Learning to Optimise Networked Systems

Victor-Alexandru Darvariu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Computer Science
University College London

May 10, 2023

I, Victor-Alexandru Darvariu, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Many systems based on relations between connected entities find a natural representation in graphs, which has led to the development of mathematical and statistical tools for understanding their structure and the phenomena that take place over them. There is comparatively little work on the study of optimising the outcome of processes on graphs with respect to a given objective function. Problems of this nature are combinatorial optimisation tasks, which are challenging for systems beyond a trivial size due to the rapid growth of the solution space.

Traditional ways of approaching such problems use either heavily tailored, objective-specific approaches or generic metaheuristics. Machine learning and decision-making algorithms have begun to emerge as an alternative data-driven paradigm for navigating the search space, allowing for generalisation between related problems and effective scaling to larger instances than those seen during training.

This thesis contributes problem formulations, solution methods, and learning representations for approximately solving several graph combinatorial optimisation problems. We first address constructing a graph for optimising a structural objective such as resilience or efficiency. We propose a deep reinforcement learning approach that uses graph neural networks as a key component for generalisation and a complementary extension of Monte Carlo Tree Search, which is suitable for spatial networks. Next, we study public goods games over networks, discussing an approach for effective imitation learning of policies mapping graph states to node selection actions. Finally, we focus on data-driven routing of flows over graphs, proposing a novel graph neural network architecture for this family of problems.

Our evaluation results show significant advantages for the proposed approaches over prior methods, representing a contribution to the broader area of machine learning for combinatorial optimisation. The research discussed in this thesis may

find meaningful applications in domains as diverse as structural engineering, urban planning, operations research, and computer networking.

Impact Statement

Given the generality of graph representations, the work herein may find applications in diverse fields, in ways that are difficult to anticipate. Below, we make an attempt to summarise the areas in which we foresee a path to impact.

Overall, the present thesis is highly relevant to the operations research discipline (which significantly overlaps with industrial engineering and management science), since it contributes flexible approaches for decision-making on networks such that resources and budgets are efficiently used. Possible application areas include the optimisation of supply chains, logistics and transportation systems, and network engineering. Given the practical relevance of these fields of study, we expect impact might occur not only in an academic setting, but that technologies based on the proposed methods may have an impact in practice. We envision that such technologies can lead to reduced operational costs and usage of resources (natural or human).

The work on graph construction presented in Chapters 3 and 4 is highly relevant to engineering and the physical sciences. There are possible applications of the proposed techniques in designing structures, both at a macroscopic and microscopic level. With respect to the former, we mention structural engineering for the stability and resilience of human-made buildings, as well as the robustness of transportation networks. Regarding the latter, similar techniques may be used to discover materials and compounds with desirable properties.

Potentially, the proposed work may also find applications in decision-making and policy design of local authorities and governments. Approaches similar to the work in Chapter 5 can be used to run simulations that assess the right means of targeting interventions in a networked system such as to encourage certain desirable outcomes. As an example scenario, important health factors such as smoking and obesity are known to be linked to individuals' social networks [71, 72], and hence a

policy-maker could be interested in the best way of targeting a public health campaign. Furthermore, this type of technique may assist authorities in deciding how to best invest and spend available resources to, for example, extend a transportation network or invest in public infrastructure. Such methods may help in making decisions that have a well-specified, objective, target outcome.

We note that the type of mathematical modelling used in this thesis necessarily makes simplifying assumptions about the real world and cannot capture all of its complexity. Due to this, we highlight the importance of involving stakeholders and domain experts in the modelling process when considering practical applications, so that assumptions, risks, and benefits are thoroughly analysed and specified. By doing so, negative impacts may be foreseen and mitigated.

Acknowledgements

The preparation of this thesis is the culmination of an unlikely journey that started nearly ten years ago – one that has challenged and helped me grow in ways I never could have foreseen. It takes a village, and I am fortunate to be able to thank the following people from this metaphorical ledge.

I owe the biggest debt to my parents, Liliana and Marius. Thank you for teaching me the value of education and the art of grit and determination. Thank you also for standing by me even as our worlds diverged as much as they have. *Teza asta e dedicată vouă.* I would also like to offer my deepest gratitude to my much-loved sister Ștefana, who is about to embark on an academic journey of her own, and to whom I wish the best of luck in navigating the stormy waters. Heaps of thanks also go out to Elena and Silvia, Camelia and Petrea, Gianina and Gabi, as well as my wider family. *Mulțumesc!*

To this day, I am still not sure what Mirco saw in my starry eyes when I showed up at his office door all those years back. He bears, by far, the single highest amount of responsibility for me ending up here. I would have never even considered research had it not been for you. Thank you for shaping my thinking, teaching me what questions to ask of the world, inspiring me to aim high, and for your friendship during the tough times. I also owe a lot to Steve, who somehow knows more things than I can hope to accumulate in several lifetimes. Thank you for your truly extraordinary support and feedback. I also owe a debt to my mentors in computer science going all the way back. Thank you to Dan, Liliana (whose memory will live on), and Zizi.

Thanks are also owed to the colleagues that I have had the pleasure of sharing an office, meals, and evenings with. Thank you to Abhinav, Beatrice, Benjamin, and Mariflor for the early days; Alessandro, Charlie, Christoffel, Liza, and Olivia for the later ones. Thanks to everyone at the Turing, especially Andrew, Nicolas, Ryan, Sam,

Shunee, and the football gang. Thank you also to Antonio, Brian, Dimitrios, Maria, Mariano, and Shahar – spending a summer at Spotify was a dream come true. I also owe an immense amount of gratitude to the friends that have kept me afloat during these years. Thank you to Adelina, Alex, Albert, Amar, Andrei, Bijal, Bristena, Călin, Cecilia, Florin, Francisc, Ioana, Jimmy, Rareș, Teona, Toni, and Vlad.

Finally, I would like to thank my life partner, Maria, for her enduring love and support. I do not see how I possibly could have done this without you.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

Table 1 introduces the acronyms and abbreviations used throughout the present thesis.

Term	Meaning
AI	Artificial Intelligence
ML	Machine Learning
SL	Supervised Learning
RL	Reinforcement Learning
IL	Imitation Learning
MDP	Markov Decision Process
MC	Monte Carlo
TD	Temporal-Difference
DQN	Deep Q-Network
MCTS	Monte Carlo Tree Search
UCT	Upper Confidence Bounds for Trees
ANN	Artificial Neural Network
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
MSE	Mean Squared Error
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
GAN	Generative Adversarial Network
LSTM	Long Short-Term Memory
GNN	Graph Neural Network

MPNN	Message Passing Neural Network
mIS	Maximal Independent Set
MIS	Maximum Independent Set
CF	Critical Fraction
LCC	Largest Connected Component
TSP	Travelling Salesperson Problem
VRP	Vehicle Routing Problem
PGG	Public Goods Game
NE	Nash Equilibrium
PSNE	Pure Strategy Nash Equilibrium
ER	Erdős–Rényi generative graph model
BA	Barabási–Albert generative graph model
WS	Watts–Strogatz generative graph model
KH	Kaiser–Hilgetag generative graph model
MCNF	Multi-Commodity Network Flow
LP	Linear Programming

Table 1: Main acronyms and abbreviations used throughout the present thesis.

Notation

Table 2 presents an overview of the notation used throughout this thesis, with the specific chapters introducing additional notation where necessary. It covers basic mathematical notation, elements related to graphs and, finally, notation specific to RL, planning, and deep learning.

Example	Explanation
$\mathbb{N}, \mathbb{R}$	The sets of natural and real numbers.
x	Lowercase letters typically denote scalars, with several exceptions in accordance with conventional notation in the fields the present thesis touches on. Most notably, as shown in subsequent rows, we use v to indicate graph vertices and e to indicate graph edges; as well as s and a to indicate a generic state and action in RL.
$\mathbf{x}$	Bold lowercase letters denote vectors.
x_i	i-th element of a vector.
$[x, y]$	Vector containing elements x and y.
$[\mathbf{x} \Vert \mathbf{y}]$	Concatenation of vectors $\mathbf{x}$ and $\mathbf{y}$.
$\mathbf{X}$	Bold uppercase letters denote matrices.
$X_{i,j}$	Matrix element at row i and column j.
(x, y)	Tuple.
$\{x\}, \{x, y, z\}$	Sets (with the former representing a singleton).
$\lvert X \rvert$	Cardinality of set X.
$\hat{x}$	Estimate of x.
$\bar{x}$	Sample mean of x.
$\text{abs}(x)$	Absolute value.
$x \bmod y$	Remainder obtained when dividing x by y.

δ_{xy}	Kronecker delta, which is equal to 1 if $x = y$, and 0 otherwise.
$d(X_1, X_2)$	Distance between points X_1 and X_2 (Euclidean unless otherwise specified).

$G = (V, E)$	A *graph* (also referred to as *network*), in which V is the set of *nodes* and E is the set of *edges*. Graphs are undirected unless otherwise specified.
v, v_i	A specific node (also called *vertex*) in the graph. Note that, when used together with v, the subscript i indicates a particular node, rather than the value at index i in a vector.
e, $e_{i,j}$	A specific edge in the graph. Note that, when used together with e, the subscript i, j indicates the edge between v_i and v_j, rather than the value at index i, j in a matrix.
$\|V\|$, N	Number of nodes in the graph.
$\|E\|$, m	Number of edges in the graph.
$\deg(v_i)$	Degree of node v_i.
$\mathbf{A}$	Adjacency matrix of the graph.
$\mathbf{L}$	Graph Laplacian. The entry $L_{i,j}$ is equal to $\deg(v_i)$ if $i = j$, -1 if $i \neq j$ and $e_{i,j} \in E$, and 0 otherwise.
$\mathcal{F}$	Objective function defined on a graph (typically, the maximisation objective).
ξ	A permutation of the graph nodes.
$\mathcal{N}(v_i)$	Open neighbourhood of node v_i, which contains all nodes adjacent to v_i.
$\mathcal{N}[v_i]$	Closed neighbourhood of node v_i, which contains v_i and all nodes adjacent to v_i.
$\mathbf{x}_{v_i}, \mathbf{x}_{e_{i,j}}$	Feature vectors associated with node v_i and edge $e_{i,j}$. Note the use of subscripts to indicate nodes and edges.
$\mathbf{h}_{v_i}^{(l)}$	Embedding vector for node v_i, as computed by layer l of a representation learning method.
$c(v_i)$, $c(e_{i,j})$	Costs associated with vertex v_i or edge $e_{i,j}$.
$\mathcal{G}$	A set of graphs.

$\mathcal{G}^{\text{train}}$	A training dataset of graphs.
t	Discrete timestep.
T	The terminal timestep.
s, S_t	A state in a Markov Decision Process (MDP), with the latter indicating the state at a specific timestep.
G_t	The graph at a specific timestep. Note that it does not denote the return, as is common in some RL notations.
a, A_t	An action in an MDP.
r, R_t	A reward in an MDP.
$\mathcal{S}$	The set of possible states in an MDP.
$\mathcal{A}(s)$	The set of possible actions at a given state.
$\mathcal{R}$	The set of all possible rewards.
$P(s'\|s, a)$	The transition function (dynamics).
$R(s, a)$	The reward function.
γ	The discount factor.
H_t	The return received from time t onwards.
$V(s)$	The value function.
$Q(s, a)$	The state-action value function.
π, π_Θ	The policy, possibly parametrised by parameters Θ.
$\pi(a\|s)$	The probability of taking action a in state s.
$C(s)$	Visit count for state s.
$C(s, a)$	Number of times action a was taken in state s.
b, b_t	The budget available to the agent.
α	Learning rate.
$\mathbf{W}, \mathbf{W}_1$	Parameter matrix of a policy or model.
Θ	Complete set of parameters of a given policy π_Θ or a Supervised Learning model.
$\mathcal{L}(\Theta)$	Loss function of a model with respect to parameters Θ.
∇_Θ	Gradient with respect to parameters Θ.

Table 2: Summary of notation used throughout the present thesis.

Chapter 1

Introduction

This section outlines the core topics, research questions, and contributions of the present thesis. We begin with a gentle introduction to networks by discussing the idea of optimising processes taking place over them and traditional methods used to address such problems. We then give an overview of how Machine Learning (ML) approaches may help in devising solutions, as well as the placement of the dissertation in this emerging body of work. We formulate the research questions that we set out to address in this thesis, and give an overview of its outline and contributions. Finally, we close with a list of publications stemming from the investigations that have been carried out in pursuit of the research questions.

1.1 Networks and Combinatorial Optimisation

Euler's work on the bridges of Königsberg, which poses the problem of finding a route through a city that crosses each bridge exactly once, is widely regarded as the genesis of graph theory [39]. The field has since developed as a branch of discrete mathematics, and solutions have been developed for problems such as route-finding [108], colouring [188], and graph enumeration [161]. Going beyond raw topology, nodes and edges in graphs are often associated with attributes: for example, an edge can be associated with the value of a distance metric [23]. Enriched with such features, graphs become powerful formalisms able to represent a variety of systems.

This flexibility led to their usage in fields as diverse as computer science, biology,

and the social sciences [263]. The term *network* is used interchangeably with *graph*, especially when discussing a specific instance or application.[1]

The modern discipline of *network science* [263] draws on concepts from a variety of fields in order to perform fundamental studies of graph-structured systems, builds predictive models and algorithms, and applies them to practical problems. To name but a few examples: probability theory and statistical physics have been used to analytically examine the properties and limit behaviour of networks [20]; graph algorithms and data structures, with origins in theoretical computer science, find broad uses when using network modelling [118]; insights into networks can be leveraged for optimisation problems that are relevant in an operational context, such as supply and logistics chains [7].

Methods from network science allow us to formally characterise *processes taking place over a graph*. For example, decision-makers might be interested in the global structural properties such as the *efficiency* with which the network exchanges information, or its *robustness* when network elements fail, aspects crucial to infrastructure networks [223, 8]. One can also use the graph formalism to model *flows* of quantities such as packets or merchandise, relevant in a variety of computer and logistics networks [6]. Taking a decentralised perspective, we may be interested in the individual and society-level outcome of *network games*, in which a network of individuals take selfish decisions in order to maximise their gain [185].

Suppose that we consider such a global process and aim to optimise its outcome by intervening in the network. For example, a local authority might decide to add new connections to a road network with the goal of minimising average trip time or congestion, or a policy-maker might intervene in a social network in order to encourage certain outcomes. These are *combinatorial optimisation* problems, which involve choosing a solution out of a large, discrete space of possibilities such that it maximises the value of a given *objective function*. Conceptually, such problems are easy to define but very challenging to solve, since one cannot simply enumerate all possible solutions beyond the smallest of graphs.

Some of the most well-known combinatorial optimisation problems, such as

[1] Regarding the difference between the terms, "graph" is more accurately used to refer to the mathematical abstraction, while "network" refers to a realisation of this general concept, such as a particular social network. The terms are synonymous in general usage [18, Chapter 2.2].

the Travelling Salesperson Problem (TSP) [308], are known to be *NP-hard* [196]. This means that, given a problem instance and candidate solution, one can check its correctness in polynomial time with respect to the inputs. Finding such a solution in the first place, however, is computationally intensive – no polynomial time algorithms are known for NP-hard problems [134].[2] Problems of this type bear relevance in many areas – the TSP, for example, has been applied in circuit design [62] and bioinformatics [4]. A significant body of work is devoted to solving them.

The lines of attack for such problems can be divided into the following categories:

- *Exact methods*: approaches that solve the problem exactly, i.e., will find the globally optimal solution if it exists. These include exact search algorithms such as brute force search. Notably, if the problem of interest has a linear objective, one can formulate it as an (integer) linear program [333], for which efficient solving methods such as the simplex method [91] and branch-and-bound [221] exist. Typically, exact methods only work well for small to medium-sized problem instances.

- *Heuristics* [271] and *approximation algorithms* [*361*]: approaches that do not guarantee to find the optimal solution, but instead find one in a best-effort fashion. For the latter category, one can also obtain theoretical guarantees on the approximation ratio between the obtained solution and the optimal one. Such approaches make use of insights about the structure of the problem and objective function at hand, and can typically scale to large problem instances.

- *Metaheuristics*: methods that, unlike heuristics, do not make any assumptions about the problem and objective at hand, and instead are generic [40, 36]. Notable examples include local search based methods (such as greedy search, hill climbing, simulated annealing [211]) and population-based approaches, many of which are nature-inspired (such as evolutionary algorithms [14] and ant colony optimisation [109]). Their generic formulation makes them widely applicable, but they are typically outperformed by algorithms that are based on some knowledge of the problem, if indeed it is available.

[2] Unless *P=NP*, one of the famous unsolved problems in computer science.

1.2 Machine Learning for Combinatorial Optimisation: Beyond Canonical Problems

In recent years, ML has started to emerge as a valuable tool in approaching combinatorial optimisation problems, with researchers in the field anticipating its impact to be transformative [31, 58]. Worthy of note are the following relationships and "points" of integration at the intersection of ML and combinatorial optimisation:

1. *ML models can be used to speed up and improve existing algorithms* by data-driven learning for improving components of classic algorithms, replacing hand-crafted expert knowledge. Examples include, for exact methods, learning to perform variable subset selection in Column Generation [255] or biasing variable selection in the branch-and-cut method for Mixed Integer Linear Programs [204]. Often, such works adopt the Supervised Learning (SL) paradigm.

2. *ML can enable the discovery of new algorithms* through the use of Reinforcement Learning (RL), another ML paradigm. Broadly speaking, RL is a mechanism for producing goal-directed behaviour through trial and error [326]. In this framework, one formulates the problem of interest as a Markov Decision Process (MDP), which can be solved in a variety of ways. In the RL paradigm, an agent interacts by means of actions with an uncertain environment, receiving rewards that are proportional to the optimality of its actions; the objective of the agent is to adjust its behaviour so as to maximise the sum of rewards received. Framing combinatorial optimisation problems as decision-making processes can enable the automatic discovery of novel algorithms, including for problems that are not yet well-studied or understood.

3. *ML models are fast to evaluate and can be independent of instance size*. This can be exploited for applications where latency is critical and decisions must be made quickly – typically the realm of well-tuned heuristics. Furthermore, the parametrisations of some ML models can be formulated independently of the size of the problem instance, which means that ML models can be applied to instances of a larger size than seen during training.

Two important pieces of the puzzle that have contributed to the feasibility of applying ML to combinatorial optimisation problems on graphs are, firstly, deep

RL algorithms [326] with function approximation such as the Deep Q-Network (DQN) [251] and, secondly, ML architectures able to operate on graphs [159]. When put together, they represent a powerful, synergistic mechanism for approaching such problems while merely requiring that the task can be expressed in the typical MDP decision-making formalism.

