modeling industrial lot sizing problems: a review. International Journal of Production Research, 46, 1619-1643.

Jiang, Z., Droms, K., Geng, Z., Casnocha, S., Xiao, Z., Gorfien, S., & Jacobia, S. J. (2012). Fed-Batch Cell Culture Process Optimization. BioProcess International.

Jin, Y., & Branke, J. (2005). Evolutionary optimization in uncertain environments-a survey. IEEE Transactions on evolutionary computation, 9, 303-317.

Kabra, S., Shaik, M. A., & Rathore, A. S. (2013). Multi-period scheduling of a multi-stage multi-product bio-pharmaceutical process. Computers & Chemical Engineering, 57, 95-103.

Kaitin, K., & DiMasi, J. (2010). Pharmaceutical innovation in the 21st century: new drug approvals in the first decade, 2000–2009. Clinical Pharmacology & Therapeutics, 89, 183-188.

Kallrath, J. (2002). Planning and scheduling in the process industry. OR spectrum, 24, 219-250.

Kalyanmoy, D. (2011). Multi-objective optimization using evolutionary algorithms: An introduction. KanGAL Report.

Kamarck, M. E. (2006). Building biomanufacturing capacity—the chapter and verse. Nature biotechnology, 24, 503-505.

Karimi, B., Fatemi Ghomi, S. M. T., & Wilson, J. M. (2003). The capacitated lot sizing problem: a review of models and algorithms. Omega, 31, 365-378.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In Proceedings of the sixteenth annual ACM symposium on Theory of computing (pp. 302-311): ACM.

Kimms, A. (1999). A genetic algorithm for multi-level, multi-machine lot sizing and scheduling. Computers & Operations Research, 26, 829-848.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. science, 220, 671-680.

Knowles, J. D., Corne, D. W., & Fleischer, M. (2003). Bounded archiving using the Lebesgue measure. In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on (Vol. 4, pp. 2490-2497): IEEE.

Köksalan, M. M., Wallenius, J., & Zionts, S. (2011). Multiple criteria decision making: from early history to the 21st century: World Scientific.

Kondili, E., Pantelides, C., & Sargent, R. (1993). A general algorithm for short-term scheduling of batch operations—I. MILP formulation. Computers & Chemical Engineering, 17, 211-227.

Laínez, J. M., Schaefer, E., & Reklaitis, G. V. (2012). Challenges and opportunities in enterprise-wide optimization in the pharmaceutical industry. Computers & Chemical Engineering, 47, 19-28.

Lakhdar, K., Farid, S., Savery, J., Titchener-Hooker, N., & Papageorgiou, L. (2006). Medium term planning of biopharmaceutical manufacture under uncertainty. Computer Aided Chemical Engineering, 21, 2069-2074.

Lakhdar, K., & Papageorgiou, L. G. (2006). An iterative mixed integer optimisation approach for medium term planning of biopharmaceutical manufacture under uncertainty. Chemical Engineering Research and Design, 86, 259-267.

Lakhdar, K., & Papageorgiou, L. G. (2008). An iterative mixed integer optimisation approach for medium term planning of biopharmaceutical manufacture under uncertainty. Chemical Engineering Research and Design, 86, 259-267.

Lakhdar, K., Savery, J., Papageorgiou, L., & Farid, S. (2007). Multiobjective Long-Term Planning of Biopharmaceutical Manufacturing Facilities. Biotechnology Progress, 23, 1383-1393.

Lakhdar, K., Zhou, Y., Savery, J., Titchener-Hooker, N. J., & Papageorgiou, L. G. (2005). Medium term planning of biopharmaceutical manufacture using mathematical programming. Biotechnology Progress, 21, 1478-1489.

Langer, E. (2009). Trends in capacity utilization for therapeutic monoclonal antibody production. In MAbs (Vol. 1, pp. 151-156): Taylor & Francis.

Langer, E., & Rader, R. A. (2017). Top Trends in Biopharmaceutical Manufacturing, 2017. Pharmaceutical Technology, 41.

Levis, A. A., & Papageorgiou, L. G. (2004). A hierarchical solution approach for multi-site capacity planning under uncertainty in the pharmaceutical industry. Computers & Chemical Engineering, 28, 707-725.

Lim, A. C., Washbrook, J., Titchener-Hooker, N. J., & Farid, S. S. (2006). A computer-aided approach to compare the production economics of fed-batch and perfusion culture under uncertainty. Biotechnology and bioengineering, 93, 687-697.

Lin, W.-Y., Lee, W.-Y., & Hong, T.-P. (2003). Adapting crossover and mutation rates in genetic algorithms. J. Inf. Sci. Eng., 19, 889-903.

Liu, S., Simaria, A. S., Farid, S. S., & Papageorgiou, L. G. (2013). Designing cost-effective biopharmaceutical facilities using mixed-integer optimization. Biotechnology Progress, 29, 1472-1483.

Lorigeon, T., Billaut, J., & Bouquard, J. (2002). A dynamic programming algorithm for scheduling jobs in a two-machine open shop with an availability constraint. Journal of the Operational Research Society, 53, 1239-1246.

Luke, S. (2009). Essentials of metaheuristics (Vol. 113): Lulu Raleigh.

Luke, S. (2013). Essentials of metaheuristics: Lulu Com.

Majozi, T., Seid, E. R., & Lee, J.-Y. (2015). Synthesis, Design, and Resource Optimization in Batch Chemical Plants: CRC Press.

Malik, A., Pinkus, G., & Sheffer, S. (2002). Biopharma's capacity crunch. McKinsey Quarterly. In.

Melanie, M. (1996). An Introduction to Genetic Algorithms.

Méndez, C. A., Cerdá, J., Grossmann, I. E., Harjunkoski, I., & Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. Computers & Chemical Engineering, 30, 913-946.

Metropolis, N., & Ulam, S. (1949). The monte carlo method. Journal of the American statistical association, 44, 335-341.

Miller, D. L., Schertz, D., Stevens, C., & Pekny, J. F. (2010). Mathematical programming for the design and analysis of a biologics facility. BioPharm International, 23.

Montagna, J. M., Vecchietti, A. R., Iribarren, O. A., Pinto, J. M., & Asenjo, J. A. (2000). Optimal design of protein production plants with time and size factor process models. Biotechnology Progress, 16, 228-237.

Mustafa, M., Washbrook, J., Titchener-Hooker, N., & Farid, S. (2006). Retrofit decisions within the biopharmaceutical industry: an EBA case study. Food and bioproducts processing, 84, 84-89.

Nie, W. (2015). Cost evaluation and portfolio management optimization for biopharmaceutical product development. UCL (University College London).

Nie, W., Zhou, Y., Simaria, A. S., & Farid, S. S. (2012). Biopharmaceutical portfolio management optimization under uncertainty. In Symposium on Computer Aided Process Engineering (Vol. 17, pp. 20).

Nvidia, C. (2011). Nvidia cuda c programming guide. Nvidia Corporation, 120, 8.

Otto, R., Santagostino, A., & Schrader, U. (2014). Rapid growth in biopharma: Challenges and opportunities. In.

Oyebolu, F. B., van Lidth de Jeude, J., Siganporia, C., Farid, S. S., Allmendinger, R., & Branke, J. (2017). A new lot sizing and scheduling heuristic for multi-site biopharmaceutical production. Journal of heuristics, 23, 231-256.

Pandey, S., Wu, L., Guru, S. M., & Buyya, R. (2010). A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In Advanced information networking and applications (AINA), 2010 24th IEEE international conference on (pp. 400-407): IEEE.

Pantelides, C. C. (1994). Unified frameworks for optimal process planning and scheduling. In Proceedings on the second conference on foundations of computer aided operations (pp. 253-274): Cache Publications New York.

Panwalkar, S. S., & Iskander, W. (1977). A survey of scheduling rules. Operations Research, 25, 45-61.

Papageorgiou, L. G., Rotstein, G. E., & Shah, N. (2000). Strategic Supply Chain Optimization for the Pharmaceutical Industries. Industrial & Engineering Chemistry Research, 40, 275-286.

Papageorgiou, L. G., Rotstein, G. E., & Shah, N. (2001). Strategic supply chain optimization for the pharmaceutical industries. Industrial & Engineering Chemistry Research, 40, 275-286.

Paparrizos, K., Samaras, N., & Stephanides, G. (2003). A new efficient primal dual simplex algorithm. Computers & Operations Research, 30, 1383-1399.

Paul, S. M., Mytelka, D. S., Dunwiddie, C. T., Persinger, C. C., Munos, B. H., Lindborg, S. R., & Schacht, A. L. (2010). How to improve R&D productivity: the pharmaceutical industry's grand challenge. Nature Reviews Drug Discovery, 9, 203-214.

Piana, S., & Engell, S. (2010). Hybrid evolutionary optimization of the operation of pipeless plants. Journal of heuristics, 16, 311-336.

Pinto, J. M., & Grossmann, I. E. (1998). Assignment and sequencing models for thescheduling of process systems. Annals of Operations Research, 81, 433-466.

Pochet, Y., & Wolsey, L. A. (2006). Production planning by mixed integer programming: Springer Science & Business Media.

Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. Swarm intelligence, 1, 33-57.

Raisanen, L., & Whitaker, R. M. (2005). Comparison and evaluation of multiple objective genetic algorithms for the antenna placement problem. Mobile Networks and Applications, 10, 79-88.

Rajapakse, A., Titchener-Hooker, N. J., & Farid, S. S. (2005). Modelling of the biopharmaceutical drug development pathway and portfolio management. Computers & Chemical Engineering, 29, 1357-1368.

Rajapakse, A., Titchener-Hooker, N. J., & Farid, S. S. (2006). Integrated approach to improving the value potential of biopharmaceutical R&D portfolios while mitigating risk. Journal of chemical technology and biotechnology, 81, 1705-1714.

Ransohoff, T. C. (2004). Considerations impacting the make vs. buy decision. Amer Pharm Outsourcing, 5, 52-63.

Ray, T., Tai, K., & Seow, C. (2001). An evolutionary algorithm for multiobjective optimization. Eng. Optim, 33, 399-424.

Reeves, C. (2003). Genetic algorithms. In Handbook of metaheuristics (pp. 55-82): Springer.

rey Horn, J., Nafpliotis, N., & Goldberg, D. E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. In Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence (Vol. 1, pp. 82-87): Citeseer.

Rotstein, G. E., Papageorgiou, L. G., Shah, N., Murphy, D. C., & Mustafa, R. (1999). A product portfolio approach in the pharmaceutical industry. Computers & Chemical Engineering, 23, Supplement, S883-S886.

Sabatier, V., Mangematin, V., & Rousselle, T. (2010). From recipe to dinner: business model portfolios in the European biopharmaceutical industry. Long Range Planning, 43, 431-447.

Samsatli, N., & Shah, N. (1996a). An optimization based design procedure for biochemical processes: Part I: Preliminary design and operation. Food and bioproducts processing, 74, 221-231.

Samsatli, N., & Shah, N. (1996b). An optimization based design procedure for biochemical processes: Part II: Detailed scheduling. Food and bioproducts processing, 74, 232-242.

Sand, G., Till, J., Tometzki, T., Urselmann, M., Engell, S., & Emmerich, M. (2008). Engineered versus standard evolutionary algorithms: A case study in batch scheduling with recourse. Computers & Chemical Engineering, 32, 2706-2722.

Saraph, P. V. (2001). Simulating biotech manufacturing operations: issues and complexities. In Proceedings of the 33nd conference on Winter simulation (pp. 530-524): IEEE Computer Society.

Savage, S. (2002). The flaw of averages. Harvard Business Review, 80, 20-21.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In Proceedings of the First International Conference on Genetic

Algorithms and Their Applications, 1985: Lawrence Erlbaum Associates. Inc., Publishers.

Schaffer, J. D., & Morishima, A. (1987). An adaptive crossover distribution mechanism for genetic algorithms. In Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms (pp. 36-40): Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Schmidt, C. W., & Grossmann, I. E. (1996). Optimization models for the scheduling of testing tasks in new product development. Industrial & Engineering Chemistry Research, 35, 3498-3510.

Schrijver, A. (1998). Theory of linear and integer programming: John Wiley & Sons.

Seada, H., & Deb, K. (2015). U-NSGA-III: a unified evolutionary optimization procedure for single, multiple, and many objectives: proof-of-principle results. In International Conference on Evolutionary Multi-Criterion Optimization (pp. 34-49): Springer.

Senaratna, N. I. (2005). Genetic algorithms: The crossover-mutation debate. Bachelor of Computer Science (Special) of the University of Colombo.

Shah, N. (1998). Planning and scheduling-single-and multisite planning and scheduling: Current status and future challenges. In AIChE Symposium Series (Vol. 94, pp. 75-90): New York, NY: American Institute of Chemical Engineers, 1971-c2002.