Regarding the former, many RL techniques are able to provably converge to the optimal solution; however, their applicability has been limited until relatively recently to small and medium-scale problems. With the advent of deep learning, RL algorithms have acquired powerful generalisation capabilities, and became equipped to overcome the curse of very high-dimensional state spaces. RL approaches combined with deep neural networks have achieved state-of-the-art performance on a variety of tasks, ranging from general game-playing to continuous control [251, 236].

With respect to the latter, architectures designed to operate on non-Euclidean data [50] have brought the successes of ML to the graph domain. Worthy of note are Graph Neural Networks (GNNs), which are based on rounds of message passing and non-linear aggregation [303]. Motivated by these advances, many works adopt such architectures in order to generalise during the learning process across different graphs which may, while being distinct in terms of concrete nodes and edges or their attributes, share similar characteristics.

The majority of work in ML for combinatorial optimisation focusses on canonical problems, either known NP-complete problems (see, e.g., Richard M. Karp's list [196]), or problems for which a reduction to a known NP-complete problem has been devised.[3] Notable examples include Maximum Independent Sets (MIS) [5], Maximum Cut [203, 5], as well as routing problems including the aforementioned TSP [346, 29, 203] and the Vehicle Routing Problem (VRP) [216, 205]. Such problems, however, have been intensely studied – research on solving the TSP alone dates back nearly 70 years to the paper of Dantzig et al. [92]. With a few exceptions [5], even though work on such benchmark problems is important for pushing the limitations

[3]To prove the NP-completeness of these decision problems, Karp derived polynomial-time reductions to the Boolean satisfiability problem (SAT). The construction of such reductions have become a standard way to prove the NP-completeness of a decision problem. The Cook-Levin theorem [79, 228], another cornerstone result in theoretical computer science, established the NP-completeness of SAT, to which the "reductibility hierarchy" of many problems can be traced. [134] gives a catalogue of such problems and an extensive treatment of the area. We also note that, while such tools are useful constructs to formally categorise the difficulty of problems, it is typically impractical to solve them by conversion to a set of Boolean clauses and use of a SAT solver.

of ML-based methods, currently they cannot directly compete with well-established, highly optimised heuristic and exact solvers [193].

The goal of this thesis is instead to devise learning-based approaches to practical problems for which no efficient, performant algorithms are currently known. In doing so, it aims to address the gap in the literature regarding the types of problems for which ML approaches would bring a benefit over prior heuristics and metaheuristics. The key insights behind this thesis are that formulating such problems as decision-making processes and using RL brings a great degree of flexibility in the range of graph combinatorial optimisation problems that can be addressed; that extending well-known RL algorithms to tailor them for certain problem families can bring substantial gains in optimisation capabilities; and, moreover, that carefully designed graph learning representations can further contribute to this goal.

1.3 Research Questions

We are now well-equipped to enunciate the research questions addressed by this thesis. They focus on three underexplored applications of the optimisation of graph processes that are relevant in practical contexts: *network structure, network games, and network flows*. Note that, in other parts of the thesis, we use the acronym ***RQ*** to refer back to a particular research question.

Research Question 1: *Is RL a suitable paradigm for optimising the structure of networked systems?*

Unlike well-studied NP-hard problems, the current state of the art in the area of methods for modifying the structure of a graph so as to optimise a given objective function relies on simple, hand-crafted heuristics [35, 348, 306]. We aim to understand whether the RL paradigm is appropriate for discovering novel heuristic algorithms for such problems, as well as the flexibility of this framework to incorporate different constraints and objectives.

Research Question 2: *How do RL-based methods for graph construction compare with traditional approaches in terms of optimality of the solutions found and computational cost?*

The application of RL to such problems over simple heuristics needs to be justified since it is significantly more complex, both algorithmically and conceptually.

The advantages and disadvantages compared to simpler methods, for example in terms of computational speed and gains in terms of the optimisation objective that they are able to obtain, should be clearly understood.

Research Question 3: *How can we make RL-based approaches for graph construction scale to large graphs? What are the necessary modifications or simplifications?*

For such approaches to be truly impactful, they need to be applicable to large, non-trivially sized graphs. There are several possibilities worth exploring in this space, from adopting decision-time planning algorithms that only consider the fraction of the MDP starting from a state of interest, to constraining the space of possible actions, and applying models trained on small networks (where training is computationally feasible) to larger graphs.

Research Question 4: *Can RL be applied to finding Maximal Independent Sets on graphs that optimise a given objective function? What are possible applications of this problem?*

While the Maximum Independent Set problem has received substantial attention in the literature, there is no known efficient algorithm for finding a Maximal Independent Set (mIS) that optimises a given objective function. Furthermore, there is an interesting connection between mISs and network games: mISs correspond to Nash Equilibria in a networked Public Goods Game [46]. Hence, such a method can also be used to find desirable equilibria in the context of this network game.

Research Question 5: *If we are given demonstrations of an expert policy defined on a set of graphs, how can we efficiently summarise its knowledge so that we may apply it to a set of different graphs? Can a policy derived in this manner have the same performance on this held-out set as the original one on the training graphs?*

Performing RL training *tabula rasa* can be computationally expensive, exhibit high variability, and suffer from sub-optimal exploration. If a well-performing algorithm is already known for a graph problem, we may want to derive a model by Imitation Learning so that it can be quickly applied to other problem instances, or use it to "warm start" RL training. A substantial limitation of policies defined over graphs is their sensitivity to graph size and labelling, constraining their applicability.

Research Question 6: *Are current Graph Neural Networks a suitable learning representation for modelling flows on networks? Assuming a multi-commodity network flow formulation with varying traffic between multiple sources and sinks, what is the right means for providing the demand input data to a model? What is the relationship between graph structure and the performance of such a model?*

Graph Neural Networks have been successfully used for learning to route traffic over graphs, a problem of substantial practical interest in networking and logistics [6, 339, 175]. However, the benefits of such a learning representation over standard architectures such as feedforward neural networks are not well understood, especially in relation to graph structure. It is also not clear how the flows of traffic should be presented as input to a learning architecture.

With the research questions having been articulated, it is also perhaps worth discussing the topics that this thesis *does not* address, and how it relates to other areas of active interest in the community.

One such area is that of graph representation learning [159], which seeks to find appropriate embeddings that may be used for a variety of downstream ML tasks, commonly node and graph classification. The field seeks both to characterise the fundamental limits of such learning architectures, as well as to investigate the design of expressive representations that can guarantee good performance on given end tasks. In contrast, we are mainly interested in addressing the combinatorial optimisation problems themselves, which can also be viewed as novel end tasks in this context. The contributions of the present thesis are largely orthogonal to this area – one can, for example, swap the learning architectures in Chapters 3 and 5 for more effective and efficient alternatives that will be developed by the community over time. An exception in this sense is the approach presented in Chapter 6: we aim to show that learning representations must be aligned with the characteristics of the combinatorial optimisation problem in order to be effective.

Works on neural algorithmic reasoning [344, 342, 182] are also closely related and share some common goals, such as obtaining models that may generalise well to larger problem instances than encountered during training. However, most of the works adopt Supervised Learning, rather than RL. Models are trained to directly

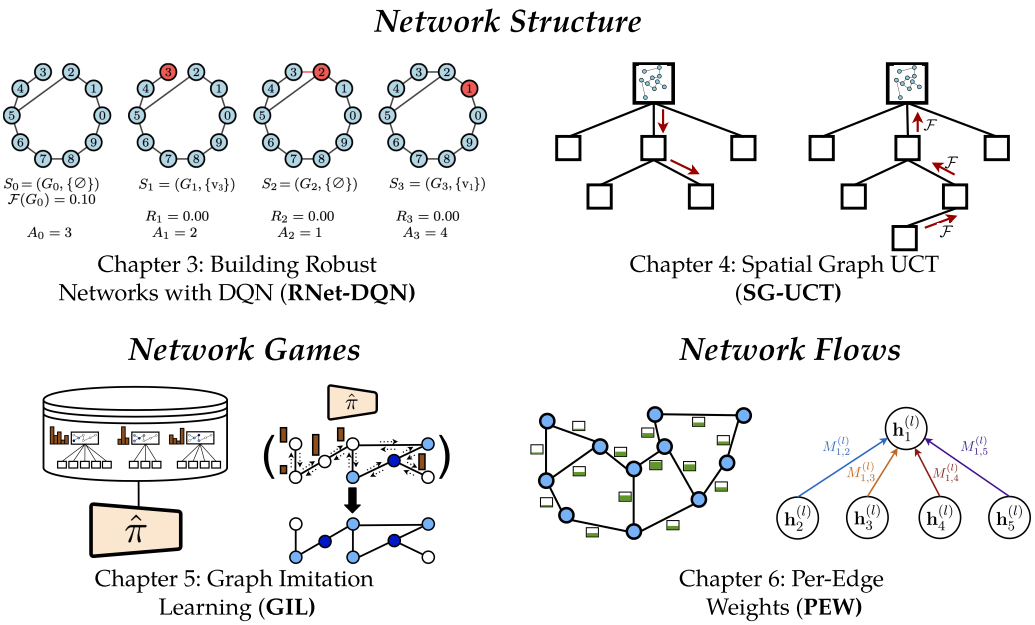


Figure 1.1: Visual summary of the main topics and contributions of the present thesis.

mimic the execution of known algorithms, without the notion of an MDP or, indeed, a policy. Our primary focus is, instead, to design new algorithms by exploiting the flexibility of the MDP framework for generic decision-making and discovery. Nevertheless, when interpreted in a broader sense, neural algorithmic reasoning argues for building architectures that are informed by, and are compatible with, an algorithmic prior [367, 58]. Given that interest in this area has been shifting towards combinatorial optimisation problems [58], we expect there to be many cross-interactions and use of such techniques in the future.

1.4 Thesis Outline and Contributions

The structure and key contributions of this thesis are summarised in Figure 1.1. Towards the previously enumerated research questions, it makes contributions along the axes of *problem formulations*, *solution methods for MDPs*, and *learning representations*.

We begin in Chapter 2 by giving an overview of the literature relevant to this thesis. We discuss formalisations of networks and processes that take place over them, and review prior methods that have been proposed for their optimisation. We then cover the essential concepts behind graph representation learning, RL, and planning – techniques that underpin the approaches that we propose. Finally, we review related work in ML for the optimisation of graph processes.

Chapter 3 addresses the problem of constructing or modifying a graph so as to optimise a given objective function through the addition of edges. It proposes a Markov Decision Process formulation and an algorithm for Building Robust Networks with DQN (**RNet-DQN**) that learns how to optimise the structure of a graph. As objective functions, we consider the resilience of the network to random failures and targeted attacks. Evaluations on synthetic and real-world power grid and road networks show that the approach can outperform prior methods based on spectral or local properties, while exhibiting strong generalisation to larger graphs than those used for training in some cases. This chapter addresses ***RQ1***, ***RQ2***, and ***RQ3***.

In Chapter 4, we take further steps towards scaling the decision-making process approach for graph construction to larger, real-world networks. If we are interested in optimising a particular graph, such as a given infrastructure network, one can sidestep the computational cost of training an RL policy, and instead use a decision-time planning method that leverages knowledge of the transition and reward functions. In our pursuit of a more realistic formulation of the problem, we consider networks in which nodes are embedded in space, an aspect that governs the density and realisability of new connections. We propose a variant of the Upper Confidence Bound for Trees (UCT) planning algorithm [213] that we call Spatial Graph UCT (**SG-UCT**). We show that SG-UCT obtains excellent performance in optimising the resilience and efficiency of spatial networks used for internet communication and urban transportation. This chapter addresses ***RQ1***, ***RQ2***, and ***RQ3***.

Chapter 5 considers the problem of finding a Maximal Independent Set of nodes on a graph that maximises the value of a given objective function. We propose an MDP formulation of this task, and use the UCT algorithm to construct an mIS incrementally, outperforming prior methods. Furthermore, we devise a technique to carry out Imitation Learning of demonstrations of the search algorithm, which is shown to perform similarly in terms of its ability to optimise the objective while being orders of magnitude faster to evaluate after having been trained. The technique, which we term Graph Imitation Learning (**GIL**), may be used more widely for other graph combinatorial optimisation problems. We motivate the study of this problem through the correspondence between mISs and equilibria of the networked best-shot Public Goods Game [46], a social dilemma scenario that takes place on a graph and

captures the tensions between selfish actions and the collective good. This chapter addresses ***RQ4*** and ***RQ5***.

Then, in Chapter 6, we consider the problem of data-driven routing of flows on graphs between many sources and sinks, a problem that finds applications in logistics and computer networks. Despite the fact that GNNs are enjoying substantial interest for this problem, their benefits with respect to standard architectures are not well understood or validated. In this chapter, we argue that the "global" message functions used in many GNNs are not suitable for this type of problem, since they constrain the routing unnecessarily. We propose a GNN model with per-edge message functions, called Per-Edge Weights (**PEW**), which we show yields substantial gains in accuracy for predicting link utilisations in a computer network scenario that uses two routing schemes and 17 Internet Service Provider topologies. Furthermore, we examine the relationship between the performance of the considered predictive models and the topological structure of the underlying graphs, showing that the relative advantage of the proposed GNN increases under variations in topology and for highly heterogeneous networks. This chapter addresses ***RQ6***.

Lastly, we summarise and conclude in Chapter 7, focussing on lessons learned, applications of the proposed techniques, as well as directions for future work.

1.5 List of Publications

The preparation of this thesis has led to the following publications. The first author performed the algorithm and study designs, wrote the implementations, analysed the results, prepared the figures, and wrote the initial drafts of the manuscripts. Mirco Musolesi and Stephen Hailes had important advisory roles and contributed to refining the manuscripts. Accompanying code for the published papers can be found at `https://github.com/VictorDarvariu/`.

- Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Goal-directed graph construction using reinforcement learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2254):20210168, 2021.

- Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Planning spatial networks with Monte Carlo tree search. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 479(2269):20220383, 2023.

- Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Solving Graph-based Public Goods Games with Tree Search and Imitation Learning. In *Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems* (*NeurIPS*), 2021.
- Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Graph Neural Modeling of Network Flows. *arXiv preprint arXiv:2209.05208*, 2022.

The author has also contributed to the following works at the intersection of ML and combinatorial optimisation on graphs, which are relevant to the contents of the present thesis.

- Christoffel Doorman, Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Dynamic Network Reconfiguration for Entropy Maximization using Deep Reinforcement Learning. In *Proceedings of the First Learning on Graphs* (*LoG*) *Conference*, 2022.
- Syu-Ning Johnn, Victor-Alexandru Darvariu, Julia Handl, and Joerg Kalcsics. Graph Reinforcement Learning for Operator Selection in the ALNS Metaheuristic. In *Proceedings of the International Conference in Optimization and Learning* (*OLA2023*), 2023.

Chapter 2

Background and Related Work

This chapter offers an overview of the concepts and key works related to our topic of study. We begin by discussing fundamental notions about graphs and the variety of generative models that have been proposed to capture their characteristics. Subsequently, we summarise the relevant literature that seeks to quantify their properties and processes taking place over them, as well as prior works that aim to optimise the outcomes of such graph processes using standard methods that are not based on ML.

The later sections cover the necessary background for understanding how ML techniques may be used to approach the optimisation of graph processes. We give a brief overview of Artificial Neural Networks and how the discrete structure of graphs can be represented for ML tasks. We then cover the RL and planning paradigms in depth, both of which are central to this thesis. Finally, with the prerequisites having been met, we review the existing work on ML for optimising the outcome of processes taking place on graphs. We treat both work that tackles well-known NP-hard problems as well as other, less-studied graph processes.

2.1 Graph Fundamentals and Properties

Graphs, also called *networks*, are the underlying mathematical objects that are the focus of the present thesis. We denote a graph G as the tuple (V, E), where V is a set of *nodes* or *vertices* that are used to represent the entities that are part of the system, and E is a set of *edges* that represent connections and relationships between the entities. We indicate an element of the set V with v or v_i and an element of E

with e or $e_{i,j}$, with the latter indicating the edge between the nodes v_i and v_j.

A common mathematical structure used for representing graphs in computer programs is the *adjacency matrix* $\mathbf{A}$, in which the entry at row i and column j is equal to 1 if an edge exists between nodes v_i and v_j, and 0 otherwise. Since many networks are sparse in terms of the number of connections per node, in practical implementations an *edge list* representation, which only records existing edges, may be used instead. If $\mathbf{A}$ is symmetric, which implies reciprocal bidirectional connectivity, the graph is known as *undirected*. Otherwise, it is called *directed*.

Nodes and edges may optionally have attribute vectors associated with them, which we denote as $\mathbf{x}_v$ and $\mathbf{x}_e$ respectively. These can capture various aspects about the problem of interest depending on the application domain, and may be either static or dynamic. Some examples include geographical coordinates in a space, the on-off status of a node, and the capacity of an edge for the transmission of information or a physical quantity. Equipped with such attributes, graphs become a powerful mathematical tool for studying a variety of systems.

One may be interested in the relevance of each node in the overall connected structure, which is a *local* property. The *degree centrality* [263, Chapter 7.1.1] is a simple way of quantifying this, and is proportional to the number of links (degree) of each node. *Eigenvector centrality* [263, Chapter 7.1.2] takes this notion further, assigning higher values to nodes connected to nodes that themselves are densely connected. *Betweenness centrality* [263, Chapter 7.1.7] associates high values with nodes that are present in many of the shortest paths between other vertices. It is difficult to overstate the relevance of these measures: *PageRank* [48], related to eigenvector centrality, is a metric that was at the basis of the first implementation of one of the most famous search engines.

The empirical study of real-world networks has also unravelled various recurring *global* characteristics that can be used to describe connected systems. A notable example is the *average path length* [263, Chapter 10.2], i.e., the mean distance between vertex pairs along the shortest paths in a network. It has been used to explain the small-world phenomenon observed in Stanley Milgram's well-known "six degrees of separation" experiment [336]. Other important quantities include the *clustering coefficient* [263, Chapter 10.6], which measures the number of connected triples, and

the level of homophily of connections between vertices that are similar (sometimes called *assortativity* [263, Chapter 10.7]).

2.2 Classic Graph Generative Models

Graph *generative models* attempt to replicate certain local and global characteristics that were observed in real-world systems, or are otherwise mathematically interesting. Such generative models are able to produce instances of graphs that, while not having identical connectivity, share common statistical properties.