Shaik, M. A., Janak, S. L., & Floudas, C. A. (2006). Continuous-time models for short-term scheduling of multipurpose batch plants: A comparative study. Industrial & Engineering Chemistry Research, 45, 6190-6209.

Shanley, A. (2014). Tufts' New Figures on Drug-Development Costs Spark Debate. In (Vol. 2016). BioPharm International.com: BioPharm International.

Sierksma, G. (2001). Linear and integer programming: theory and practice: CRC Press.

Siganporia, C. (2016). Strategic Biopharmaceutical Production Planning for Batch and Perfusion Processes. UCL (University College London).

Siganporia, C. C., Ghosh, S., Daszkowski, T., Papageorgiou, L. G., & Farid, S. S. (2014). Capacity planning for batch and perfusion bioprocesses across multiple biopharmaceutical facilities. Biotechnology Progress.

Silver, E. A. (1973). A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. Prod. Inventory Manage., 2, 64-74.

Simaria, A. S., Turner, R., & Farid, S. S. (2012). A multi-level meta-heuristic algorithm for the optimisation of antibody purification processes. Biochemical Engineering Journal, 69, 144-154.

Spears, W. M., & Anand, V. (1991). A study of crossover operators in genetic programming. In International Symposium on Methodologies for Intelligent Systems (pp. 409-418): Springer.

Stack-Overflow. (2018). Stack Overflow Developer Survey 2018. In.

Stanley, K. O., & Miikkulainen, R. (2002). Efficient evolution of neural network topologies. In Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on (Vol. 2, pp. 1757-1762): IEEE.

Stone, R. E., & Tovey, C. A. (1991). The simplex and projective scaling algorithms as iteratively reweighted least squares methods. SIAM review, 33, 220-237.

Stonier, A., Simaria, A. S., Smith, M., & Farid, S. S. (2012). Decisional tool to assess current and future process robustness in an antibody purification facility. Biotechnology Progress, 28, 1019-1028.

Svinivas, N. (1995). Multiobjective optimization using nondominated sorting in genetic algorithms. IEEE Trans. Evol. Comput., 2, 221-248.

Syberfeldt, A., Ng, A., John, R. I., & Moore, P. (2009). Multi-objective evolutionary simulation-optimisation of a real-world manufacturing problem. Robotics and Computer-Integrated Manufacturing, 25, 926-931.

Taherdangkoo, M., Paziresh, M., Yazdi, M., & Bagheri, M. (2013). An efficient algorithm for function optimization: modified stem cells algorithm. In Open Engineering (Vol. 3, pp. 36).

Tait, K. (1998). Pharmaceutical industry. In (pp. 79.74-75).

Ting, C.-K., Lee, C.-N., Chang, H.-C., & Wu, J.-S. (2009). Wireless heterogeneous transmitter placement using multiobjective variable-length genetic algorithm. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39, 945-958.

Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. Information processing letters, 85, 317-325.

Vanek, J., Michálek, J., & Psutka, J. (2017). A Comparison of Support Vector Machines Training GPU-Accelerated Open Source Implementations. arXiv preprint arXiv:1707.06470.

Varma, V. A., Pekny, J. F., Blau, G. E., & Reklaitis, G. V. (2008). A framework for addressing stochastic and combinatorial aspects of scheduling and resource allocation in pharmaceutical R&D pipelines. Computers & Chemical Engineering, 32, 1000-1015.

Vasquez-Alvarez, E., & Pinto, J. (2004). Efficient MILP formulations for the optimal synthesis of chromatographic protein purification processes. Journal of Biotechnology, 110, 295-311.

Vieira, M., Pinto-Varela, T., Moniz, S., Barbosa-Póvoa, A. P., & Papageorgiou, L. G. (2016). Optimal planning and campaign scheduling of biopharmaceutical processes using a continuous-time formulation. Computers & Chemical Engineering, 91, 422-444.

Wagner, H. M., & Whitin, T. M. (1958). Dynamic version of the economic lot size model. Management science, 5, 89-96.

Walsh, G. (2010). Biopharmaceutical benchmarks 2010. Nature biotechnology, 28, 917.

Widmer, M., Hertz, A., & Costa, D. (2008). Metaheuristics and Scheduling. Production Scheduling, 33-68.

Yang, Y., Farid, S. S., & Thornhill, N. F. (2014). Data mining for rapid prediction of facility fit and debottlenecking of biomanufacturing facilities. Journal of Biotechnology, 179, 17-25.

Yuan, Y., Xu, H., & Wang, B. (2014). An improved NSGA-III procedure for evolutionary many-objective optimization. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (pp. 661-668): ACM.

Zhang, W. (1996). Reinforcement learning for job-shop scheduling.

Zitzler, E., & Künzli, S. (2004). Indicator-based selection in multiobjective search. In International Conference on Parallel Problem Solving from Nature (pp. 832-842): Springer.

Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. TIK-report, 103.

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Transactions on evolutionary computation, 3, 257-271.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & Da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on evolutionary computation, 7, 117-132.

Appendix

Appendix A

A.1. Publications

Jankauskas, K., Papageorgiou, L.G. and Farid, S.S., 2017. Continuous-Time Heuristic Model for Medium-Term Capacity Planning of a Multi-Suite, Multi-Product Biopharmaceutical Facility. In Computer Aided Chemical Engineering (Vol. 40, pp. 1303-1308). Elsevier.

Jankauskas, K., Papageorgiou, L. G., & Farid, S. S. (2017). Fast Genetic Algorithm Approaches to Solving Discrete-Time Mixed Integer Linear Programming Problems of Capacity Planning and Scheduling of Biopharmaceutical Manufacture. Computer Aided Chemical Engineering (submitted)

Jankauskas, (Eli Lilly authors TBD), Papageorgiou, LG., Farid, SS. 2017. Multi-objective Capacity Planning For Multi-product Biopharmaceutical Facilities Under Uncertainty Using a Flexible Genetic Algorithm Approach. (Journal TBD) (in preparation)

A.1. Conferences

Jankauskas, K., Papageorgiou, LG., Farid, SS. 2018. Continuous-Time Heuristic Model for Medium-Term Capacity Planning Of A Multi-Suite, Multi-Product Biopharmaceutical Facility (Keynote), 27th European Symposium on Computer Aided Process Engineering (ESCAPE), Barcelona, Spain, October 2-6.

Jankauskas, K., McCartney, GR., Osborne, MD., Papageorgiou, LG., Farid, SS. 2017. Multi-Objective Capacity Planning for Multi-Product Biopharmaceutical Facilities Under Uncertainty, 253rd ACS National Meeting, San Francisco, USA, April 2-6.

Jankauskas, K., Papageorgiou, LG., Farid, SS. 2016. Production Scheduling of A Multi-Product Biopharmaceutical Facility Using a Genetic Algorithm, 28th European Conference on Operational Research (EURO), Poznan, Poland, July 4-8.

Appendix B

This appendix provides the technical details of the core flexible GA-based DST components developed during this PhD thesis.

B.1. Gene

A gene class/structure was created for encapsulating problem-specific information, e.g. encoded variables and methods for mutating them. This way, the variable-length chromosome (see the next Appendix) can be used to solve different biopharmaceutical scheduling problems just by plugging in a corresponding gene encoding the minimum required variables. For example, the *SingleSiteSimpleGene* displayed in *Figure B.1.b* does not have fields and methods associated with the *USP* suites, e.g. *usp_suite_num* and *mutate_usp_suite_num()*, as this was not mandated by the scheduling problems described in Chapters 5 and 6.

a) **b)**

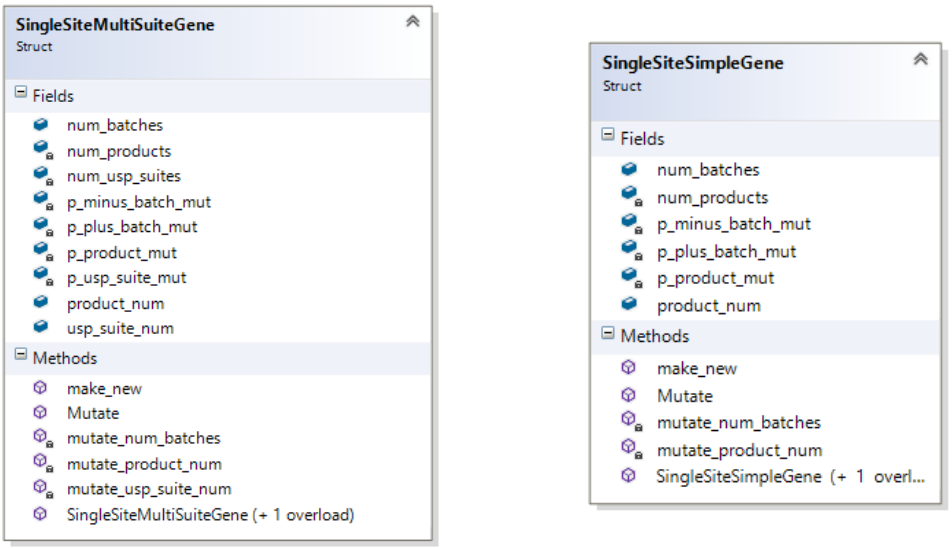


Figure B.1. Structure of a single variable-length chromosome gene:
(a) Gene structure utilised in Chapter 4
(b) Gene structure utilised in Chapters 5 and 6

Algorithm B.1. C++ implementation of a gene used in Chapter 4.

```
#include <utility>
#include "utils.h"

struct SingleSiteMultiSuiteGene
{
    SingleSiteMultiSuiteGene() {}

    SingleSiteMultiSuiteGene(
        int num_products,
        int num_usp_suites,
        double p_product_mut,
        double p_usp_suite_mut,
        double p_plus_batch_mut,
        double p_minus_batch_mut
    )
    {
        this->num_products = num_products,
        this->num_usp_suites = num_usp_suites,
        this->p_product_mut = p_product_mut,
        this->p_usp_suite_mut = p_usp_suite_mut,
        this->p_plus_batch_mut = p_plus_batch_mut,
        this->p_minus_batch_mut = p_minus_batch_mut,
        this->num_batches = 1;
        this->product_num = utils::random_int(1, num_products);
        this->usp_suite_num = utils::random_int(1, num_usp_suites);
    }

    SingleSiteMultiSuiteGene make_new()
    {
        return std::move(
            SingleSiteMultiSuiteGene(
                num_products,
                num_usp_suites,
                p_product_mut,
                p_usp_suite_mut,
                p_plus_batch_mut,
                p_minus_batch_mut
            )
        );
    }

    inline void Mutate()
    {
        mutate_product_num();
        mutate_usp_suite_num();
        mutate_num_batches();
    }

    int product_num;
    int usp_suite_num;
    int num_batches;

private:
    inline void mutate_product_num()
    {
        if (utils::random() >= p_product_mut) {
            return;
        }

        int random_product_num = 0;
        do { random_product_num = utils::random_int(1, num_products); }
        while (product_num == random_product_num);
        product_num = random_product_num;
```

```
    }

    inline void mutate_usp_suite_num()
    {
        if (utils::random() >= p_usp_suite_mut) {
            return;
        }

        int random_usp_suite_num = 0;
        do { random_usp_suite_num = utils::random_int(1, num_usp_suites); }
        while (usp_suite_num == random_usp_suite_num);
        usp_suite_num = random_usp_suite_num;
    }

    inline void mutate_num_batches()
    {
        if (utils::random() < p_plus_batch_mut) {
            num_batches += 1;
        }

        if (num_batches > 0 && utils::random() < p_minus_batch_mut) {
            num_batches -= 1;
        }
    }

    int num_products;
    int num_usp_suites;
    double p_product_mut;
    double p_usp_suite_mut;
    double p_plus_batch_mut;
    double p_minus_batch_mut;
};
```

Algorithm B2. C++ implementation of a gene used in Chapter 5.

```
#include <utility>
#include "utils.h"

struct SingleSiteSimpleGene
{
    SingleSiteSimpleGene() {}

    SingleSiteSimpleGene(
        int num_products,
        double p_product_mut,
        double p_plus_batch_mut,
        double p_minus_batch_mut

    )
    {
        this->num_products = num_products;
        this->num_batches = 1;
        this->p_product_mut = p_product_mut;
        this->p_plus_batch_mut = p_plus_batch_mut;
        this->p_minus_batch_mut = p_minus_batch_mut;
        this->product_num = utils::random_int(1, num_products);
    }

    SingleSiteSimpleGene make_new()
    {
        return std::move(
            SingleSiteSimpleGene(
                num_products,
                p_product_mut,
```

```
                p_plus_batch_mut,
                p_minus_batch_mut
            )
        );
    }

    inline void Mutate()
    {
        mutate_product_num();
        mutate_num_batches();
    }

    int product_num;
    int num_batches;

private:
    inline void mutate_product_num()
    {
        if (utils::random() >= p_product_mut) {
            return;
        }

        int random_product_num = 0;
        do { random_product_num = utils::random_int(1, num_products); }
        while (product_num == random_product_num);
        product_num = random_product_num;
    }

    inline void mutate_num_batches()
    {
        if (utils::random() < p_plus_batch_mut) {
            num_batches += 1;
        }

        if (num_batches > 1 && utils::random() < p_minus_batch_mut) {
            num_batches -= 1;
        }
    }

    int num_products;
    double p_product_mut;
    double p_plus_batch_mut;
    double p_minus_batch_mut;
};
```