For example, Erdős and Rényi [114] studied the properties of graphs randomly drawn from the distribution of all graphs with a certain number of nodes and edges. Barabási and Albert [19] introduced a graph generation model based on preferential attachment that produces networks that mimic the scale-free degree distributions observed in real-world networks such as the World Wide Web. In another seminal work, Watts and Strogatz [354] proposed the "small-world" model based on the random rewiring of regular networks, which produces networks with characteristically small diameters. The authors showed that many real-world networks, such as the connectome of the nematode *C. elegans*, have this property. Stochastic Block Models (SBMs) [170], which have a long tradition in the social sciences, posit that connections in a network structure are governed by the presence of communities. Also called blocks, they provide a compact description of the network in terms of the intra- and inter-community connection probabilities.

Another family of graph generative models considers nodes that are placed in a metric space. In random geometric graphs [277], nodes are positioned uniformly randomly on the unit square, and connections occur for each pair of nodes with distance below a fixed threshold. Soft random geometric graphs [278] introduced probabilistic connections while maintaining the fixed threshold. The models proposed by Waxman [355] and Kaiser and Hilgetag [194] added a further degree of realism in relation to physical communication networks: namely, considering the cost of establishing a link as inversely proportional to the distance between nodes [23]. Planarity (i.e., the fact that edges intersect only at nodes) is another property of interest. The authors of [22] proposed a growth model for planar urban street networks which, in the absence of a central network designer, creates new connections that observe a local maximisation principle. Despite its simplicity, it is able to reproduce

patterns resembling existing cities, and it is closely related to a mechanism that yields leaf venation patterns similar to those observed in nature [299].

The generative models listed above are precisely defined in mathematical terms, an aspect that has permitted the analytical study of their properties. Other families of approaches instead consider *a posteriori* fitting of a model in a data-driven way based on some existing observations. Bayesian SBMs [276] are a family of approaches for nonparametric Bayesian inference of network structure, with extensions of this model having been proposed for dealing with hierarchical structure [274] or uncertainty around the existence of links [275]. Deep graph generative models [209, 233, 373, 235] have the same goal, but instead use a form of deep neural network trained with gradient descent. We will discuss these in detail in Section 2.6.2, once the necessary technical background has been introduced.

Let us make a brief intermission in order to mention the relationship between graph generative models and the present thesis. Graph generative models are a means of generating topologies that obey a certain prescribed set of rules or match empirical observations. In contrast, the present thesis seeks to allow a decision-maker to intervene in a network such that an objective in which they are interested can be maximised. Graphs produced by generative models serve as "starting points" or "underlying structure" to which the proposed algorithms are applied. They are also practically useful since they can be used to generate synthetic network topologies with controlled characteristics, enabling statistically robust evaluations.

2.3 Graph Processes

Beyond the statistical quantities mentioned in Section 2.1, graphs can also serve as a backdrop to simulate processes of interest. This is realised via the study of interactions that are dynamic in nature and take place on the topology, as governed by a set of mathematical rules [20]. Quantities related to these processes will serve as the optimisation objective in the present thesis.

In the following subsections, we carry out an in-depth treatment of each such process that is relevant to this thesis. Namely, we review work in network robustness, efficiency, flows, and games on networks. Furthermore, we cover recent works that seek to optimise the outcome of these processes using traditional (i.e., not based on ML) methods.

We note that this selection is not exhaustive. Some important types of processes that this thesis does not address are, among others: epidemic models such as Susceptible, Infected, Recovered (SIR) [202], synchronisation phenomena [20, Chapter 7], and opinion dynamics [20, Chapter 10]. However, at a high level, the proposed methods may also be adapted for the purpose of optimising other graph processes, as we will discuss in Section 7.4.

2.3.1 Robustness

Overview of Robustness

A process taking place on graphs that has attracted significant interest from the network science community and practitioners is *robustness* [262] (sometimes also called *resilience*), which is typically defined as the capacity of a graph to withstand random failures, targeted attacks on key nodes, or some combination thereof. This is a desirable characteristic in general, since it implies that a system can continue to function in the face of adversity; the most salient example where this property is essential is that of infrastructure networks. Previous works have studied the robustness of communication networks such as the Internet [77], energy distribution systems [61], and urban transportation networks [104, 359], just to name a few.

The existing literature proposes various definitions of robustness. The resilience of a graph to random errors and targeted attacks was first discussed by Albert et al. [8], who examined the average shortest path distance as a function of the number of removed nodes. Performing an analysis of two scale-free communication networks, they found that this type of network has good robustness to random failure but is vulnerable to targeted attacks. Other works consider a network to be robust if a significant fraction (called *Critical Fraction*) of nodes have to be removed before it breaks into more than one connected component [76] or the size of its Largest Connected Component diminishes [35]. A more recent measure of robustness proposed by Schneider et al. [306] is based on averaging the proportion of nodes in the Largest Connected Component as the percentage of attacked nodes increases; this can smoothly capture the functioning of the network before complete collapse.

In the case of targeted removal of nodes, there exists some variation in the possible strategies for attacking the network. An extensive investigation by Holme et al. [172] analysed the robustness of several real-world networks as well as some

generated by means of synthetic models. The authors investigated different attack strategies based on degree and betweenness centrality, finding that recomputing the centralities *after* the removal of nodes can yield more efficient attack strategies. A more recent investigation considered a wider scope of definitions of centrality than degree or betweenness [184].

Another relevant line of work on the study of resilience focusses on *percolation phenomena* taking place on grids [321] and graphs [56]. Using this framework, one can study the number of connected components as the fraction of nodes that are present in the network grows. For high fractions a single giant component typically occurs, while for small values the system consists of many tiny components. The value at which the switch between these two regimes happens is called the *percolation threshold*, and is a possible quantification of the failure rate that a network can withstand. The related cascading failure models [353] are more appropriate for networks that carry loads, such as power grids. Nodes are equipped with capacities, and upon node failure the load is distributed to the neighbours. Exceeding the capacity may trigger further failures, leading to avalanche-like phenomena [17].

Beyond the empirical analysis of real-world networks, various analytical results have also been obtained that describe the percolation thresholds of network models under random removal of nodes [76] and targeted attacks [77]. Optimal configurations under the joint objective of resilience to both attack strategies has also been found – the optimal network has a bi-modal or tri-modal degree distribution [340, 329]. Relationships between robustness and other graph properties have also been discovered: there is a positive association between the assortativity of the degree distributions of graphs and their robustness to vertex removal [260, 261]. Another property of interest is the *spectral gap*, defined as the difference (or ratio) between the two largest eigenvalues. Networks with high spectral gap have been proven to be highly connected, which also implies good robustness [174].

There exists evidence to suggest that the topological robustness of infrastructure systems is correlated to operational robustness [319]. More broadly, the resilience of systems is highly important in structural engineering and risk management [73, 132].

Optimising Robustness

Building a robust network from scratch is often impractical, since networks are generally designed with a specific purpose in mind. For this reason, prior works have addressed the problem of modifying *existing* networks in order to improve their robustness. Beygelzimer et al. [35] approached this problem by considering edge addition or rewiring, based on random or preferential (with respect to the degree of a node) modifications. In [306], the authors proposed a "greedy" modification scheme based on random edge selection and swapping if the resilience metric improves. For the problem of *de novo* graph generation, Wu and Holme [366] have proposed an algorithm that produces graphs with similarly high values of robustness at a much lower computational cost.

Schneider et al. [307] discussed an approach for improving the robustness of coupled networks. Rather than rewiring an existing network, the authors proposed making certain nodes "autonomous", which are selected based on degree and betweenness centrality. They found that this method can increase robustness significantly. Wang et al. [350] considered both preferential (based on node degree) and greedy (with respect to effective graph resistance [113]) modification schemes, finding that greedy strategies are generally more effective but also more expensive to compute. Related problems have also attracted the attention of the operations research community. Work on the Pre-disaster Transportation Network Preparation problem [273, 364] has treated the scenario of increasing the robustness of a highway network to cope with random failures of links.

2.3.2 Efficiency

Overview of Efficiency

Consider a network process that consists of nodes positioned in a physical space exchanging information, as is the case with transportation networks. Efficiency is a metric quantifying the distance that information has to travel along the shortest path defined by the links of the network in relation to the straight line distances between pairs of points. It is hypothesised to be an underlying principle for the organisation of networks [223, 224], including neural and social networks.

It is a more suitable metric for measuring the exchange of information than the inverse average path length between pairs of nodes. In the extreme case where the

network is disconnected (and thus some paths lengths are infinite), this metric does not go to infinity. More generally, this metric is better suited for systems in which information is exchanged in a parallel, rather than sequential, way [223]. A recent paper [33] extends the formalisation of efficiency to weighted networks, in which the links are equipped with capacities.

Optimising Efficiency

In comparison with the literature on robustness optimisation, the body of work on efficiency optimisation is limited. In [105], the authors considered a problem with node upgrade actions and the goal of reducing pairwise shortest path distances. They proposed two greedy algorithms that are shown to perform well in practice when compared to an exact algorithm.

2.3.3 Network Flows

Overview of Network Flows

The efficiency metric described above makes two implicit assumptions: that traffic between pairs of points is uniformly distributed, and that it travels in its entirety along the single shortest path. These assumptions are not suitable for communication networks such as the Internet, for which traffic volumes are highly heterogeneous and dynamic [13, 121], and different routing schemes that can split traffic among multiple paths to the destination are instead desirable in order to achieve a better utilisation of the network.

The Multi-Commodity Network Flow (MCNF) family of problems is a more general process of entities in a network exchanging information. It requires, in addition to the network topology, specifying a demand matrix $\mathbf{D}$ that captures the amount of traffic to be routed between each pair of vertices. The typical optimisation objective for a routing scheme is to minimise the ratio of utilisations versus capacity of the network links, possibly also sharply penalising high levels of congestion.

MCNF formulations are ubiquitous in Internet traffic engineering works [124, 195, 154] that serve as our motivating application, but also find wider uses, for example in routing goods in a logistics network or cars in a rail network [6, Chapter 17]. Monitoring the levels of congestion allows Internet Service Providers, for example, to gauge when further infrastructure investments to increase capacity are needed [151].

Optimising the Routing of Network Flows

MCNF solutions can be computed optimally in polynomial time given a demand matrix using Linear Programming (LP) techniques [331]. However, there are two debatable assumptions behind these approaches that are worth discussing, which in practice means that such problems are far from solved.

Firstly, they assume that the matrix of demands is fixed. This is an assumption that is not realistic in networks that show clear variations in traffic patterns, such as those between morning and evening [124]. Secondly, in such solutions, the decision variables specify the quantity of flow routed along each arc for each source-destination pair. This introduces complexity at the implementation level because it necessitates the configuration of source-destination routes [69]. Software-Defined Networking (SDN) technology [245] introduces the concept of programmable switches, which gives a centralised controller a global view and decision-making capabilities over the routing decisions; despite many successful solutions such as B4 [187], which is used to route traffic between Google's datacenters, transitioning to this model remains challenging in general due to interoperability requirements [69].

Other routing protocols are destination-based, a design that greatly simplifies implementation at the infrastructure level. Open Shortest Path First (OSPF) [257] is perhaps the most widespread among such protocols. In this specification, nodes maintain a database of the topology of the network and derive routes by calculating shortest paths to destinations in this topology as defined over weighted links. The Equal-Cost Multi Path (ECMP) extension [177] specifies that flows are split equally among shortest paths if multiple exist. Extensive prior works have studied the task of traffic engineering (TE) by setting weights for the OSPF-ECMP model using metaheuristics such as local search [126], memetic [54], and genetic [115] algorithms. In fact, this problem has been shown to be NP-hard [124] and, furthermore, it is impossible to approximate the optimal configuration within any constant ratio [68].

2.3.4 Network Games

Overview of Network Games

Let us move away from communication networks and consider a decentralised game-theoretic [231] scenario in which a social network connects self-interested agents. The actors have the liberty to take individual actions and derive utility based on their

neighbours' actions as well as their own. Such scenarios are referred to as *network games* [185] and, in case the set of possible actions is binary and utilities are defined in terms of graph neighbours, as *graphical games* [198].

An intuitive example is the network majority game, which bears resemblance to the Voter model [171]. In this game, each player in a network structure decides on an action of either 0 or 1, and receives positive utility if their chosen action agrees with the majority of the actions taken by neighbours, and negative utility otherwise [185].

In the majority game, there clearly are many possible combinations of actions taken by the players, which are called *action profiles*. How might we quantify whether an action profile is "better" than another? In such settings, this is typically called a *solution concept* [231]. One classic solution concept is the *Nash Equilibrium* (*NE*), an outcome in which no player would gain higher utility by unilaterally changing their action, given the actions of all other players. The *Pure Strategy Nash Equilibrium* (*PSNE*) is an equilibrium involving *pure* (i.e., non-probabilistic) strategies.

For the network majority game, finding a NE is fairly straightforward. This is due to the structure of the set of equilibria, which form an ordered lattice [185]. To reach an equilibrium, one can follow *Best-Response* dynamics, in which players are repeatedly allowed to adjust their action in *response* to that of a neighbour, and pick the *best* (i.e., highest utility) choice. However, this is not the case in general in game theory. A large spectrum of techniques exist for 2-player and n-player games that find a sample equilibrium or enumerate all equilibria, with the task being computationally challenging or intractable in many cases [244, 98]. In graphical games, more efficient algorithms can be derived for restricted cases such as complete graphs or trees [198, 375].

The broader literature on network games focusses on several fundamental questions regarding the behaviour of agents that are connected by a network structure [185]: examples include proving the existence of and characterising profiles of behaviour in equilibria, reasoning in the presence of partial or probabilistic information [131], and examining the effect of new links in the graph [46].

Public Goods Games (*PGGs*) are used to model scenarios in which individuals can choose to invest in a costly good that can be used, typically free of charge, by other members of a society [226]. It is a type of social dilemma [215], since it may

not be justified to make such a contribution from a purely selfish point of view, and yet the entire population would be better off if the pool of public resources is large. Such dynamics are observed in a variety of scenarios of societal importance such as vaccination programs [127], investing in research and development [186], or meeting climate change targets [201, 249]. Since network connections are known to shape decision-making in a variety of systems and at several scales [147], works in the literature have pursued finding NE in *networked* PGGs.

Optimising NE in Networked PGGs

Recent studies in this area treated the properties of binary networked PGGs [375] as well as designing strategies for inducing equilibria in such games by manipulating the graph structure itself [200].

For the more specific best-shot networked PGGs, in which utilities are a maximum of neighbour quantities, Best-Response dynamics have been proven to converge to a PSNE [230]. Two methods have been proposed for finding PSNEs in such games. Dall'Asta et al. [90] proposed a method based on simulated annealing for finding equilibria of maximum social welfare, which was proven to converge to the globally optimal solution given infinite time. Levit et al. [230] introduced a decentralised heuristic algorithm based on *side payments*, which can be used by agents that are unsatisfied with their outcome to convince their neighbours to switch their action. The configurations that such heuristics arrive at are nevertheless inferior to optimal solutions found by Exhaustive Search. Additionally, these methods cannot optimise for objectives other than social welfare.

2.4 Artificial Neural Networks on Graphs

Given that we have covered the *what* of the network processes we intend to optimise in the first 3 sections of this chapter, let us now move on to addressing the *how* – namely, the relevant ML tools. In this section, we discuss how one can represent graphs and tame their discrete structure for ML tasks. We begin with a broad overview of graph representation learning and traditional approaches. We then cover GNNs and related "deep" approaches for embedding graphs.

2.4.1 Artificial Neural Networks

The early history of Artificial Neural Networks (ANNs) begun in the 1940s and substantially overlaps with work on the biologically-inspired *cybernetics* [358, 146]. One of the earliest models of the means by which the brain processes information is the McCulloch-Pitts Neuron [243], which could distinguish between two different categories of input, and required the manual setting of model parameters. The Perceptron, introduced by Rosenblatt [292], was the first algorithm that allowed the automatic adjustment of parameters from examples. However, the linear nature of the Perceptron meant that it could not distinguish functions that are not linearly separable. The XOR is a notorious example of such a function, and its presence in the Perceptrons book of Minsky and Papert [250] caused a drop in interest and research activity on ANNs.

The second wave of interest in ANNs originated in the cognitive science community in the 1980s, and came to be known as *connectionism* [241, 242]. This literature made two key contributions. Firstly, it introduced the idea of distributed representations, which stipulates that each feature should be involved in the intermediate representation of multiple inputs instead of being pre-programmed for a particular input. Secondly, it proposed the backpropagation algorithm [298] that enables an efficient computation of the gradients of model weights with respect to the loss function, and is still in wide use today.

In its modern sense, an ANN takes a vector of inputs $\mathbf{x}$, performs information processing, and produces a vector of outputs $\hat{\mathbf{y}}$. A possible instantiation is

$$\hat{\mathbf{y}} = \text{ReLU}\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right) \tag{2.1}$$

where $\mathbf{W}$ is a matrix of weights, $\mathbf{b}$ is a vector of biases, and the Rectified Linear Unit $\text{ReLU}(x) = \max(0, x)$ is an *activation function* that is applied element-wise. The weights and biases are the parameters of the ANN and control the information processing. One can repeat this recipe and stack several *layers* of the ANN; for example, a two-layer ANN may be defined as:

$$\hat{\mathbf{y}} = \text{ReLU}\left(\mathbf{W}_2\ \text{ReLU}\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) \tag{2.2}$$

where, analogously, $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices, and $\mathbf{b}_1$, $\mathbf{b}_2$ are bias vectors. An ANN with two or more such layers is commonly called Multi-Layer Perceptron (MLP). It forms a common building block for many neural network architectures.

How does one arrive at the "right" values for the weights and biases? Let us refer to them collectively as the *model parameters* Θ. One needs to define a *loss function* $\mathcal{L}(\Theta)$ which specifies the "goodness" of a particular choice of parameters, and is typically the minimisation objective. Concretely, in the case of univariate regression, one seeks to predict scalar outputs y, and is provided with a dataset of inputs and ground truth real-values $\mathcal{D} = \cup_k \{\mathbf{x}^{(k)}, y^{(k)}\}$. Given an input $\mathbf{x}^{(k)}$, the model produces an approximate value $\hat{y}^{(k)}$. The Mean Squared Error (MSE) loss is defined as:

$$\mathcal{L}_{\text{MSE}}(\Theta) = \frac{\sum_k (y^{(k)} - \hat{y}^{(k)})^2}{|\mathcal{D}|} \tag{2.3}$$

It is common to use *mini-batch Stochastic Gradient Descent* (*SGD*) to learn the parameters Θ in a data-driven fashion. One can start with a random guess of the values Θ and adjust them in an iterative process that is driven by the loss function. A mini-batch of datapoints is sampled uniformly at random, and the gradient $\nabla_\Theta \mathcal{L}(\Theta)$ of the loss function with respect to the parameters is computed. One then takes a step in the direction of the gradient according to a *learning rate* α that dictates the magnitude of the step: $\Theta' \leftarrow \Theta - \alpha \nabla_\Theta \mathcal{L}(\Theta)$.

The choice of α is dictated by the optimisation algorithm, and it may be fixed or adaptive. A popular choice in this literature is the Adam optimisation algorithm [207], which keeps track of the mean and variance of the gradients to adjust the learning rate. The process continues until certain stopping criteria are met, such as executing a certain number of mini-batches.

When a model consists of many layers, it is common to refer to such approaches as *deep learning*. Stacking many layers leads to a form of hierarchical feature processing, with the further layers receiving higher-level features as input [146]. Interest

in deep learning has surged in the 2010s as a result of an increase in computational power of modern hardware combined with increased levels of training data availability, enabling deep neural networks to attain excellent empirical results in a variety of domains. They have been able to overtake traditional approaches in a variety of fields including computer vision [217], speech and signal processing [148], and natural language processing [15] while requiring significantly less feature engineering.

The *universal function approximation* theorem [178, 82] establishes that MLPs of arbitrary width can approximate any bounded continuous real function. This serves as a theoretical foundation of these remarkable empirical results. However, it is not prescriptive about the required width, nor does it account for the tasks themselves, some of which we might expect to be more easily approximated than others.

MLPs, while being generic, do not take advantage of the structure of the problems under consideration, and may require a prohibitively large amount of data or number of parameters in order to obtain good predictive performance. Variations in the neural network architecture can encode structural inductive biases about the problem, such that a model may be arrived at with tractable cost. A now canonical example is that of Convolutional Neural Networks (CNN) [130, 225] for computer vision. Data in this domain is highly local, grid-like, and compositional at multiple scales. These characteristics are encoded by design in the architecture, avoiding the large costs needed for discovering them *tabula rasa* with MLPs (e.g., as was achieved in a recent work [334]).

2.4.2 Graph Representation Learning

As mentioned previously, graphs are the structure of choice for representing information in many other fields and are able to capture interactions and similarities. The success of neural networks, however, did not immediately transfer to the domain of graphs as there is no obvious equivalent for this class of architectures: graphs do not necessarily present the same type of local statistical regularities [50].

Suppose we wish to represent a graph as a vectorial input $\mathbf{x}$ that can be fed to a ML model, such as an MLP. An obvious choice is to take the adjacency matrix $\mathbf{A}$ and apply vectorization in order to transform it to a column vector. In doing so, however, two challenges become apparent. One is the fact that by relabelling the nodes in the graph according to a permutation, the input vector is modified substantially,

and may result in a very different output when fed to a model, even though the structure has remained identical. Furthermore, if a new node joins the network, our previous model is no longer applicable by default since the shape of the input vector has changed. How might we address this, and design a better graph representation?

Such questions are studied in the field of graph representation learning [159]. Broadly speaking, the field is concerned with learning a mapping that translates the discrete structure of graphs into vectorial representations with which downstream ML approaches can work effectively. Work in this area is focussed on deriving vectorial embeddings for a node, a subgraph, or an entire graph. A distinction can be drawn between *shallow* and *deep* embedding methods [159]. The former category consists of manually-designed approaches such as those using local node statistics, characteristic graph matrices, or graph kernels. In the latter category, methods typically use a deep neural network trained with gradient descent, and learn the representation in a data-driven fashion.

The literature on shallow embeddings is extensive. Prior work has considered factorisations of characteristic graph matrices [25, 268]. Another class of approaches constructs embeddings based on random walks: nodes will have similar embeddings if they occur along similar random paths in the graphs. Methods such as DeepWalk and node2vec [279, 149] fall into this category. Other approaches such as struc2vec [288] and GraphWave [106] assign similar embeddings to nodes that fulfil a similar structural role within the graph (e.g., hub), irrespective of their proximity.

2.4.3 Deep Graph Embedding Methods

Broadly speaking, deep embedding methods rely on the idea of neighbourhood aggregation: the representation of a node is determined in several rounds of aggregating the embeddings and features of its neighbours, to which a non-linear activation function is applied.

The learning architectures are usually parameter-sharing: the manner of performing the aggregation and the corresponding weights are the same for the entire network. Another important aspect for deep embedding methods is that they can incorporate task-specific supervision: the loss corresponding to the decoder can be swapped for e.g. cross-entropy loss in the case of classification tasks. A final distinction in this family of approaches is the way subgraph embeddings are con-

structed: variants include performing a sum or mean over node embeddings in a subgraph [112, 85], introducing a dummy node [232], using layers that perform clustering [100], or learning the hierarchical structure end-to-end [370].

Several works have proposed increasingly feasible versions of convolutional filters on graphs, based primarily on spectral properties [52, 165, 100, 210] or their approximations. An alternative line of work is based on message passing on graphs [320, 303] as a means of deriving vectorial embeddings. Both Message Passing Neural Networks [144] and Graph Networks [24] are attempts to unify related methods in this space, abstracting the commonalities of existing approaches with a set of primitive functions. The term "Graph Neural Network" is used in the literature, rather loosely, as an umbrella term to mean a deep embedding method.

Let us now take a closer look at some GNN terminology and variants that are relevant to the present thesis. Recall that we are given a graph $G = (V, E)$ in which nodes v_i are equipped with feature vectors $\mathbf{x}_{v_i}$ and, optionally, with edge features $\mathbf{x}_{e_{i,j}}$. The goal is to derive an embedding vector $\mathbf{h}_{v_i}$ for each node that captures the features as well as the structure of interactions on the graph. The computation of the embedding vectors happens in layers $l \in 1, 2, ..., L$, where L denotes the final layer. We use $\mathbf{h}_{v_i}^{(l)}$ to denote the embedding of node v_i in layer l. The notation $\mathbf{W}^{(l)}$, possibly indexed by a subscript, denotes a weight matrix that represents a block of learnable parameters in layer l of the GNN model. Unless otherwise specified, the embeddings are initialised with the node features, i.e., $\mathbf{h}_{v_i}^{(0)} = \mathbf{x}_{v_i}, \forall v_i \in V$.

Message Passing Neural Network

The Message Passing Neural Network (MPNN) [144] is a framework that abstracts several graph learning architectures, and serves as a useful conceptual model for deep embedding methods in general. It is formed of layers that apply a *message function* $M^{(l)}$ and *vertex update function* $U^{(l)}$ to compute embeddings as follows:

$$\begin{aligned} \mathbf{m}_{v_i}^{(l+1)} &= \sum_{v_j \in \mathcal{N}(v_i)} M^{(l)} \left(\mathbf{h}_{v_i}^{(l)}, \mathbf{h}_{v_j}^{(l)}, \mathbf{x}_{e_{i,j}} \right) \\ \mathbf{h}_{v_i}^{(l+1)} &= U^{(l)} \left(\mathbf{h}_{v_i}^{(l)}, \mathbf{m}_{v_i}^{(l+1)} \right) \end{aligned} \tag{2.4}$$

where $\mathcal{N}(v_i)$ is the open neighbourhood of node v_i. Subsequently, a *readout function* $\mathcal{I}$ is applied to compute an embedding for the entire graph from the set of final

node embeddings: $\mathcal{I}(\{\mathbf{h}_{v_i}^{(L)}|v_i \in V\})$. The message and vertex update functions are learned and differentiable, e.g., some form of MLP. The readout function may either be learned or fixed *a priori* (e.g., summing the node embeddings). A desirable property for it is to be invariant to node permutations.

structure2vec

structure2vec (S2V) [85] is one of the earlier GNN variants, and it is inspired by probabilistic graphical models [214]. The core idea is to interpret each node in the graph as a latent variable in a graphical model, and to run inference procedures similar to mean field inference [347] and loopy belief propagation [272] to derive vectorial embeddings. Additionally, the approach replaces the "traditional" probabilistic operations (sum, product, and renormalisation) used in the inference procedures with nonlinear functions (namely, neural networks), yielding flexibility in the learned representation. It was shown to perform well for classification and regression in comparison to other graph kernels, as well as to be able to scale to medium-sized graphs representing chemical compounds and proteins.

One possible realisation of the mean field inference variant of S2V computes embeddings in each layer based on the following update rule:

$$\mathbf{h}_{v_i}^{(l+1)} = \text{ReLU}\left(\mathbf{W}_1\mathbf{x}_{v_i} + \mathbf{W}_2 \sum_{v_j \in \mathcal{N}(v_i)} \mathbf{h}_{v_j}^{(l)}\right) \tag{2.5}$$

where $\mathbf{W}_1, \mathbf{W}_2$ are weight matrices that parametrise the model. We note that there are two differences to the other architectures discussed in this section. Firstly, the weight matrices are not indexed by the layer superscript, since they are shared between all the layers. Additionally, the node features $\mathbf{x}_{v_i}$ appear in the message-passing step of every layer, rather than only being used to initialise the embeddings. A possible alternative is to use vectors of zeros for initialisation, i.e., $\mathbf{h}_{v_i}^{(0)} = \mathbf{0}\, \forall v_i \in V$.

Graph Convolutional Network

The Graph Convolutional Network (GCN) method [210] is substantially simpler in nature, relying merely on the multiplication of the node features with a weight matrix, together with a degree-based normalisation. It is motivated as a coarse, first-order, approximation of localised spectral filters on graphs [100]. Since it can be formulated as a series of matrix multiplications, it has been shown to scale well to

large graphs with millions of edges, while obtaining superior performance to other embedding methods at the time. It can be formulated as:

$$\mathbf{h}_{v_i}^{(l+1)} = \text{ReLU}\left(\mathbf{W}_1^{(l)} \sum_{v_j \in \mathcal{N}[v_i]} \frac{\mathbf{h}_{v_j}^{(l)}}{\sqrt{(1+\deg(v_i))(1+\deg(v_j))}}\right) \tag{2.6}$$

where $\deg(v_i)$ indicates the degree of node v_i, and $\mathcal{N}[v_i]$ is the closed neighbourhood of node v_i, which includes all its neighbours and v_i itself.

Graph Attention Network

Note how, in the GCN formula above, the summation implicitly performs a rigid weighting of the neighbouring nodes' features. The Graph Attention Network (GAT) model [343] proposes the use of attention mechanisms [15] as a way to perform flexible aggregation of neighbour features instead. Learnable aggregation coefficients enable an increase in model expressibility, which also translates to gains in predictive performance over the GCN for node classification.

Let $\zeta_{i,j}^{(l)}$ denote the attention coefficient that captures the importance of the features of node v_j to node v_i in layer l. It is computed as:

$$\zeta_{i,j}^{(l)} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{\theta}^T \left[\mathbf{W}_1^{(l)}\mathbf{h}_{v_i}^{(l)} \| \mathbf{W}_1^{(l)}\mathbf{h}_{v_j}^{(l)} \| \mathbf{W}_2^{(l)}\mathbf{x}_{e_{i,j}}\right]\right)\right)}{\sum_{v_k \in \mathcal{N}[v_i]} \exp\left(\text{LeakyReLU}\left(\boldsymbol{\theta}^T \left[\mathbf{W}_1^{(l)}\mathbf{h}_{v_i}^{(l)} \| \mathbf{W}_1^{(l)}\mathbf{h}_{v_k}^{(l)} \| \mathbf{W}_2^{(l)}\mathbf{x}_{e_{i,k}}\right]\right)\right)} \tag{2.7}$$

where $\exp(x) = e^x$ is the exponential function, $\boldsymbol{\theta}$ is a weight vector that parametrises the attention mechanism, and $[\cdot\|\cdot]$ denotes concatenation. The LeakyReLU(x) activation function, which outputs non-zero values for negative inputs according to a small slope α_{LR}, is equal to $\alpha_{\text{LR}}x$ if $x < 0$, and x otherwise. Given the attention coefficients, node embeddings are computed according the rule below.

$$\mathbf{h}_{v_i}^{(l+1)} = \sum_{v_j \in \mathcal{N}[v_i]} \zeta_{i,j}^{(l)} \mathbf{W}_1^{(l)} \mathbf{h}_{v_j}^{(l)} \tag{2.8}$$

Analogously, it is also possible to use multiple attention "heads", which can improve model performance in some settings [343].

Relational Graph Convolutional Network

The Relational Graph Convolutional Network (RGCN) model [305] is an extension of the GCN that is purpose-built for knowledge graphs. In such graphs, nodes represent entities, and edges are tagged with a relation type χ out of a set $\mathcal{X}$. The model proposes using different parametrisations for relations of different types as a means of capturing their diverse semantics. Concretely, this is realised by using a separate weight matrix $\mathbf{W}_{\chi}^{(l)}$ for each relation type χ in layer l. The model computes embeddings according to:

$$\mathbf{h}_{v_i}^{(l+1)} = \text{ReLU}\left(\sum_{\chi\in\mathcal{X}}\sum_{v_j\in\mathcal{N}^{\chi}(v_i)}\frac{1}{\mu_{i,\chi}}\mathbf{W}_{\chi}^{(l)}\mathbf{h}_{v_j}^{(l)} + \mathbf{W}_{1}^{(l)}\mathbf{h}_{v_i}^{(l)}\right) \tag{2.9}$$

where $\mathcal{N}^{\chi}(v_i)$ denotes the neighbourhood of v_i along the relation type χ, and $\mu_{i,\chi}$ is a problem-specific normalisation factor that can be fixed or learned.

RGCNs have been successfully applied to a variety of knowledge graph tasks, such as link prediction and entity classification. Despite this success, perhaps surprisingly, recent preliminary work shows that randomly trained relation weights may perform similarly well [101], highlighting the need for further principled investigations into this model class.

Relational Graph Attention Network

Relational Graph Attention Networks (RGATs) [55] follow the footprint of RGCNs and extend the GAT approach to the relational setting. As with the RGCN, one uses separate weight matrices $\mathbf{W}_{\chi}^{(l)}$ for each relation type. The fusion of the approaches presents several choices with respect to the attention mechanism used and the ways in which the attention coefficients should be aggregated when dealing with several relation types. The *additive self-attention, across-relation* variant of RGAT is defined as follows. To compute the coefficients for each relation, one first needs to compute intermediate representations $\mathbf{g}_{v_i,\chi}^{(l)} = \mathbf{W}_{\chi}^{(l)}\mathbf{h}_{v_i}$ by performing multiplication with a weight matrix. Subsequently, the "query" and "key" representations are defined as below, where $\mathbf{Q}_{\chi}^{(l)}$ and $\mathbf{K}_{\chi}^{(l)}$ represent per-relation query and key kernels respectively:

$$\mathbf{q}_{v_i,\chi}^{(l)} = \mathbf{g}_{v_i,\chi}^{(l)} \cdot \mathbf{Q}_{\chi}^{(l)} \text{ and } \mathbf{k}_{v_i,\chi}^{(l)} = \mathbf{g}_{v_i,\chi}^{(l)} \cdot \mathbf{K}_{\chi}^{(l)} \tag{2.10}$$

Then, the attention coefficients $\zeta_{i,j,\chi}^{(l)}$ are computed according to:

$$\zeta_{i,j,\chi}^{(l)} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{q}_{i,\chi}^{(l)} + \mathbf{k}_{j,\chi}^{(l)} + \mathbf{W}_1^{(l)}\mathbf{x}_{e_{i,j}}\right)\right)}{\sum_{\chi' \in \mathcal{X}} \sum_{v_k \in \mathcal{N}^{\chi'}[v_i]} \exp\left(\text{LeakyReLU}\left(\mathbf{q}_{i,\chi}^{(l)} + \mathbf{k}_{k,\chi}^{(l)} + \mathbf{W}_1^{(l)}\mathbf{x}_{e_{i,k}}\right)\right)} \tag{2.11}$$

Finally, the node embeddings are computed as:

$$\mathbf{h}_{v_i}^{(l+1)} = \text{ReLU}\left(\sum_{\chi \in \mathcal{X}} \sum_{v_j \in \mathcal{N}^{\chi}(v_i)} \zeta_{i,j,\chi}^{(l)} \mathbf{g}_{v_j,\chi}^{(l)}\right) \tag{2.12}$$

Despite the increased expressivity of RGAT, the authors of the original paper were not able to find a case in which it is guaranteed to perform better than the RGCN [55]. However, its ability to include edge features natively alongside different relation types is a good fit for the problem we consider in Chapter 6.

2.5 Decision-making Processes and Solution Methods

We now move on to discussing the other crucial set of methods used in this thesis: decision-making processes and approaches for solving them. We begin by defining the key elements of Markov Decision Processes. We also give a broad overview of solution methods for MDPs and discuss some conceptual "axes" along which they may be compared and contrasted. We then cover several relevant methods for constructing a policy, including those that perform policy iteration, learn a policy directly, or perform planning from a state of interest. Finally, we discuss other research directions in RL that are relevant to graph combinatorial optimisation.

In the following chapters, we make use of Q-learning [352], Behavioural Cloning [283], and several search techniques including a variant of Monte Carlo Tree Search [213]. However, we consider it important to compare and contrast the wide range of methods that have been proposed in the past for solving MDPs, since they rest on diverse assumptions and principles. Indeed, other algorithmic choices may be more suitable for optimising graph processes different to those presented here, or may be a better fit for real-world use cases with specific constraints on data collection and interaction with the environment.

2.5.1 Markov Decision Processes

RL refers to a class of methods for producing goal-driven behaviour. In broad terms, decision-makers called *agents* interact with an uncertain *environment*, receiving numerical *reward signals*; their objective is to adjust their behaviour in such a way as to maximise the sum of these signals. Modern RL bases its origins in optimal control and Dynamic Programming methods for solving such problems [34] and early work in trial-and-error learning in animals. It has important connections to conditioning in psychology as well as neuroscience – for example, a framework to explain the activity of dopamine neurons through temporal-difference learning has been developed [253]. Such motivating connections to learning in biological systems, as well as its applicability in a variety of decision-making scenarios, make RL an attractive way of representing the basic ingredients of the Artificial Intelligence problem.

One of the key building blocks for RL is the Markov Decision Process (MDP). An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ is a set of states in which the agent can find itself;
- $\mathcal{A}$ is the set of actions the agent can take, and $\mathcal{A}(s)$ denotes the actions that the agent can take in state s;
- P is the state transition function: $P(S_{t+1} = s' | S_t = s, A_t = a)$, which sets the probability of agents transitioning to state s' after taking action a in state s;
- R is a reward function, and denotes the expected reward when taking action a in state s: $R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$;
- $\gamma \in [0, 1]$ is a discount factor that controls the agent's preference for immediate versus delayed reward.