B.2. Variable-length Chromosome

The variable-length chromosome described in Chapter 4 and later used in Chapters 5 and 6 has been implemented using a template class (see *BaseChromosome* in Figure B.2). Templates in C++ programming language make classes more abstract by letting the user define the behavior of the class without specifically knowing what datatype will be handled by the operators/methods of the class. This way the variable-length chromosome can be compatible with genes comprising varying number of mutation operators, parameters, and encoded variables. The ability to have a single class that can handle several different gene types means the codebase is easier to maintain and more reusable. The same variable-length chromosome base can be applied to different biopharmaceutical capacity planning and scheduling problems, e.g. Chapter 4 and 5, by specifying a problem-specific gene datatype. Gene parameter values are passed from the chromosome using a variadic parameter pack which improves the abstraction even further.

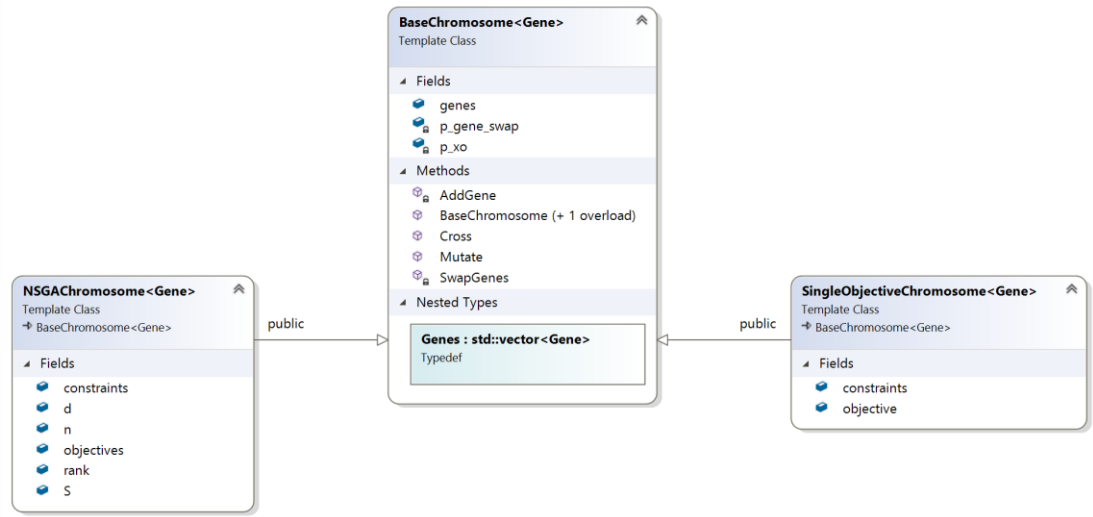


Figure B.2. Variable-length chromosome.

NSGAChromosome and *SingleObjectiveChromosome* classes extend the *BaseChromosome* class with GA-specific attributes, e.g. *NSGAChromosome* add attributes required by the NSGA-II algorithm.

Algorithm B.3. C++ implementation of the variable-length chromosome base.

```
#include <vector>
#include <cstdlib>
#include <utility>
#include <algorithm>
#include <functional>
#include "utils.h"

template<class Gene>
class BaseChromosome
{
public:
    typedef std::vector<Gene> Genes;

    explicit BaseChromosome() {}

    template<class... GeneParams>
    explicit BaseChromosome(
        int starting_length,
        double p_xo,
        double p_gene_swap,
        GeneParams... params
    ) :
        p_xo(p_xo),
        p_gene_swap(p_gene_swap)

    {
        while (starting_length-- > 0) {
            genes.push_back(std::move(Gene(params...)));
        }
    }

    inline void Cross(BaseChromosome &other)
    {
        if (utils::random() > p_xo) {
            return;
        }

        if (genes.size() < 2 || other.genes.size() < 2) {
            return;
        }

        int i;

        if (genes.size() < other.genes.size()) {
            for (i = 0; i != genes.size(); ++i) {
                if (utils::random() <= 0.50) {
                    std::swap(genes[i], other.genes[i]);
                }
            }
            for (; i != other.genes.size(); ++i) {
                if (utils::random() <= 0.50) {
                    genes.push_back(other.genes[i]);
                }
            }
        }
        else {
            for (i = 0; i != other.genes.size(); ++i) {
                if (utils::random() <= 0.50) {
                    std::swap(genes[i], other.genes[i]);
                }
            }
            for (; i != genes.size(); ++i) {
                if (utils::random() <= 0.50) {
```

Algorithm B.3. (continued) C++ implementation of the variable-length chromosome base.

```
                    other.genes.push_back(genes[i]);
                }
            }
        }
    }

    inline void Mutate()
    {
        for (auto &gene : genes) {
            gene.Mutate();
        }

        AddGene();
        SwapGenes();
    }

    Genes;

private:
    inline void AddGene()
    {
        genes.push_back(genes.back().make_new());
    }

    inline void SwapGenes()
    {
        if (utils::random() >= p_gene_swap) {
            return;
        }

        int g1 = 0, g2 = 0;
        do {
            g1 = utils::random_int(0, genes.size() - 1);
            g2 = utils::random_int(0, genes.size() - 1);
        } while (g1 == g2);

        std::swap(genes[g1], genes[g2]);
    }

    double p_xo;
    double p_gene_swap;
};
```

B.3. Genetic Algorithm

A base GA class was developed to improve the re-usability of the codebase and to provide a standardised interface for single- and multi-objective GAs. It was also implemented as a template class to make it possible to specify different scheduling heuristics, i.e. as *FitnessFunction* (see Figure B.3 below). This way the codebase for the GA could be re-used to solve the various biopharmaceutical scheduling problems applying different scheduling heuristics.