A *trajectory* $S_0, A_0, R_1, S_1, A_1, R_2, ...S_{T-1}, A_{T-1}, R_T$ is defined by the sequence of the agent's interactions with the environment until the terminal timestep T. The *return* $H_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ denotes the sum of (possibly discounted) rewards that are received from timestep t until termination. We also define a *policy* $\pi(a|s)$, a distribution of actions over states which fully defines the behaviour of the agent. Given a particular policy π, the *value function* $V_\pi(s)$ is defined as the expected return

when following the policy π in state s. Similarly, the *action-value function* $Q_\pi(s, a)$ is defined as the expected return when starting from s, taking action a, and subsequently following π.

There exists at least one policy π_*, called the *optimal policy*, which has an associated optimal action-value function Q_*, defined as $\max_\pi Q_\pi(s, a)$. The *Bellman optimality equation* $Q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma Q_*(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$ is satisfied by the optimal action-value functions. Solving this equation provides a possible route to finding an optimal policy and, hence, solving the RL problem.

2.5.2 Dimensions of RL Algorithms

How is a policy learned? There exists a spectrum of algorithms for this task. Before delving into the details of specific approaches, which we shall do in the following section, we begin by giving a high-level picture of the main axes that may be used to characterise these methods.

Model-based versus Model-free

One important distinction is that between *model-based* algorithms (which assume knowledge of the MDP) and *model-free* algorithms (which require only samples of agent-environment interactions). To be specific, the state space $\mathcal{S}$ and action space $\mathcal{A}$ are assumed to be known; a *model* $\mathcal{M} = (P, R)$ refers to knowing, or having some estimate of, the transition and reward functions P, R.

Model-based methods can incorporate knowledge about the world to greatly speed up learning. The model can either be given a priori or learned. In the former case, it typically takes the form of a set of mathematical descriptions that fully define P and R. In the latter case, learning P corresponds to a density estimation problem, while learning R is a Supervised Learning problem. Learning architectures for this purpose can range from probabilistic models such as Gaussian Processes [102] to deep neural networks [266].

Model-based RL is especially advantageous where real experience is expensive to generate, and executing poor policies may have a negative impact (e.g., in robotics). When equipped with a model of the world, the agent can *plan* its policy, either at the time actions need to be taken focussing on the current state (*decision-time planning*), or not focussed on any particular state (*background planning*). Model-free algorithms, on the other hand, can yield simpler learning architectures. This comes at the

expense of higher sample complexity: they typically take more interactions with the environment to train. The two categories can also be combined: it is possible to use a model to generate transitions, to which a model-free algorithm can be applied [150].

It is worth noting that, in the context of this thesis, we typically have access to analytical descriptions of the transition and reward functions, which are defined based on the graph process of interest.

Other Considerations

Approaches may also be divided into *on-policy* and *off-policy*. The distinction relies on the existence of two separate policies: the *behaviour* policy, which is used to interact with the environment, and the *target* policy, which is the policy that is being learned. For on-policy methods, the behaviour and target policy are identical, while they are different in off-policy algorithms. Off-policy methods are more flexible and include on-policy approaches as a special case. They can enable, for example, learning from policy data generated by a controller or a human.

The category of sample-based or Monte Carlo (MC) methods rely on *samples* of interactions with the environment, rather than complete knowledge of the MDP. They aim to solve the Bellman optimality equations using the returns of sampled trajectories, albeit approximately, which requires less computation than an exact method while still yielding competent policies. Temporal-Difference (TD) methods, in addition to being based on samples of experience, use "bootstrapping" of the value estimate based on previous estimates. This leads to estimates that are biased but have less variance. It possesses certain advantages such as naturally befitting an online scenario in which learning can occur during an episode without needing to wait until the end when the return is known; applicability in non-episodic tasks; as well as empirically better convergence properties [326].

2.5.3 Policy Iteration Methods

A common framework underlying many RL algorithms is that of Policy Iteration (PI). It consists of two phases that are applied alternatively, starting from a policy π. The first phase is called *policy evaluation* and aims to compute the value function by updating the value of each state iteratively. The second phase, *policy improvement*, refines the policy with respect to the value function, most commonly by acting greedily with respect to it. Under certain conditions, this scheme is proven to converge to the opti-

mal value function and optimal policy [326]. *Generalised* Policy Iteration (GPI) refers to schemes that combine *any* form of policy evaluation and policy improvement.

Let us look at some concrete examples of algorithms. Dynamic Programming (DP), which is model-based, applies the PI scheme as described above. It is one of the earliest solutions developed for MDPs [28]. However, since it involves updating the value of every state in the entire state space, it is computationally intensive.

The Q-learning [352] algorithm is an off-policy TD method that follows the GPI blueprint. Unlike DP, it only requires samples of interactions with the environment. It is proven to converge to the optimal value functions and policy in the tabular case with discrete actions, so long as, in all the states, all actions have a non-zero probability of being sampled [352]. The agent updates its estimates according to:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s',a') - Q(s,a)\big) \tag{2.13}$$

In the case of high-dimensional state and action spaces, a popular means of generalising across similar states and actions is to use a function approximator for estimating $Q(s,a)$. An early example of such a technique is the Neural Fitted Q-iteration (NFQ) [289], which uses a neural network. The DQN algorithm [251], which improved NFQ by use of an experience replay buffer and an iteratively updated target network for state-action value function estimation, has yielded state-of-the-art performance in a variety of domains ranging from general game-playing to continuous control [251, 236].

A variety of general and problem-specific improvements over the DQN have been proposed [166]. Prioritised Experience Replay weighs samples of experience proportionally to the magnitude of the encountered TD error [304], arguing that such samples are more important for the learning process. Double DQN uses two separate networks for action selection and Q-value estimation [341] to address the overestimation bias of standard Q-learning. Distributional Q-learning [27] models the distribution of returns, rather than only estimating the expected value.

The DQN has been extended to continuous actions via the DDPG algorithm [236], which features an additional function approximator to estimate the action which maximises the Q-value. TD3 [129] is an extension of DDPG that applies additional tricks that improve stability and performance over standard DDPG.

2.5.4 Learning a Policy Directly

An alternative approach to RL is to parametrise the policy $\pi(a|s)$ by some parameters Θ directly instead of attempting to learn the value function. In effect, the objective is to find parameters Θ^* which make the parametrised policy π_Θ produce the highest expected return over all possible trajectories τ that arise when following the policy:

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}}\, \mathbb{E}_{\tau\sim\pi_\Theta} \sum_t^T R(S_t, A_t) \tag{2.14}$$

If we define the quantity under the expectation as J_Θ, the goal is to adjust the parameters Θ in such a way that the value of J is maximised. We can perform gradient ascent to improve the policy in the direction of actions that yield high return, formalising the notion of trial and error. The gradient can be written as:

$$\nabla_\Theta J(\Theta) = \mathbb{E}_{\tau\sim\pi_\Theta}\left[\left(\sum_t^T \nabla_\Theta \log \pi_\Theta(A_t|S_t)\right)\left(\sum_t^T R(S_t, A_t)\right)\right] \tag{2.15}$$

Such approaches are called *policy gradient* algorithms, a well-known example of which is REINFORCE [360]. A number of improvements over this basic scheme exist: for example, in the second sum term one can subtract a baseline (e.g., the average observed reward) such that only the probabilities of actions that yield rewards *better than average* are increased. This estimate of the reward will still be noisy, however. An alternative is to fit a model, called *critic* to estimate the value function, which will yield lower variance [326, Chapters 13.5-6].

This class of algorithms is called Actor-Critic, and modern asynchronous variants have been proposed, such as A3C, which parallelises training with the effect of both speeding up and stabilising the training [252]. Soft Actor-Critic (SAC) introduces an additional term to maximise in addition to the expected reward: the entropy of a stochastic policy [156]. Other policy gradient variants concern the way the gradient step is performed; modern algorithms in this class include Trust Region Policy Optimisation (TRPO) [309] and Proximal Policy Optimisation (PPO) [310].

Despite the popularity of these algorithms, a surprising finding of recent work is that a carefully constructed random search [238] or using evolutionary strategies [302] are viable alternatives to model-free RL for navigating the policy space.

Another means of learning a policy directly is through *Imitation Learning* (IL). Instead of environment interactions or model knowledge, it relies on expert trajectories generated by a human or algorithm that performs very well on the task. The simplest form of IL is Behavioural Cloning (BC) [283, 16]. It can take the form of a classification problem, in which the goal is to learn a state to action mapping from the expert trajectories. Alternatively, in case trajectories include probabilities for all actions, one can train a probabilistic model that minimises the distance (e.g., the Kullback–Leibler divergence) between the expert and model probability distributions.

A possible downside of BC is that it violates the i.i.d. assumption and may suffer from compounding errors as soon as the policy makes a mistake and trajectories diverge from those seen during training. Follow-up works considered reducing the impact of this issue by enabling the agent to query the expert for more data [294] or using Generative Adversarial Networks [168]. The imitated policies can be used by themselves, or as an initialisation for RL [229, 372]. IL has found success for robotics tasks such as autonomous vehicle navigation [284, 75] and flying drones [313].

This concludes our discussion of the ways in which a policy may be learned from data. Before moving on to describing model-based approaches for search and decision-time planning, let us take the opportunity to mention the relationship between the RL algorithms discussed up to this point and the graph representation learning methods covered in Section 2.4.3. Namely, the set of model weights Θ of a deep graph embedding approach serve as an effective basis for parametrising a policy π_Θ or an approximation $\hat{Q}_\Theta$ of the true action-value function. For updating the parameters Θ using gradient descent, instead of using cross-entropy or Mean Squared Error losses with respect to the labels of "ground truth" examples as one would in a Supervised Learning setting, one computes losses with respect to learning targets that are defined by the RL algorithm.

2.5.5 Search and Decision-Time Planning Methods

Recall the fact that, in the case of model-based methods, we have access to the transition function P and reward function R. This means that an agent does not necessarily need to interact with the world and learn directly through experience. Instead, the agent may use the model in order to *plan* the best course of action, and subsequently execute the carefully thought-out plan in the environment. Furthermore, *decision-*

time planning concerns itself with constructing a plan starting from the current state that the agent finds itself in, rather than devise a policy for the entire state space.

To achieve this, the agent can perform rollouts using the model, and use a *search* technique to find the right course of action. Search has been one of the most widely utilised approaches for building intelligent agents since the dawn of AI [301, Chapters 3-4]. Methods range from relatively simple in-order traversal (e.g., Breadth-First Search and Depth-First Search) to variants that incorporate heuristics (e.g., A*). It has been applied beyond single-agent discrete domains to playing multi-player games [301, Chapter 5]. Such games have long been used as a *Drosophila* of Artificial Intelligence, with search playing an important part in surpassing human-level performance on many tasks [258, 57, 314].

Search methods construct a *tree* in which the nodes are states in the MDP. Children nodes correspond to the states obtained by applying a particular action to the state at the parent node, while leaf nodes correspond to terminal states, from which no further actions can be taken. The root of the search tree is the current state. The way in which this tree is expanded and navigated is dictated by the particulars of the search algorithm.

It is worth first discussing two of the simpler search techniques that are used in this thesis. *Exhaustive Search* (ES) refers to expanding all possible paths in the MDP from the current state and picking the best trajectory (i.e., the trajectory yielding the highest return). It is typically impractical due to the computational and memory requirements, but can serve as a useful benchmark on small problems. *Greedy Search* creates, at each step, a search tree of shallow depth rooted at the current state. It subsequently picks the best child node, and repeats the search with the child node at the root until a terminal state is reached. It is less resource-intensive than ES, but may also lead to short-sighted decisions. Despite this, it is commonly used in practice to good effect in a variety of problems [80].

In many applications, however, the branching factor b and depth d of the search tree make it impossible to explore all paths, or even perform greedy search. There exist proven ways of reducing this space, such as alpha-beta pruning. However, its worst-case performance is still $\mathcal{O}(b^d)$ [301, Chapter 5.3]. A different approach to breaking the curse of dimensionality is to use random (also called Monte Carlo)

rollouts: to estimate the goodness of a position, run random simulations from a tree node until reaching a terminal state [3, 332].

Monte Carlo Tree Search (MCTS) is a model-based planning technique that addresses the inability to explore all paths in large MDPs by constructing a policy *from the current state* [326, Chapter 8.11]. It relies on two core principles: firstly, that the value of a state can be estimated by sampling trajectories and, secondly, that the returns obtained by this sampling are informative for deciding the next action at the root of the search tree. We review its basic concepts below and refer the interested reader to [51] for more information.

In MCTS, each node in the search tree stores several statistics such as the sum of returns and the node visit count in addition to the state. For deciding each action, the search task is given a computational budget expressed in terms of node expansions or wall clock time. The algorithm keeps executing the following sequence of steps until the search budget is exhausted:

1. **Selection**: The tree is traversed iteratively from the root until an expandable node (i.e., a node containing a non-terminal state with yet-unexplored actions) is reached.

2. **Expansion**: From the expandable node, one or more new nodes are constructed and added to the search tree, with the expandable node as the parent and each child corresponding to a valid action from its associated state. The mechanism for selection and expansion is called *tree policy*, and it is typically based on the node statistics.

3. **Simulation**: Trajectories in the MDP are sampled from the new node until a terminal state is reached and the return (discounted sum of rewards) is recorded. The *default policy* or *simulation policy* dictates the probability of each action, with the standard version of the algorithm simply using uniform random sampling of valid actions. We note that the intermediate states encountered when performing this sampling are not added to the search tree.

4. **Backpropagation**: The return is backpropagated from the expanded node upwards to the root of the search tree, and the statistics of each node that was selected by the tree policy are updated.

The tree policy used by the algorithm needs to trade off exploration and exploitation in order to balance actions that are already known to lead to high returns against yet-unexplored paths in the MDP for which the returns are still to be estimated. The exploration-exploitation trade-off has been widely studied in the multi-armed bandit setting, which may be thought of a single-state MDP. A representative method is the Upper Confidence Bound (UCB) algorithm [12], which computes confidence intervals for each action and chooses, at each step, the action with the largest upper bound on the reward, embodying the principle of optimism in the face of uncertainty.

Upper Confidence Bounds for Trees (UCT) [213] is a variant of MCTS that applies the principles behind UCB to the tree search setting. Namely, the selection decision at each node is framed as an independent multi-armed bandit problem. At decision time, the *tree policy* of the algorithm selects the child node corresponding to action a that maximises

$$UCT(s,a) = \bar{r_a} + 2\epsilon_{\text{UCT}}\sqrt{\frac{2\ln C(s)}{C(s,a)}}, \tag{2.16}$$

where $\bar{r_a}$ is the mean reward observed when taking action a in state s, $C(s)$ is the visit count for the parent node, $C(s,a)$ is the number of child visits, and ϵ_{UCT} is a constant that controls the level of exploration [213].

MCTS is easily parallelisable either at root or leaf level, which makes it a highly practical approach in distributed settings. It is also a generic framework that does not make any assumptions about the characteristics of the problem at hand. The community has identified ways in which domain heuristics or learned knowledge [139] can be integrated with MCTS such that its performance is enhanced. This includes models learned with RL based on linear function approximation [312] as well as deep neural networks [153].

MCTS has been instrumental in achieving state-of-the-art performance in domains previously thought intractable. It has been applied since its inception to the game of Go, perceived as a grand challenge for Artificial Intelligence. The main breakthrough in this space was achieved by combining search with deep neural networks for representing policies (*policy networks*) and approximating the value of positions (*value networks*); the resulting approach was able to surpass human expert performance [314]. Subsequent works have significantly improved on this by

performing search and learning together in an iterative way: the search acts as the expert, and the learning algorithms imitate or approximate the play of the expert. The AlphaGo Zero [315, 316] and Expert Iteration algorithms [11] both epitomise this idea, having been proposed concurrently.

Monte Carlo Tree Search has found wide applicability in a variety of decision-making and optimisation scenarios. It is not solely applicable to two-player games, and has been successful in a variety of single-player games (also called *solitaire* or *puzzle* [51, Section 7.4]) such as Morpion Solitaire [293] and Hex [258, 11]. Indeed, given that such games involve creating connections on a regular grid, they served as a source of inspiration for leveraging it to optimise generic graph processes. Techniques combining MCTS with deep neural networks have also been fruitfully applied outside of games in areas such as combinatorial optimisation [222, 43], neural architecture search [349], and knowledge graph completion [311].

Let us also briefly discuss some of the limitations of MCTS [51]. Firstly, being a discrete search algorithm, it is limited by the depth and branching factors of the considered problem. To obtain good performance, a task-dependent, possibly lengthy, process is required to find effective ways of reducing them. Secondly, it is not directly applicable to continous control problems, and discretization can lead to a loss of generality. Thirdly, it has proven difficult to analyze using theoretical tools, a characteristic shared by other approximate search algorithms. An implication of this is that its performance as a function of the computational budget or parameters are not well understood, requiring adjustments based on trial-and-error.

2.5.6 Overview of Other Relevant RL Techniques

There are several other topics in RL that are relevant to the present thesis. The related area of Inverse RL focusses on extracting the reward function itself from demonstrated trajectories [264]. Early algorithms in this area focussed on learning reward functions that are linear with respect to some state features [1]. However, there may be a variety of potential reward functions that match the behaviour of the expert. The principle of maximum entropy has been applied to learn reward functions that are as "random" as possible while still matching the expert trajectories [381]. Subsequent works draw a connection to Generative Adversarial Networks, adopting them as a means of scaling Inverse RL to problems with large state spaces and unknown

dynamics [123, 128]. Recently, a technique has also been devised for performing scalable Inverse RL in a multi-agent setting [374].

The topic of performing effective exploration is also highly relevant in RL. Recall the aforementioned principle of optimism in the face of uncertainty leveraged in the UCB algorithm for the bandit setting. An alternative approach to UCB is the probabilistic Thompson sampling [63], which works by estimating the distribution of rewards conditioned on the history. Even though the same principles apply to MDPs, it is significantly more difficult to estimate regret bounds in this setting since state spaces can be very large. Exploration strategies become increasingly important in Deep RL settings, in which using function approximation to quantify the "familiarity" of a state [26] or creating a hashcode based on latent variables fit to the state space [328] are viable options.

2.6 ML for Optimising Graph Processes

Having introduced the necessary ML notions, in this section we are well-equipped to review other related works that apply such methods to the optimisation of processes taking place on graphs. In the first part of this section, we review the extensive body of work that applies ML to classic combinatorial optimisation problems. However, as we have argued extensively in Chapter 1, such methods can rarely compete with state-of-the art solvers when applied to these well-studied problems.

In the second part, we review works that tackle the optimisation of other graph processes with ML methods, focussing on those relevant to this thesis. For such problems, solution methods close to optimality or strong heuristics are typically not known. We examine learning-based approaches for constructing graphs and routing network flows, processes that have been addressed from a variety of perspectives. For completeness, we also briefly cover papers that discuss the optimisation of other graph processes.

2.6.1 Classic Graph Combinatorial Optimisation Problems

As we have discussed extensively in the Introduction (Chapter 1), in the past years there has been a growing interest from the community in addressing canonical NP-hard combinatorial optimisation problems on graphs with ML.

A problem that has received a large amount of attention is the aforementioned

TSP. It involves finding a route through a set of points placed in a space such that the total travel distance is minimised, subject to the constraint that each point is only visited once. While it is not a graph problem at first glance, it is possible to formulate the TSP as a process occurring over a fully connected graph. Finding a solution involves the selection of a set of weighted edges such that the total sum of weights is minimised, subject to the constraint that they form a Hamiltonian cycle. This graph-centric view of the problem enables one to use powerful tools and algorithms to reason about and address it.

In this section, we set out to review works that leverage ML for classic combinatorial optimisation problems on graphs. Despite clear differences between such *problems* and those addressed by this thesis, the *methods* used for solving them are similar to those that we have leveraged. We consider that it is important to examine the possible methodological choices so that we may relate our design decisions to this body of work. Even within the realm of classic graph combinatorial optimisation problems, such methodological choices are not always clearly justified, persisting as a challenge to the field [193].

The application of ML techniques for graph combinatorial problems is by no means new and can be traced to the application of Hopfield networks by the eponymous author to the TSP in 1985 [176]. This paper has spurred more than a decade of research on applications of ML to a large variety of combinatorial optimisation problems. The advantages that neural networks enabled, such as flexibility in the range of problems that can be addressed, as well as their fast evaluation times, were also recognised. Overall, however, this wave of work had varying degrees of success. At the time, the rather damning conclusion was that such approaches usually cannot compete with standard heuristics and exact methods [318], a finding that persists today to a certain extent [193].

Interestingly, this review anticipated the significant potential of neural networks if sufficient hardware and software advances were made in order to render their training and use more practical. Lately, a confluence of progress in tooling as well as learning representations and RL have led to a modern revival of this strand of research. In broad terms, the present thesis also belongs to this wave of interest.

Let us first review some of the seminal papers that have reignited interest in this

area. In 2015, Vinyals et al. [346] proposed a recurrent neural network architecture (Pointer Network) for sequence modelling, the inputs to which are discrete and outputs correspond to a position (*pointer*) in the input sequence. This overcame a limitation of previous Recurrent Neural Network (RNN) models that constrained input and output sizes to be equal. It enabled the model to be used on output sequences of variable lengths and, hence, on instances of combinatorial optimisation problems that are different in size. One of the case studies used in this paper was the TSP, for which the authors used Supervised Learning of labelled data produced by a combinatorial solver. The approach showed good results on instances of size up to 50, and demonstrated the ability to satisfactorily apply a model on larger instances than the ones on which it had been trained.

Bello et al. [29] noted the inherent limitation of Supervised Learning from demonstrations of known solvers. Instead, they proposed an actor-critic RL method trained using the policy gradient, as well as improved procedures for inference. These enhancements enabled the resulting model to scale well to TSP instances of size up to 100. Subsequently, Khalil et al. [203] achieved several remarkable results. They proposed a framework (S2V-DQN) based on the DQN algorithm together with a GNN representation than can solve several graph combinatorial optimisation problems at once. It was shown to scale to TSP instances upwards of 1000 nodes, while maintaining excellent approximation ratios and outperforming classic heuristics.

We now discuss subsequent works, formulating some dimensions along which they can be classified.

Problems Addressed

As we have noted, many works in this area have considered applications to several variants of the TSP [346, 29, 203, 216, 220, 206], as well as other problems that aim to devise optimal tours for visiting nodes placed in a space.[1] An example is work on the Prize Collecting TSP [206], which additionally involves prizes for nodes that are visited and penalties for those that are not.

[1]We note that some works in this field refer to this family of problems as *routing* problems, since they involve devising routes, i.e., sequences of nodes to be visited. This is an unfortunate clash in terminology with problems that involve the routing of flows over graphs, which were discussed in Section 2.3.3, and to which Chapter 6 is dedicated. The problems have little in common in structure and solution methods beyond this superficial naming similarity.

The VRP has also been approached with ML methods. In the VRP, there is a special node called *depot*, at which several vehicles need to start and end their tours. The goal is to devise a set of tours for the vehicles (rather than a single tour as in the TSP) that minimises the total cost. Different extensions of the problem were considered in the literature: differing capacities for vehicles and customer nodes (the Capacited VRP, CVRP) [259], as well as time windows between which the deliveries must be made (the VRP with Time Windows, VRPTW) [119].

Beyond the TSP and related problems, other works have considered ML for graph combinatorial optimisation problems that involve the *selection of a subset of nodes in a graph*. A widely used concept in this setting is that of a Maximal Independent Set: a set of nodes defined such that every node is either a member of the set, or is directly connected to a member of this set. The Maximum Independent Set is a Maximal Independent Set of the largest possible size. A strongly related problem is the Minimum Vertex Cover (MVC), the smallest set of nodes that includes at least one endpoint of all the edges. It is known that the MIS and MVC sets are complements of each other, and hence solutions to one problem can be used to solve the other. Both have been approached using a variety of ML techniques [203, 234, 5, 43].

ML approaches have also been applied to combinatorial optimisation problems that *do not take place on a graph*, such as the knapsack [29, 220], bin packing [222], as well as the job shop scheduling problem [378]. However, the body of work in this area is substantially smaller.

Learning Paradigm

Some works adopted Supervised Learning [346], but the vast majority opted for Reinforcement Learning [29, 203, 216, 220]. Other papers combined both approaches by adopting Supervised Learning as a pre-training step, and subsequently performing fine-tuning [29, 372]. Since data used for supervised training usually comes in the form of demonstrations of a known solver or algorithm, this may also be viewed as a form of Imitation Learning, which additionally requires a compatible MDP formulation (e.g., as performed in [110]).

Learning Representation

Many of the earlier works adopted the aforementioned Pointer Network architecture [346, 29, 259, 83]. Subsequently, the attention model proposed by Kool

et al. [216] gained traction and was used in several recent works [220, 205], while other papers proposed their own variants of attention-based policies [365, 179, 237]. Other works used GNNs such as the previously discussed S2V [203], GCN [234], and the scalability-targeted GraphSAGE that performs message-passing over a sampled neighbourhood of fixed size [160, 5]. The Pointer Network and attention-based models are overwhelmingly used for TSP and its extensions, whereas GNNs are popular as well as effective in problems for which graph structure matters [203].

Performing Inference

After training a model, the most straightforward way to construct a solution is to perform greedy evaluation of the trained policy. While simple, this basic scheme has been shown to perform well even on very large instances with up to 1000 nodes [203]. Various improvements over this scheme exist. For example, one may allow the agent to reverse its decisions and continue to explore at test time [21].

Better solutions over greedy evaluation may also be found by using the model to perform sampling of many trajectories, subsequently selecting the one with the best objective function value. This, however, discards the reward information contained in the sampled trajectories. Active search [29] uses this data and continues to adjust the model weights at inference time. Beam Search, a method akin to Breadth-First Search with a limited number of expansions per level according to a score, has also been applied for performing inference [346, 192].

Other approaches take the recipe of combining search and ML further. Li et al. [234] proposed a tree search method that uses a GNN to bias the navigation of the search space towards promising solutions, achieving gains over S2V-DQN for the MIS problem. However, a recent paper [43] that attempted to reproduce its results debunked the contribution of the GNN for this purpose, showing that its priors for node expansion are as effective as using random values. The use of graph kernelisation was instead found to be responsible for the observed performance.

Abe et al. [2] proposed an extension of AlphaGo Zero for addressing graph combinatorial optimisation problems. Their MCTS-based approach, which was paired with various types of GNNs, showed improvements over S2V-DQN in some settings. Laterre et al. [222] also used MCTS together with a neural network. Their method additionally reshapes the reward signal such that the agent is incentivised

to improve over its performance in the last iteration, justified as a form of "self-play" for single-player games.

Symmetries

Symmetries that are present in the problem and solution spaces are also highly relevant. For example, a TSP solution that visits nodes (A, B, C) is equivalent to one that visits (B, C, A). Exploiting this may lead to better sample complexity and more robust models. Approaches for doing so have considered encoding symmetry with specific terms in the loss function [220, 206] or augmenting the dataset of sampled solutions at inference time [220]. A recent work [110] proposed exploiting symmetries at the level of the MDP formulation itself by proposing a transformation of the original MDP that reduces the state space. Symmetries may also be encoded in the learning representation itself, as can be achieved by the use of GNNs with appropriate permutation-invariant readout functions [203].

Construction and Improvement Methods

A dichotomy is present in this literature between *construction* and *improvement* methods. The former, which makes up the majority of papers discussed in this section, refers to approaches that build the solution incrementally, starting from an empty set. For example, in the context of the TSP, one would begin with the empty sequence and continue adding nodes until the tour is complete.

The latter category begins with an already-built solution that is refined over time. This is achieved by applying certain heuristic operators which have a high chance to improve the solution, possibly integrated inside a higher-level search or metaheuristic algorithm. In this space, Wu et al. [365] and da Costa et al. [83] considered learning the selection of a pair of nodes to which a local improvement heuristic is applied, training an actor-critic algorithm for this purpose. An example of such a pairwise local heuristic that has been widely used for TSP and its extensions is *2-opt* [81], which involves reversing a segment of the solution, e.g., (A, B, C, D) becomes (D, C, B, A). If applied at an appropriate location, it can lead to the "uncrossing" of routes, which greatly reduces travel costs.

Hottung and Tierney [179] adopted the Adaptive Large Neighbourhood Search (ALNS) metaheuristic, which is based on simulated annealing, and applies pairs of "destroy" and "repair" operators that deconstruct and reconstruct parts of the

solution. Their approach learns the repair operators using an actor-critic mechanism. Lu et al. [237] used a large library of predefined operators from the literature, and learned a controller for selecting which repair operator to apply using a policy gradient method. Chen and Tian [67] trained two policies: a region-picking policy that selects the part of the solution to be improved, and a rule-picking policy that selects one of the predefined rewriting rules.

One of the advantages of improvement methods is that the large body of prior work in operations research can be leveraged. Algorithms for constructing a solution of good quality may already exist, and it is reasonable to start the procedure from one such solution. An example algorithm is the "savings" procedure proposed by Clarke and Wright [74] for the TSP and VRP. In terms of disadvantages, improvement methods require knowledge of strong heuristic operations. Hence they may not perform well if the problem at hand has not been sufficiently studied and such operators are not known.

2.6.2 Learning to Construct Graphs

A shared characteristic of the graph combinatorial optimisation problems discussed in the previous subsection is that they do not involve changes in topology to the graph. Concretely, one needs to find a solution while assuming that the network structure remains fixed. However, as discussed in Section 2.3, networks in the real world undergo structural changes regularly, which influences the outcome of processes taking place over them. In this prior section, we have also covered a variety of non-ML methods that were applied for modifying the topology of a graph in order to optimise a quantity of interest related to such a process.

The problem of learning to construct a graph or to modify its structure has received comparatively less attention in the ML literature. It may be classified into two strands of work. The first considers the explicit optimisation of a given objective function using RL. The second focusses on building graphs that are similar to a dataset of known examples in some way, which is a form of unsupervised learning. This is typically performed using a deep generative model, and may be seen as a ML based alternative of the classic models discussed in Section 2.2.

Prior to delving into the details, let us compare these lines of enquiry at a high level. RL methods require the ability to simulate the process of interest so that an

agent may learn through interactions with an environment. Generative models, on the other hand, primarily use datasets of examples of the "finished product", without access to the steps of the generation process. They additionally require large collections of related examples, which may not always be available depending on the domain of interest. RL methods are also more granular and can be used to extend existing structures, whereas generative models are typically "one-shot".

Explicitly Optimising an Objective Function

The work of Dai et al. [86] treats the problem of learning to modify the topology of a graph through edge additions and removals. It considers an adversarial setting in which the objective is to induce a deep graph or node-level classifier to make labelling errors. The problem can be thought of as the graph-based equivalent of finding adversarial perturbations for image classifiers based on deep neural networks [38, 327]. The approach proposed by the authors, called RL-S2V, is a variant of S2V-DQN in which the action space is decomposed for scalable training: edge additions are formulated as two node selections. The evaluation performed by the authors showed that it it compares favourably to attacks based on random edge additions and those discovered by a genetic algorithm.

You et al. [372] considered learning to construct molecular graphs. The objective functions that the method seeks to optimise are the drug-likeness and synthetic accessibility of molecules. The action space is defined as the addition of bonds or certain chemical substructures, and the transition function enables the environment to enforce validity rules with respect to physical laws. The proposed approach, called Graph Convolutional Policy Network (GCPN), uses a GNN representation of the policy, which is trained using PPO. In addition to the objective functions, the reward structure incentivises the method to generate molecules that are similar to a given dataset of examples. GCPN was shown to outperform a series of previous generative models for the task.

GraphOpt [338] tackles the *inverse problem* of the one discussed thus far. Namely, given a graph, the goal is to learn a plausible underlying objective function that has lead to its generation. The authors proposed an MDP formulation of graph construction through edge additions, and used maximum entropy inverse RL and GNNs. The authors empirically showed that the learned model is able to generate

graphs that match statistics such as the degree and clustering coefficient distributions of the observed graphs. The resulting model can also be used to generate similar examples of networks that optimise the discovered objective, as well as for performing link prediction.

Deep Generative Models of Graphs

We begin our discussion of these approaches by giving a brief overview of generative modelling. It is a central problem in unsupervised learning that concerns itself with learning or approximating the data-generating process; once learned, such models can be used to produce samples or as a starting point for supervised tasks.

Deep generative models have been successful in generating realistic text and images at unprecedented scale and sample quality [286, 267, 197]. Naturally, interest in generative models has also extended to the domain of graphs, which poses distinct challenges: their discrete nature, combinatorial complexity, and the fact that the structure is permutation-invariant. Additionally, graph data is not as widely available as images or sounds; often, models need to work with very few samples. Still, significant advances have been achieved in this domain.

There are three main classes of deep generative models currently in wide use:

1. Variational Autoencoders (VAEs): an explicit model for mapping data to and from a continuous latent space. It is trained by jointly optimising the evidence lower bound (ELBO) between an encoder and decoder, which typically use deep neural network architectures [208].

2. Generative Adversarial Networks (GANs): an implicit approach, in which data generation is modelled as a two-player minimax game with two opponents: a generator and discriminator. The generator aims to produce data indistinguishable from original training samples, whereas the discriminator tries to tell them apart [145].

3. Deep Auto-Regressive Models: a neural network is used to model the conditional distribution. Typically, a recurrent architecture such as Long-Short Term Memory (LSTM) [169] or Gated Recurrent Unit (GRU) [70] is used to capture conditional dependencies over long horizons.

Several works in the ML literature have considered the generation of graphs

with similar topological properties to a provided dataset. Li et al. [233] modelled the graph generation process as a sequence of decisions: addition of a new node to the graph, whether to add an edge, and picking the node for such an edge addition. They made use of a GRU architecture paired with a GNN, and showed good results for reconstructing both synthetic and real-world graphs. GraphRNN [373] modelled the process differently by considering two separate recurrent networks: one graph-level network for node generation, and an edge-level network which is used to predict the connectivity of the added node. It is faster to evaluate since it does not involve using a GNN for message passing, although it needs to impose a canonical ordering of node labels (achieved by using Breadth-First Search).

The Graph Recurrent Attention Network [235] method generates blocks of nodes and associated edges at a time, and thus can scale to larger graphs while maintaining good output quality. It is based on GNNs with attention for handling long-term dependencies. The authors of NetGAN [41] framed the problem of graph generation slightly differently. Their method is concerned with generating graphs similar to a *single* source graph. They instead proposed learning the distribution of random walks over graph as a surrogate for its topology, and used a GAN architecture to distinguish between genuine and generated random walks on the graph. Both the generator and discriminator used a LSTM architecture. A graph reconstruction method was also proposed based on the visitation counts for the nodes across the random walks.

Beyond the generation of raw topologies, progress in this area has been driven by applications in chemistry and material sciences. The manageable scale of the search space, which is estimated to lie between 10^{23} and 10^{60} for drug-like molecules [282], highlights the possibility of practical impact using computational methods. For example, deep generative modelling can act as a way of generating plausible candidates, so that fewer molecules need to be manually investigated in the drug discovery process. For representing molecules, works used either the SMILES string-based encoding [356] or a graph-based representation [372].

Character VAE [155] is based on a VAE in conjunction with a GRU to learn a generative model of molecular graphs represented as SMILES strings. However, the decoder does not necessarily output valid SMILES strings or chemically valid

molecules. GrammarVAE [219] improved on this approach by generating parse trees instead of raw strings – thus their syntactic validity could be ensured. SD-VAE [87] built further in this direction by not only considering syntactic but also semantic validity, similar to how a computer program may be syntactically valid but semantically erroneous at the same time. The authors of ORGAN [152] also operated with SMILES representations, but instead used a sequential GAN together with an RL objective for training with domain-specific feedback.

SMILES string representations of molecules are limited to a certain extent since two molecules that are structurally very similar can have completely different representation strings. Furthermore, it is simpler to check chemical properties directly on graphs than such strings [372]. For this reason, works have considered generating the graph connectivity information and node/edge attributes directly. GraphVAE was the first in this series of works [317], in which the authors considered generating the entire graph in one step. To achieve this, they used a VAE architecture together with a graph matching algorithm in order to quantify the reconstruction ability of the decoder. However, this matching procedure is an $\mathcal{O}(|V|^4)$ operation and can only apply to relatively small graphs. MolGAN [99] circumvented this problem by using a GAN, which does not require computing the explicit likelihood of node permutations. The generator was trained to minimise the discriminator loss together with an RL objective that quantifies the "goodness" of the generated molecules.

Nevertheless, models for molecular graphs generated by all of the above-listed approaches still suffer from relatively low validity scores. The Junction Tree VAE or JT-VAE [190] significantly improved on previous approaches by considering the basic valid building blocks of molecules as constructing a tree-like scaffolding (which they call *junction tree*), as well as applying GNNs specifically designed for this setting. The aforementioned GCPN method subsequently outperformed JT-VAE by a large margin, highlighting the potential of RL with graph representations to carry out a structured exploration of the solution space, as well as its ability to generate solutions that obey validity constraints.

2.6.3 Learning to Route Network Flows

The routing of flows across a network topology has been approached from two different perspectives in the ML literature. A first wave of interest considered routing

at the packet level in a multi-agent RL formulation. The second, more recent, wave of interest originated in the computer networks community, which has begun to recognise the potential of ML methods in this space [120, 189].