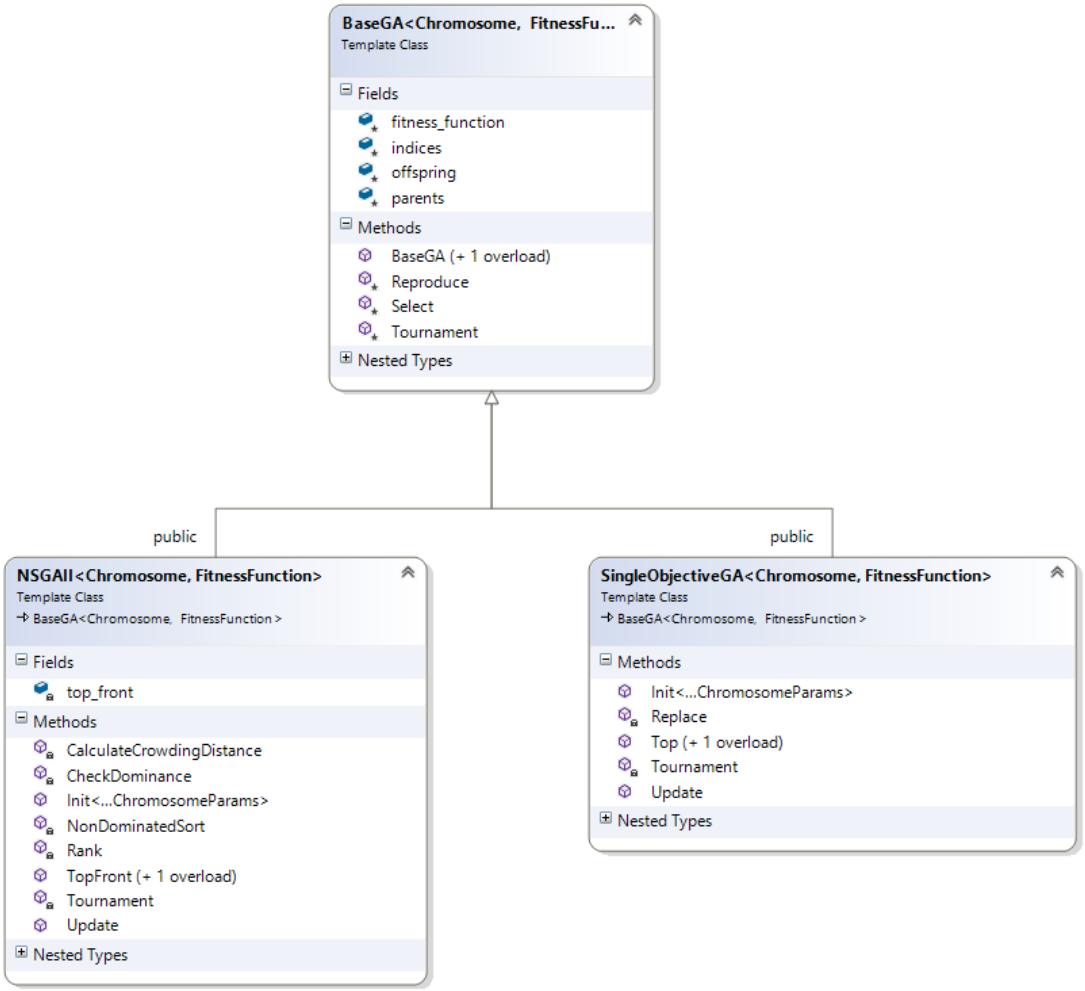


Figure B.3. Diagrams of Genetic Algorithms developed in this thesis.

B.4. API Usage Examples

This section provides practical examples (Algorithm B.4 and B.5) of how the GA-based DST is used to solve the case studies described in Chapters 4 and 5 using the Python API designed in this thesis.

Algorithm B.4. Demonstration of how the GA-based Decision Support Tool is used to solve a single-objective scheduling problem described in the case study 1 of Chapter 4 using Python Application Programming Interface designed in this work.

```
# Import Python Pandas library for data I/O
import pandas as pd

# Import the desired model from the GA-based DST
from biopharma_scheduling.single_site.deterministic import DetSingleSiteMultiSuite

# Data setup
start_date = '2016-11-02'
demand = pd.read_csv('demand.csv', index_col='date')
product_data = pd.read_csv('product_data.csv')
usp_changeover_days = pd.read_csv('usp_changeover_days.csv')
dsp_changeover_days = pd.read_csv('dsp_changeover_days.csv')
num_usp_suites = 2
num_dsp_suites = 2

# Specify which objective or objectives to optimise.
# The objectives are pre-defined by the imported model
# For example, calling DetSingleSiteMultiSuite.AVAILABLE_OBJECTIVES
# will list all available objectives which can be minimised or
# maximised by specifing the coefficient of -1 or 1, respectively
# {
#     'total_backlog_penalty',
#     'total_batch_backlog',
#     'total_batch_supply',
#     'total_batch_throughput',
#     'total_batch_waste',
#     'total_changeover_cost',
#     'total_cost',
#     'total_production_cost',
#     'total_profit',
#     'total_revenue',
#     'total_storage_cost',
#     'total_waste_cost'
#  }
objectives = {
    'total_profit': 1 # Coefficient of 1 indicates maximisation
}

# Specify the GA parameters
ga_params = {
    'num_runs': 20,
    'popsize': 100,
    'num_gens': 100,
    'starting_length': 1,
    'p_xo': 0.026776,
    'p_product_mut': 0.004667,
    'p_usp_suite_mut': 0.015991,
    'p_plus_batch_mut': 0.896385,
```

Algorithm B.4. (continued) Demonstration of how the GA-based Decision Support Tool is used to solve a single-objective scheduling problem described in the case study 1 of Chapter 4 using Python Application Programming Interface designed in this work.

```
    'p_minus_batch_mut': 0.853790,
    'p_gene_swap': 0.403328
}

# Create an instance of the model
model = DetSingleSiteMultiSuite(
    **ga_params, # unpacks GA parameters set-up earlier
    random_state=7, # fix the seed for random number generator
    num_threads=-1, # will evaluate solutions in parallel using all available cores
    verbose=True, # will report progress status to the user
)

# Fit the model using the GA params and the data defined earlier
model.fit(
    start_date,
    objectives,
    num_usp_suites,
    num_dsp_suites,
    demand,
    product_data,
    usp_changeover_days,
    dsp_changeover_days
)

# After the model has been fit, the solutions, i.e. schedules,
# will be contained in model.schedules list. If only one objective
# was specified for the scheduling problem then it was solved using
# a single-objective GA.
#
# model.schedules will contain a single best solution that was found
# during the specified number of GA runs.
schedule = model.schedules[0]

# Will list the estimated values of all objectives
schedule.objectives

# Will list a production schedule table for campaigns
schedule.campaigns

# Will list a production schedule table for individual batches
schedule.batches

# Will display a Gantt chart
schedule.campaigns_gantt()

# Will display product inventory, supply, waste, and backlog profiles
schedule.batch_inventory
schedule.batch_supply
schedule.batch_waste
schedule.batch_backlog
```