The first work in this area dates back to a 1994 paper in which Boyan and Littman proposed Q-routing, a means of performing routing of packets with multi-agent Q-learning [44]. In this framework, an agent is placed on packet-switching nodes in a network; nodes may become congested and so picking the shortest path may not always yield the optimal result. Agents receive neighbours' estimate of the time remaining after sending a packet and iteratively update their estimates of the Q values in this way. The authors showed, on a relatively small network, that this approach is able to learn policies that can adapt to changing topology, traffic patterns, and load levels.

Subsequent works have introduced variations or improvements on this approach: Stone [322] considered the case in which nodes are not given information about their neighbours, and applied a form of Q-learning with function approximation where states are characterised by feature vectors. Another approach that instead uses policy gradient methods [330] was able to learn co-operative behaviour without explicit inter-agent communication, adapt better to changing topology, and is amenable to reward shaping. Peshkin and Savova [280] proposed a softmax policy trained using a variant of REINFORCE, which was shown to perform substantially better than Q-routing in scenarios for which the optimal policy is stochastic.

Recent research on routing with ML in the computer networks community [142, 339, 300] generally considers routing at the *flow* level rather than the more granular packet level, which tends to be a more scalable formulation of the problem, and is more aligned to current routing infrastructure.

There have been several recent works on learning to route using either conventional neural network architectures or GNN-like architectures. Geyer and Carle [142] proposed a variant of the Gated GNN [232] and trained it to predict paths taken by conventional routing algorithms. Rusek et al. [300] proposed a MPNN variant and used it to predict graph-level metrics such as delay and jitter. Reis et al. [287] used an MLP representation and Supervised Learning to predict the full path that a flow should take through the network.

Other works have considered learning the routing protocol itself with RL in a variety of problem formulations: Valadarsky et al. [339] used an MLP and considered learning per-edge coefficients that are used with "softmin" routing. Xu et al. [368] proposed an MLP approach for learning traffic split ratios for a set of candidate paths. Zhang et al. [380] used a CNN to re-route a proportion of important (critical) flows. Almasan et al. [10] introduced a formulation that routes flows sequentially, which then become part of the state. It uses a MPNN representation. Most recently, Hope and Yoneki [175] adopted the formulation in [339], showing that the use of Graph Networks [24] improves performance in one graph topology.

Finally, a work that does not fit in either category but is highly relevant was performed in the neural algorithmic reasoning literature. Georgiev and Liò [140] trained two GNN variants to mimic the steps taken by the Ford-Fulkerson algorithm for finding the maximum flow that can be routed across a graph without violating capacity constraints [80, Chapter 24]. The authors showed that the considered GNNs obtain strong generalisation on unseen inputs.

2.6.4 Learning to Optimise Other Graph Processes

To complete our picture of related work, let us now discuss the literature on ML for the optimisation of other graph processes.

Learning in Network Games

Network games, which are the remaining class of graph processes addressed by the contributions of this thesis, have attracted comparatively less attention from the research community.

A core question that has been addressed in this area is the inference of the structure of a network from data about the actions taken by the players in a network game, under a variety of formulations and assumptions. Honorio and Ortiz [173] studied the class of linear influence games [183] with binary actions and linear payoffs, deriving an algorithm for this purpose with guarantees in terms of sample complexity. Garg and Jaakkola [135] treated the problem of recovering the network in graphical games that are structured as trees, and the utilities are such that the game is a potential game. Leng et al. [227] considered a network game with continuous actions, proposing an algorithm that recovers the individual marginal benefits in addition to the graph structure. Most recently, the authors of [295] proposed a

method that does not require knowledge of the utility function, and yet compares favourably to existing methods.

Trivedi and Zha [337] focussed on learning in the related class of *network emergence games*. In this category of games, it is the strategic behaviour of agents themselves that leads to the creation of links, rather than the graph merely governing the structure of interactions. The paper seeks to recover the unknown utility function from an observed graph structure, using a GNN-parametrised policy and a multi-agent inverse RL algorithm. Methodologically, this work is substantially closer to the deep learning centric approach of this thesis, whereas the works discussed in the above paragraphs rely on tractable inference procedures that make use of simplifying assumptions (e.g., linearity and convexity).

Learning for Spreading Processes

A few recent papers have considered applications of ML to spreading processes on graphs, especially in the context of the COVID-19 pandemic. Meirom et al. [248] proposed an approach based on RL and GNNs for controlling spreading processes taking place on a network of agents. Their method was applied for controlling an epidemic spreading process, for which the agent decides which node should be tested and subsequently isolated, as well as an influence maximisation process, for which the agent decides which node should be selected as the target to be influenced. Their method was shown to perform better than several prior heuristics (e.g., removing highly central nodes in epidemic processes).

Panagopoulos et al. [269] considered a higher level of abstraction in modelling the spread of an epidemic by extracting the network of regions within a country. Two regions are connected in the network if they share a geographical border, with edge weights quantifying the number of people moving between them. The task is to estimate the number of future cases for a given region. The authors successfully leveraged a MPNN for this purpose, additionally demonstrating that the model transfers well when applying it to different countries.

Learning to Search

Search processes have also been studied in the ML literature, especially in the context of reasoning in knowledge graphs [324, 42]. The task is typically formulated as completing a query: given an entity and a relation, the goal is to find the missing

entity. This is realised through guided walks over the knowledge graph. A model is trained using queries that are known to be true, and subsequently applied to tuples for which the knowledge is incomplete.

Das et al. [97] formulated the task as an MDP and tackled it with an LSTM architecture trained using a policy gradient algorithm. M-Walk [311] obtained improvements over this approach by combining a policy network with Monte Carlo Tree Search. Finally, a recent work by Zhang et al. [379] addressed the issue of degrading performance of such models with increases in path length. The authors proposed a design with two policies that act cooperatively: one higher-level policy for picking the cluster in the knowledge graph to be searched, and a fine-grained policy that operates at the entity level.

2.7 Summary

In this chapter, we have discussed the required background notions and covered the relevant related work. We began by formally defining networks and high-level mathematical properties that may be used to examine them. We then covered a series of processes taking place over graphs – namely, robustness, efficiency, network flows, and network games; as well as conventional methods that have been proposed in the past for their optimisation. Subsequently, we discussed the core methodological tools used in this thesis: learning representations operating on graphs and techniques for solving decision-making processes. Finally, we have reviewed other works that use Machine Learning for the optimisation of graph processes, to which the present thesis is closely related.

Given that we have completed our discussion of related work, let us close this chapter by placing the research questions formulated in Section 1.3 in broader context.

Research questions ***RQ1***, ***RQ2***, and ***RQ5*** address the gap in the literature regarding the application of RL techniques beyond canonical combinatorial optimisation problems to other graph processes of practical interest. Specifically, these research questions target the problems of goal-directed graph construction (Chapters 3 and 4), and the determination of an optimal Maximal Independent Set (Chapter 5). As discussed in Section 2.3, current approaches for solving these problems primarily rely on hand-crafted heuristics and generic metaheuristics. The use of RL may enable substantial advantages in addressing them.

Furthermore, ***RQ3*** and ***RQ6*** are motivated by challenges specific to graph combinatorial optimisation, and seek to tackle the design of algorithms and learning representations that are well-suited for problems in this space. Chapter 4 shows a blueprint for how a standard algorithm for decision-making (UCT) may be fruitfully extended for the particulars of a family of combinatorial optimisation problems. The inductive bias that is encoded in the learning representation is also important. As demonstrated by Chapter 6, the wrong choice of inductive bias can prove harmful, and it is crucial to design the learning architecture such that it is aligned to the problem characteristics.

Finally, an important practical aspect that we address is how to obtain scalability on large problem instances (***RQ3*** and ***RQ5***). Chapters 3 and 5 study the application of models trained on small problem instances to larger ones. This technique can obtain impressive results in scenarios in which the model transfers well. Further options that we adopt are the use of decision-time planning to examine a small fraction of the entire decision-making process (Chapters 4 and 5), pruning the action space (Chapter 4), and using demonstrations of a well-performing algorithm to collect data instead of online environment interaction (Chapter 5).

Chapter 3

Goal-directed Graph Construction using Reinforcement Learning

As discussed extensively in Chapter 2, several metrics have been devised to quantify the global characteristics of graph-structured systems. However, comparatively little is currently known about how to construct a graph or improve an existing one by adding edges so as to optimise a target objective function. Our starting hypothesis is that it may be possible to discover, in a data-driven way, algorithms that perform better than current heuristics that are based on local and spectral properties.

In this chapter, we pursue a generic framework for optimising the structural properties of graphs. Towards this goal, we formulate the construction of a graph as a decision-making process in which a central agent creates topologies by trial and error and receives rewards proportional to the value of the target objective. We then propose a high-level algorithm based on RL and GNNs to learn strategies for graph construction and improvement. Our core case study focusses on robustness to failures and attacks, a dimension relevant for the infrastructure and communication networks that power modern society.

3.1 Introduction

Graphs are mathematical abstractions that can be used to model a variety of systems, from infrastructure and biological networks to social interactions. Various methods for analysing networks have been developed: these have been often used for

understanding the systems themselves and range from mathematical models of how families of graphs are generated [354, 19] to measures of centrality for capturing the roles of vertices [37] and global network characteristics [263], to name but a few.

A measure that has attracted significant interest from researchers and practitioners is *robustness* [262] (sometimes called *resilience*), which is typically defined as the capacity of the graph to withstand random failures, targeted attacks on key nodes, or some combination thereof. A network is considered robust if a large fraction (*Critical Fraction*) of nodes have to be removed before it becomes disconnected [76], its diameter increases [8], or its Largest Connected Component diminishes in size [35]. Previous work has focussed on the robustness of communication networks, such as the Internet [77] and infrastructure networks used for transportation and energy distribution [61], for which resilience is a key property.

In many practical cases, an initial network is given and the only way of improving its robustness is through the modification of its structure. This problem was first approached by considering edge addition or rewiring, based on random and preferential (with respect to node degree) modifications [35]. Alternatively, a strategy has been proposed that uses a "greedy" modification scheme based on random edge selection and swapping if the resilience metric improves [306]. Another line of work focusses on the spectral decomposition of the graph Laplacian, and using properties such as the algebraic connectivity [348] and effective graph resistance [350] to guide modifications. While simple and interpretable, these strategies may not yield the best solutions or generalise across networks with varying characteristics and sizes. Certainly, better solutions may be found by Exhaustive Search, but the time complexity of exploring all the possible topologies and the cost of computing the metric render this strategy infeasible. With the goal of discovering better strategies than existing methods, we ask whether *generalisable network construction strategies for improving robustness can be learned*.

Starting from this motivation, we formalise the process of graph construction and improvement as an MDP in which rewards are proportional to the value of a graph-level objective function. We consider two objective functions that quantify robustness as the Critical Fraction of the network in the presence of random failures and targeted attacks. Inspired by recent successes of RL in solving combinatorial optimisation

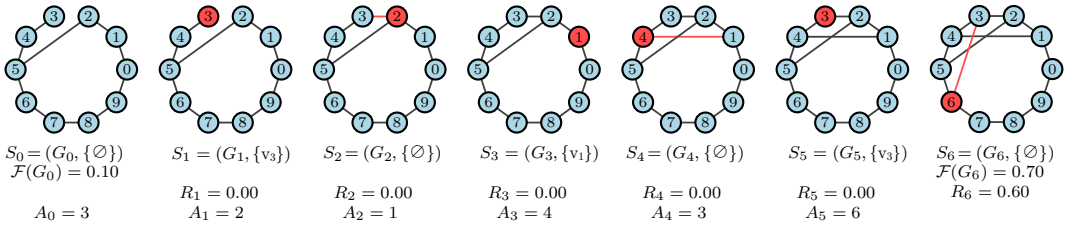


Figure 3.1: Illustration of a Graph Construction MDP (GC-MDP) trajectory. The agent is provided with a start state $S_0 = (G_0, \{\varnothing\})$. It must make $b = 3$ edge additions over a sequence of 6 node selections (actions A_t), receiving rewards R_t proportional to the value of an objective function $\mathcal{F}$ applied to the graph. In this case, $\mathcal{F}$ quantifies the robustness of the network to targeted node removal, computed by removing nodes in decreasing order of their degree and in decreasing order of the labels if two nodes have the same degree. We observe an improvement of the robustness of the graph from $\mathcal{F}(G_0) = 0.1$ to $\mathcal{F}(G_6) = 0.7$. Actions and the corresponding edges are highlighted.

problems on graphs [29, 203], we make use of GNN architectures [303] together with the DQN [251] algorithm. Recent work in goal-directed graph generation and improvement considers performing edge additions for adversarially attacking GNN classifiers [86] and generating molecules with certain desirable properties using domain-specific rewards [372]. In contrast, to the best of our knowledge, this is the first time that RL is used *to learn how to construct a graph such as to optimise a global structural property*. While in this chapter we focus on robustness, other intrinsic global properties of graphs, such as efficiency [223] or communicability [117], could be used as optimisation targets.

The contribution of this chapter is twofold. Firstly, we propose a framework for improving global structural properties of graphs, by introducing the Graph Construction Markov Decision Process (GC-MDP). Secondly, focussing on the robustness of graphs under failures and attacks as a core case study, we offer an in-depth empirical evaluation that demonstrates significant advantages over existing approaches in this domain, both in terms of the quality of the solutions found as well as the time complexity of model evaluation. Since this approach addresses the problem of building robust networks with a DQN, we name it *RNet–DQN*.

3.2 Methods

In this section, we first introduce the proposed formalism of graph construction and the objective functions used. Subsequently, we propose a method for learning heuristics for constructing robust graphs, which relies on RL with function approximation.

3.2.1 Robust Graph Construction as an MDP

Modelling Graph Construction. Let $\mathcal{G}^{(N)}$ be the set of labelled, undirected, unweighted graphs with N nodes; each such graph $G = (V, E)$ consists of a vertex set V and edge set E. Let $\mathcal{G}^{(N,m)}$ be the subset of $\mathcal{G}^{(N)}$ with $|E| = m$. We also let $\mathcal{F}\colon \mathcal{G}^{(N)} \to [0, 1]$ be an objective function, and $b \in \mathbb{N}$ be a modification budget that defines the number of new edges that can be added. Given an initial graph $G_0 = (V, E_0) \in \mathcal{G}^{(N,m_0)}$ containing the edge set E_0 with cardinality m_0, the aim is to perform b edge additions to G_0 such that the resulting graph $G_* = (V, E_*)$ satisfies:

$$G_* = \operatorname*{argmax}_{G' \in \mathcal{G}'} \mathcal{F}(G'), \tag{3.1}$$
$$\text{where} \quad \mathcal{G}' = \{G = (V, E) \in \mathcal{G}^{(N,m_0+b)} \mid E_0 \subset E\}.$$

This combinatorial optimisation problem can be cast as a sequential decision-making process as follows. The agent will iteratively select two nodes between which a new edge will be added until the budget b is exhausted. Before diving into the details of the mathematical formulation, an important aspect to consider is the design of the action space. A straightforward choice would be to frame it as the selection of one of the $\mathcal{O}(N^2)$ non-edges to be added at each timestep, which would not be effective on large graphs.

Instead, to aid scalability, we decompose the addition of an edge into two separate node selection decisions, which means that $\mathcal{O}(N)$ choices need to be considered at each timestep. Concretely, the agent first selects, at time t, the node v_i from which an edge will be constructed. From this point onwards, in accordance with the Markov assumption, the agent "commits" to having v_i be the origin of an edge. Subsequently, at time $t + 1$, the agent selects node v_j, which is the other end of the edge. When transitioning to the state at time $t + 2$, the new edge $e_{i,j}$ between v_i and v_j is added to the graph, and the process repeats. Tasks are episodic, and each episode proceeds for at most $2b$ steps. A trajectory visualisation is shown in Figure 3.1.

Formally, we define the Graph Construction MDP (GC-MDP) as follows:

1. *State*: The state S_t is a tuple $(G_t, \{\sigma_t\})$ containing the graph $G_t = (V, E_t)$ and a singleton containing an *edge stub* σ_t. At even timesteps ($t \bmod 2 = 0$), σ_t is

equal to the empty set $\varnothing$. At odd timesteps, σ_t is equal to $v_{k_{\text{from}}}$, where $v_{k_{\text{from}}} \in V$ is the node that was selected in the previous timestep, from which an edge must be built.

2. *Action*: A_t corresponds to the *selection of an index identifying a node* in V. The set of available actions, containing the indices of the nodes that may be selected, are defined as below. The first clause states that maximally connected nodes cannot be selected as the edge stub, while the second clause forbids actions that would lead to the construction of an already-existing edge.

$$\mathcal{A}(S_t) = \begin{cases} \{\text{select } k_{\text{from}} \quad \mid v_{k_{\text{from}}} \in V \wedge \deg(v_{k_{\text{from}}}) < |V| - 1\}, \text{ if } t \bmod 2 = 0 \\ \{\text{select } k_{\text{to}} \quad \mid v_{k_{\text{to}}} \in V \wedge e_{k_{\text{from}}, k_{\text{to}}} \notin E_t\}, \text{ otherwise.} \end{cases} \tag{3.2}$$

3. *Transitions*: The transition dynamics are deterministic, meaning that, from a state s, there is a single state s' that can be reached with probability 1. Recall that the Kronecker delta δ_{xy} is equal to 1 if $x = y$ and 0 otherwise. The transition model is defined as $P(S_{t+1} = s' | S_t = s, A_t = a) = \delta_{ss'}$,

$$\text{where } s' = \begin{cases} \left((V, E_t), \{v_a\}\right), \text{ if } t \bmod 2 = 0 \\ \left(\left(V, E_t \cup \{e_{k_{\text{from}}, a}\}\right), \{\varnothing\}\right), \text{ otherwise.} \end{cases} \tag{3.3}$$

Written in plain English, when transitioning from an even timestep, the model "marks" the node corresponding to the selected action as the edge stub so that it forms part of the next state. Otherwise, it adds the edge corresponding to the selections to the topology of the next state, and resets the edge stub.

4. *Reward*: The reward R_t is defined as follows[1]:

$$R_t = \begin{cases} \mathcal{F}(G_t) - \mathcal{F}(G_0), \text{ if } t = 2b \\ 0, \text{ otherwise.} \end{cases} \tag{3.4}$$

[1]Since $\mathcal{F}$ is very expensive to estimate, we deliberately only provide the reward at the end of the episode in order to make the training feasible computationally, to the detriment of possible credit assignment issues. Intermediate rewards based on the true objective or a related quantity represent a middle ground which we leave for future work.