Algorithm B.5. Demonstration of how the GA-based Decision Support Tool is used to solve a multi-objective biopharmaceutical scheduling problem with constraints described in Chapter 5 using Python Application Programming Interface designed in this work.

```
# Import Python Pandas library for data I/O
import pandas as pd

# Import the desired model from the GA-based DST
from biopharma_scheduling.single_site.deterministic import DetSingleSiteSimple

# Data setup
start_date = '2016-12-01'
demand = pd.read_csv('demand.csv', index_col='date')
inventory_targets = pd.read_csv('inventory_targets.csv', index_col='date')
product_data = pd.read_csv('product_data.csv')
changeover_days = pd.read_csv('changeover_days.csv')

# Specify which objective or objectives to optimise.
# The objectives are pre-defined by the imported model
# For example, calling DetSingleSiteSimple.AVAILABLE_OBJECTIVES
# will list all available objectives which can be minimised or
# maximised by specifing the coefficient of -1 or 1, respectively:
# {
#     'total_backlog_penalty',
#      'total_cost',
#      'total_inventory_penalty',
#      'total_kg_backlog',
#      'total_kg_inventory_deficit',
#      'total_kg_supply',
#      'total_kg_throughput',
#      'total_kg_waste',
#      'total_production_cost',
#      'total_profit',
#      'total_revenue',
#      'total_storage_cost',
#      'total_waste_cost'
# }
objectives = {
    'total_kg_throughput': 1, # maximise
    'total_kg_inventory_deficit': -1 # minimise
}

constraints = {
    'total_kg_backlog': [-1, 0], # total_kg_backlog <= 0
    'total_kg_waste': [-1, 0] # total_kg_waste <= 0
}

# Specify the GA parameter
ga_params = {
    'num_runs': 50,
    'num_gens': 1000,
    'popsize': 600,
    'starting_length': 1,
    'p_xo': 0.108198,
    'p_product_mut': 0.0,
    'p_plus_batch_mut': 0.608130,
    'p_minus_batch_mut': 0.765819,
    'p_gene_swap': 0.471346,
}

# Create an instance of the model
model = DetSingleSiteSimple(
    **ga_params, # unpacks GA parameters set-up earlier
```

Algorithm B.5. (continued) Demonstration of how the GA-based Decision Support Tool is used to solve a multi-objective biopharmaceutical scheduling problem with constraints described in Chapter 5 using Python Application Programming Interface designed in this work.

```
    random_state=7, # fix the seed for random number generator
    num_threads=-1, # will evaluate solutions in parallel using all available cores
    verbose=True, # will report progress status to the user
)

# Fit the model using the GA params and the data defined earlier
model.fit(
    start_date,
    objectives,
    kg_demand,
    product_data,
    changeover_days,
    kg_inventory_target,
    constraints
)

# After the model has been fit, the solutions, i.e. schedules,
# will be contained in model.schedules list. Since more than one
# objective is specified, the scheduling problem will be solved
# as a multi-objective problem.
#
# model.schedules will contain a single best Pareto
# front which is a result of the best Pareto fronts from each
# individual run combined together and sorted again using a
# non-dominated  sorting algorithm.
#
# Sorting the Pareto objective using either one of the two
# objectives, will make it easier to obtain the boundary
# solutions X and Y
sorted_schedules = sorted(model.schedules, key=lambda schedule:
schedule.objectives['total_kg_throughput'].values[0])
solution_x = sorted_schedules[0]
solution_y = sorted_schedules[-1]

# Will list the estimated values of all solution X objective
solution_x.objectives

# Will list a solution X production schedule table
solution_x.campaigns

solution_x.batches

# Will display a Gantt chart for solution X campaigns
solution_x.campaigns_gantt()

# Will display a Gantt chart for solution X tasks, e.g. inoculation, USP, DSP
solution_x.tasks_gantt()

# Solution X inventory, supply, waste, and backlog profiles
solution_x.kg_inventory
solution_x.kg_supply
solution_x.kg_waste
solution_x.kg_backlog
```

Appendix C

This appendix summarises the mathematical model presented by Lakhdar et al. (2005).

C.1. Production Constraints

Constraints 1 and 2 represent the manufacture of product in *USP* and *DSP* suites. Upstream production, B_{ipt}, and downstream production, B_{jpt}, are represented by continuous rates of production, CR_p and FR_p, which are combined with their respective *USP* and *DSP* lead times, α_p and β_p, and *USP* and *DSP* production times, CT_{ipt} and FT_{jpt}. Constraints 3 and 4 activate lead time in *USP* suite *i* and *DSP* suite *j* if the same product *p* has not been manufactured in the preceding time period, *t – 1*. Constraints 5 and 6 ensures that only one product *p* is produced in any *USP* suite *i* and *DSP* suite *j* at any time period *t*.

$$B_{ipt} = Z_{ipt} + CR_p\left(CT_{ipt} - \alpha_p Z_{ipt}\right) \quad \forall\, i,p,t \tag{1}$$

$$B_{jpt} = Z_{jpt} + FR_p\left(FT_{jpt} - \beta_p Z_{jpt}\right) \quad \forall\, j,p,t \tag{2}$$

$$Z_{ipt} \geq Y_{ipt} - Y_{ip,t-1} \quad \forall\, i,p,t \tag{3}$$

$$Z_{jpt} \geq Y_{jpt} - Y_{jp,t-1} \quad \forall\, j,p,t \tag{4}$$

$$\sum_p Y_{ipt} \leq 1 \quad \forall\, i,t \tag{5}$$

$$\sum_p Y_{jpt} \leq 1 \quad \forall\, j,t \tag{6}$$

C.2. Timing Constraints

Constraints 7 and 8 represent the appropriate minimum and maximum production times for *USP* and *DSP* suites, which are only activated when Y_{ipt} and Y_{jpt} are equal to 1. Constraints 9 and 10 ensure that the total *USP* or *DSP* time does not exceed the specified production time horizon, H_t.

$$CT_p^{min} Y_{ipt} \leq CT_{ipt} \leq CT_p^{max} Y_{ipt} \quad \forall\, i, p, t \tag{7}$$

$$FT_p^{min} Y_{jpt} \leq FT_{jpt} \leq FT_p^{max} Y_{jpt} \quad \forall\, j, p, t \tag{8}$$

$$\sum_p CT_{ipt} \leq H_t \quad \forall i, t \tag{9}$$

$$\sum_p FT_{jpt} \leq H_t \quad \forall j, t \tag{10}$$

C.3. Storage Constraints

Constraints 11 and 12 enforce an inventory balance in upstream and downstream production and force the total downstream production to meet the product demand. Constraints 13 and 14 ensure that the amount of upstream and downstream product stored over timer period *t* is positive and below the maximum available storage capacities, C_p and F_p. Both upstream and downstream product inventory is constrained by the limited product shelf-life. Constraints 15 and 16 ensure the total amount of stored upstream product and downstream product is used after the next ζ_p or σ_p time periods, respectively.

$$CI_{pt} = CI_{p,t-1} + \sum_i B_{ipt} - \frac{1}{\lambda_p} \sum_j B_{jpt} - CW_{pt} \quad \forall\, p, t \tag{11}$$

$$FI_{pt} = FI_{p,t-1} + \sum_j B_{jpt} - S_{pt} - FW_{pt} \quad \forall\, p, t \tag{12}$$

$$0 \leq CI_{pt} \leq C_p \quad \forall\, p, t \tag{13}$$

$$0 \leq FI_{pt} \leq F_p \quad \forall\, p, t \tag{14}$$

$$CI_{pt} \leq \sum_j \sum_{\theta=t+1}^{t+\zeta_p} B_{jp\theta} \quad \forall\, p, t \tag{15}$$

$$FI_{pt} \leq \sum_{\theta=t+1}^{t+\sigma_p} S_{p\theta} \quad \forall\, p, t \tag{16}$$

C.4. Backlog Constraints

Constraint 17 penalises the amount of product *p* that was late for delivery at time period *t*, Δ_{pt}.

$$\Delta_{pt} = \Delta_{p,t-1} + D_{pt} - S_{pt} \quad \forall\, p, t \tag{17}$$

C.5. Objective Function

The objective function is to maximise profit which is equal to the difference between total sales and total operating costs. All costs and prices are in relative monetary units (RMU).

$$\max Profit = \sum_p \sum_t (\nu_p S_{pt} - \sum_i \eta_p B_{ipt} - \sum_i \psi_p Z_{ipt} - \sum_j \eta_p B_{jpt} - \sum_j \psi_p Z_{jpt} - \rho_p CI_{pt} - \omega_p FI_{pt} - \delta_p \Delta_{pt} - \tau_p (CW_{pt} + FW_{pt})) \tag{18}$$

Appendix D

This appendix summarises the mathematical model presented by Lakhdar et al. (2007).

D.1. Production Constraints

Constraint 1 represents biopharmaceutical production. The number of batches produced in facility *i* of product *p* at time period *t, B_{ipt}*, is represented by a continuous production rate, r_{ip}, production lead time, $\alpha_{ip,}$ and production time T_{ipt}. Constraint 2 converts the integer number of batches, B_{ipt}, into kilograms, K_{ipt}, using a yield conversion factor, yd_{ip}. Constraint 3 activates lead time in facility *i* if the same product *p* has not been manufactured in the preceding time period, *t – 1*. Constraint 4 ensures that only one product *p* is produced in any facility *i* at any time period *t*.

$$B_{ipt} = Z_{ipt} + r_{pt}\left(T_{ipt} - \alpha_{ip}Z_{ipt}\right) \quad \forall\, i, p \in PI_i, t \in TI_i \tag{1}$$

$$K_{ipt} = B_{ipt}yd_{ip} \quad \forall\, i, p \in PI_i, t \in TI_i \tag{2}$$

$$Z_{ipt} \geq Y_{ipt} - Y_{ip,t-1} \quad \forall\, i, p \in PI_i, t \in TI_i \tag{3}$$

$$\sum_{p \in PI_i} Y_{ipt} \leq 1 \quad \forall\, i, t \in TI_i \tag{4}$$

D.2. Timing Constraints

Constraints 5 and 6 represent the appropriate minimum and maximum campaign durations, T_{ip}^{min} and T_{ip}^{max}, which are only activated when Y_{ipt} is equal to 1.

$$T_{ip}^{min}Y_{ipt} \leq T_{ipt} \quad \forall\, i, p \in PI_i, t \in TI_i \tag{5}$$

$$T_{ipt} \leq \min\left\{T_{ip}^{max}, H_t\right\}Y_{ipt} \quad \forall\, i, p \in PI_i, t \in TI_i \tag{6}$$

D.3. Storage Constraints

Constraint 7 enforces inventory balance for production and forces the total production to meet the product demand. Constraint 9 enforces that the amount of product *p* in inventory at time period *t* is below the maximum storage capacity, C_p, while the constraint 10 ensures that the global storage capacity, C_p^{tot}, is not exceeded. The duration a product can be is stored in inventory is limited by the constraint 10.

$$I_{pt} = I_{p,t-1} + \sum_i K_{ipt} - S_{pt} - W_{pt} \quad \forall\, p \in PI_i, t \in TI_i \tag{7}$$

$$0 \leq I_{pt} \leq C_p \quad \forall\, p, t \tag{8}$$

$$0 \leq \sum_p I_{pt} \leq C_p^{tot} \quad \forall\, t \tag{9}$$

$$I_{pt} \leq \sum_{\theta=t+1}^{t+\zeta_p} S_{p\theta} \quad \forall\, p, t \tag{10}$$

D.4. Backlog Constraints

Constraint 11 penalises the amount of product *p* that was late for delivery at time period *t*, Δ_{pt}.

$$\Delta_{pt} = \pi_p \Delta_{p,t-1} + D_{pt} - S_{pt} \quad \forall\, p, t \tag{11}$$

D.5. Objective Function

The objective function is to maximise profit which is equal to the difference between total sales and total operating costs. All costs and prices are in relative monetary units (RMU).

$$\max Profit = \sum_p \sum_{t \in TI_i} (\nu_p S_{pt} - \rho_p I_{pt} - \delta_p \Delta_{pt} - \sum_{i \in IP_p} (\eta_{ip} B_{ipt} + \psi_{ip} Z_{ipt})) \tag{12}$$