Definition of Objective Functions for Robustness. We are interested in the robustness of graphs as objective functions. Given a graph G_t, we let the *Critical Fraction* $\text{CF}(G_t, \xi) \in [0, 1]$ be the minimum fraction of nodes that have to be removed from G_t in some order ξ for it to become disconnected (i.e., have more than one connected component). Connectedness is a crucial operational constraint and the higher this fraction is, the more robust the graph can be said to be.[2] The order ξ in which nodes are removed can have an impact on CF, and corresponds to different scenarios: random removal is typically used to model arbitrary failures, while targeted removal is adopted as a model for attack. Formally, we consider both random permutations ξ_{random} of nodes in G_t, as well as permutations $\xi_{targeted}$, which are subject to the constraint that nodes must appear in the order of their degree, i.e.,

$$\forall v_i, v_j \in V.\ \xi_{targeted}(v_i) \leq \xi_{targeted}(v_j) \iff \deg(v_i) \geq \deg(v_j). \tag{3.5}$$

We define the objective functions $\mathcal{F}$ in the following way:

1. *Expected Critical Fraction to Random Removal*:

$$\mathcal{F}_{random}(G_t) = \mathbb{E}_{\xi_{random}}[\text{CF}(G_t, \xi_{random})] \tag{3.6}$$

2. *Expected Critical Fraction to Targeted Removal*:

$$\mathcal{F}_{targeted}(G_t) = \mathbb{E}_{\xi_{targeted}}[\text{CF}(G_t, \xi_{targeted})] \tag{3.7}$$

We use MC sampling for estimating these quantities. For completeness, Algorithm 3 in Appendix A describes how the simulations are performed. In the remainder of the chapter, we use $\mathcal{F}_{random}(G_t)$ and $\mathcal{F}_{targeted}(G_t)$ to indicate their estimates obtained in this way. We highlight that evaluating an MC sample has time complexity $\mathcal{O}(|V| \cdot (|V| + |E_t|))$: it involves checking connectedness (an $\mathcal{O}(|V| + |E_t|)$ operation) after the removal of each of the $\mathcal{O}(|V|)$ nodes. Typically, many such samples need to be used to obtain a low-variance estimate of the quantities. Coupled

[2] We note that while connectedness is required for the specific objective functions considered in this chapter, it is not required by either the GC-MDP formulation or the learning mechanism itself. The approach is applicable for other quantifiers of robustness (e.g., those that evaluate the size of the Largest Connected Component), as well as fundamentally different objective functions that measure graph properties of interest.

with the number of possible topologies, the high cost renders even shallow search methods infeasible in this domain.

3.2.2 Learning to Build Graphs with Function Approximation

While the problem formulation described in Section 3.2.1 may allow us to work with a tabular RL method, the number of states quickly becomes intractable – for example, there are approximately 10^{57} labelled, connected graphs with 20 vertices [265]. Thus, we require a means of considering graph properties that are label-agnostic, permutation-invariant, and generalise across similar states and actions. GNN architectures address these requirements. In particular, we use the S2V architecture[3] [85] as described in Section 2.4.3. Recall that, given an input graph $G_t = (V, E_t)$ in which nodes $v_i \in V$ have feature vectors $\mathbf{x}_{v_i}$, its objective is to produce for each node v_i an embedding vector $\mathbf{h}_{v_i}$. For each round or *layer* $l \in \{1, 2, ..., L\}$, the network simultaneously applies updates of the form:

$$\mathbf{h}_{v_i}^{(l+1)} = \text{ReLU}\left(\mathbf{W}_1 \mathbf{x}_{v_i} + \mathbf{W}_2 \sum_{v_j \in \mathcal{N}(v_i)} \mathbf{h}_{v_j}^{(l)}\right) \tag{3.8}$$

where $\mathcal{N}(v_i)$ is the open neighbourhood of node v_i. We initialise embeddings with $\mathbf{h}_{v_i}^{(0)} = \mathbf{0}\ \forall v_i \in V$, and let $\mathbf{h}_{v_i} = \mathbf{h}_{v_i}^{(L)}$ as a shorthand to indicate the embeddings after the execution of all the message passing rounds. Once node-level embeddings are obtained, a permutation-invariant representation for the state S_t can be derived by summing the node embeddings: $h(S_t) = \sum_{v_i \in V} \mathbf{h}_{v_i}$. The initial node features $\mathbf{x}_{v_i}$ are one-hot 2-dimensional vectors indicating whether v_i is the edge stub or not.

Recall, as we described in Section 2.5.3, that in Q-learning [352], the agent estimates the action-value function $Q(s, a)$. Subsequently, it derives a deterministic policy that acts greedily with respect to it. The agent interacts with the environment and updates its estimates according to:[4]

$$Q(s, a) \leftarrow Q(s, a) + \alpha\big(r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s', a') - Q(s, a)\big) \tag{3.9}$$

[3]The problem formulation does not depend on the specific GNN or RL algorithm used. While further advances developed by the community in these areas [240, 166] can be incorporated, in this chapter we focus on aspects specific to the challenges of optimising the global properties of graphs.

[4]Note that we deliberately use s and a in this equation instead of S_t and A_t since, when using a replay buffer, the time at which a given state s is observed is not necessarily equal to the time at which an update is performed.

During learning, exploratory random actions are taken with probability ϵ. In the case of high-dimensional state and action spaces, approaches that use a neural network to estimate $Q(s, a)$ have been successful in a variety of domains ranging from general game-playing to continuous control [251, 236]. Specifically, we use the DQN algorithm together with two parametrisations of the Q-function depending on whether the state contains a non-empty edge stub:

$$Q(s,a) = \mathbf{W}_3 \text{ ReLU } (\mathbf{W}_4 [\mathbf{h}_{v_a} \| h(s)]) \tag{3.10}$$

or

$$Q(s,a) = \mathbf{W}_5 \text{ ReLU } (\mathbf{W}_6 [\mathbf{h}_{\sigma} \| \mathbf{h}_{v_a} \| h(s)]), \tag{3.11}$$

where $[\cdot\|\cdot]$ represents concatenation. This lets the model learn *combinations* of relevant node features (e.g., that connecting two central nodes has high Q-value). The use of GNNs has several advantages: firstly, the parameters $\Theta = \{\mathbf{W}_i\}_{i=1}^{6}$ can be learned in a goal-directed fashion for the RL objective, allowing for flexibility in the representation. Secondly, the embeddings can generalise to larger graphs since they control how to combine neighbour features in the message passing rounds and are not restricted to graphs of a particular size. We note that the underlying S2V parameters $\mathbf{W}_1, \mathbf{W}_2$ are shared between the two Q-function parametrisations.

3.3 Evaluation Protocol

Learning Environment. We build a learning environment that allows for the definition of an arbitrary graph objective function $\mathcal{F}$ and provides a standardised interface for agents. Our implementation of the environment, RNet–DQN and baseline agents, and experimental suite is provided as a code repository containing Docker image blueprints that enable the reproduction of the results presented herein (up to hardware differences), including the relevant tables and figures. The instructions about how to obtain, configure, and run the code are provided in Appendix A.1.

Baselines. We compare against the following approaches:

- *Random*: This strategy randomly selects an available action.
- *Greedy*: This strategy uses lookahead and selects the action that gives the

biggest improvement in the estimated value of $\mathcal{F}$ over one edge addition.

- *Preferential*: Previous works have considered preferential additions between nodes with the two lowest degrees [35], connecting a node with the lowest degree to a random node [348] or connecting the two nodes with the Lowest Degree Product (LDP) [350], i.e., adding an edge between the vertices v_i, v_j that satisfy $\operatorname{argmin}_{i,j\ .\ i\neq j} \deg(v_i) \cdot \deg(v_j)$. We use the latter as we found it works best in all settings tested.

- *Fiedler Vector* (*FV*): The concept of Fiedler Vector was introduced by [122] and for robustness improvement by [348]. This strategy adds an edge between the vertices v_i, v_j that satisfy $\operatorname{argmax}_{i,j\ .\ i\neq j} |y_i - y_j|$, where $\mathbf{y}$ is the Fiedler Vector, i.e., the eigenvector of the graph Laplacian $\mathbf{L}$ corresponding to the second smallest eigenvalue, and y_i denotes the i-th element of vector $\mathbf{y}$.

- *Effective Graph Resistance* (*ERes*): The concept of Effective Graph Resistance was introduced by [113] and for robustness improvement as a local pairwise approximation by [350]. This strategy selects vertices v_i, v_j that satisfy $\operatorname{argmax}_{i,j\ .\ i\neq j} \Omega_{i,j}$, which is defined as $(\tilde{\mathbf{L}}^{-1})_{i,i} + (\tilde{\mathbf{L}}^{-1})_{j,j} - 2(\tilde{\mathbf{L}}^{-1})_{i,j}$, where $\tilde{\mathbf{L}}^{-1}$ is the pseudoinverse of $\mathbf{L}$.

- *Supervised Learning* (*SL*): We consider a Supervised Learning baseline by regressing on $\mathcal{F}$ to learn an approximate $\hat{\mathcal{F}}$. We use the same S2V architecture as RNet–DQN, which we train using MSE loss on the ground truth values of $\mathcal{F}$ values instead of the Q-learning loss. To choose actions for a graph G, the agent considers all graphs G'' that can be obtained by adding a single edge to G, selecting the one that satisfies $\operatorname{argmax}_{G''} \hat{\mathcal{F}}(G'')$.

Neural Network Architecture. For all experiments, we use an S2V embedding vector of length 64. The neural network architecture used for RNet–DQN and SL is formed of state-action embeddings obtained using S2V followed by an MLP; the single output unit corresponds to the $Q(s, a)$ estimate for RNet–DQN and the predicted $\hat{\mathcal{F}}$ for SL respectively. Details of hyperparameters used for the two learning-based models are provided in Appendix A.

Evaluation Protocol. We evaluate RNet–DQN and baselines on both synthetic and real-world graphs. We allow agents a number of edge additions b equivalent to

a percentage η of total possible edges. As an evaluation metric, we report the cumulative reward obtained by the agents, which quantifies the improvement in the objective function value between the final and original graphs. Training is performed separately for each graph family, objective function $\mathcal{F}$, and value of η. Where an agent is non-deterministic (either through intrinsic stochasticity or need for training), we repeat its evaluation (and training where applicable, starting from a different random initialisation of the network weights) to compute confidence intervals. For the learned models, we record both average and maximum performance. No hyperparameter tuning is performed due to computational budget constraints. Details about the experimental settings are provided in Appendix A.3.

Synthetic Graphs. We consider graphs generated through the following models:

- *Erdős–Rényi* (*ER*): A graph sampled uniformly out of $\mathcal{G}^{(N,m_{\text{ER}})}$ [114]. We use $m_{\text{ER}} = \frac{20}{100} \cdot \frac{N(N-1)}{2}$, which represents 20% of all possible edges.
- *Barabási–Albert* (*BA*): A growth model where N nodes each attach preferentially to m_{AB} existing nodes [19]. We use $m_{\text{AB}} = 2$.

We consider graphs with $|V| = 20$, allowing agents to add a percentage of the total number of edges equal to $\eta \in \{1, 2, 5\}$, which yields $b \in \{2, 5, 10\}$. For RNet–DQN and SL, we train on a disjoint set of graphs $\mathcal{G}^{\text{train}}$. We periodically measure performance on another set $\mathcal{G}^{\text{validate}}$, storing the best model found. We use $|\mathcal{G}^{\text{train}}| = 10^4$ and $|\mathcal{G}^{\text{validate}}| = 10^2$. The performance of all agents is evaluated on a set $\mathcal{G}^{\text{test}}$ with $|\mathcal{G}^{\text{test}}| = 10^2$ generated using the ER and BA models. In order to evaluate out-of-distribution generalisation, we repeat the evaluation on graphs with up to $|V| = 100$ (only up to $|V| = 50$ for Greedy and SL due to computational cost, see next section) and scale m_{ER} (for ER) and b accordingly. For non-deterministic agents, evaluation (and training, where applicable) is repeated across 50 random seeds.

Real-World Graphs. In order to evaluate our approach on real-world graphs, we consider infrastructure networks (for which robustness is a critical property) extracted from two datasets: *Euroroad* (road connections in mainland Europe and parts of Western and Central Asia [382, 218], $|V| = 1174$) and *Scigrid* (a dataset of the European power grid [247], $|V| = 1479$). We split these graphs by the country in which the nodes are located, selecting the Largest Connected Component in case they

Objective	$\mathcal{G}$	b	Random	LDP	FV	ERes	Greedy	SL		RNet–DQN	
								avg	*best*	*avg*	*best*
$\mathcal{F}_{random}$	BA	2	$0.018_{\pm 0.001}$	0.036	0.051	0.053	0.033	$0.048_{\pm 0.002}$	**0.057**	$0.051_{\pm 0.001}$	**0.057**
		5	$0.049_{\pm 0.002}$	0.089	0.098	0.106	0.079	$0.099_{\pm 0.003}$	0.122	$0.124_{\pm 0.001}$	**0.130**
		10	$0.100_{\pm 0.003}$	0.158	0.176	0.180	0.141	$0.161_{\pm 0.008}$	0.203	$0.211_{\pm 0.001}$	**0.222**
	ER	2	$0.029_{\pm 0.001}$	0.100	0.103	0.103	0.082	$0.094_{\pm 0.001}$	0.100	$0.098_{\pm 0.001}$	**0.104**
		5	$0.071_{\pm 0.002}$	0.168	0.172	**0.175**	0.138	$0.158_{\pm 0.002}$	0.168	$0.164_{\pm 0.001}$	0.173
		10	$0.138_{\pm 0.002}$	0.238	0.252	**0.253**	0.217	$0.221_{\pm 0.005}$	0.238	$0.240_{\pm 0.001}$	0.249
$\mathcal{F}_{targeted}$	BA	2	$0.010_{\pm 0.001}$	0.022	0.018	0.018	0.045	$0.022_{\pm 0.002}$	0.033	$0.042_{\pm 0.001}$	**0.047**
		5	$0.025_{\pm 0.001}$	0.091	0.037	0.077	0.077	$0.055_{\pm 0.003}$	0.077	$0.108_{\pm 0.001}$	**0.117**
		10	$0.054_{\pm 0.003}$	0.246	0.148	0.232	0.116	$0.128_{\pm 0.014}$	0.217	$0.272_{\pm 0.002}$	**0.289**
	ER	2	$0.020_{\pm 0.002}$	0.103	0.090	0.098	**0.149**	$0.102_{\pm 0.002}$	0.118	$0.122_{\pm 0.001}$	0.128
		5	$0.050_{\pm 0.002}$	0.205	0.166	0.215	**0.293**	$0.182_{\pm 0.008}$	0.238	$0.268_{\pm 0.001}$	0.279
		10	$0.098_{\pm 0.003}$	0.306	0.274	0.299	0.477	$0.269_{\pm 0.016}$	0.374	$0.461_{\pm 0.003}$	**0.482**

Table 3.1: Mean cumulative reward per episode obtained by the agents on synthetic graphs with $|V| = 20$, grouped by objective function, graph family, and number of edge additions b. Each reported value represents the improvement in the expected Critical Fraction between the final and initial graphs.

are disconnected. We then select those with $20 \leq |V| \leq 50$, obtaining 6 infrastructure graphs for Scigrid and 8 for Euroroad. The partitioning and selection procedure yields infrastructure graphs for the following countries:

- *Euroroad*: Finland, France, Kazakhstan, Poland, Romania, Russia, Turkey, Ukraine.
- *Scigrid*: Switzerland, Czech Republic, United Kingdom, Hungary, Ireland, Sweden.

Since, in this context, the performance on individual instances matters more than generalisability, we train and evaluate on each graph separately (effectively, the sets $\mathcal{G}^{\text{train}}, \mathcal{G}^{\text{validate}}, \mathcal{G}^{\text{test}}$ all have cardinality 1 and contain the same graph). *SL* is excluded for this experiment since we consider a single network. Evaluation is repeated across 10 random seeds.

3.4 Evaluation Results

In Table 3.1, we present the results of our experimental evaluation for synthetic graphs. We also display the evolution of the validation loss during training in Figure 3.2. Out-of-distribution generalisation results are shown in Figure 3.3. The results for real-world graphs are provided in Table 3.2. Additionally, Figure 3.4 displays examples of the original and improved topologies found by our approach.

Main Findings. We summarise our findings as follows:

Objective	Dataset	Instance	Random	LDP	FV	ERes	Greedy	RNet–DQN *avg*	RNet–DQN *best*
$\mathcal{F}_{random}$	Euroroad	Finland	$0.080_{\pm 0.019}$	0.133	0.163	0.170	0.162	$0.161_{\pm 0.015}$	**0.189**
		France	$0.057_{\pm 0.016}$	0.149	0.181	0.163	0.151	$0.178_{\pm 0.013}$	**0.202**
		Kazakhstan	$0.107_{\pm 0.018}$	0.165	0.191	0.180	0.160	$0.179_{\pm 0.010}$	**0.203**
		Poland	$0.082_{\pm 0.033}$	0.186	0.170	0.201	0.140	$0.196_{\pm 0.014}$	**0.230**
		Romania	$0.076_{\pm 0.025}$	0.196	0.170	**0.243**	0.203	$0.207_{\pm 0.013}$	0.235
		Russia	$0.084_{\pm 0.016}$	0.135	0.224	0.157	0.187	$0.199_{\pm 0.017}$	**0.230**
		Turkey	$0.092_{\pm 0.023}$	0.191	0.198	0.198	0.191	$0.215_{\pm 0.011}$	**0.247**
		Ukraine	$0.071_{\pm 0.017}$	0.158	0.186	0.151	0.098	$0.163_{\pm 0.022}$	**0.205**
	Scigrid	Switzerland	$0.050_{\pm 0.035}$	0.191	0.160	0.174	0.182	$0.198_{\pm 0.017}$	**0.226**
		Czech Republic	$0.091_{\pm 0.020}$	0.242	0.239	0.252	0.214	$0.334_{\pm 0.020}$	**0.375**
		United Kingdom	$0.111_{\pm 0.020}$	0.263	0.273	0.290	0.224	$0.321_{\pm 0.022}$	**0.379**
		Hungary	$0.051_{\pm 0.029}$	0.176	0.179	0.175	0.117	$0.148_{\pm 0.017}$	**0.185**
		Ireland	$0.090_{\pm 0.014}$	0.208	0.211	0.213	0.177	$0.201_{\pm 0.013}$	**0.228**
		Sweden	$0.097_{\pm 0.029}$	0.187	0.213	0.195	0.197	$0.213_{\pm 0.022}$	**0.276**
$\mathcal{F}_{targeted}$	Euroroad	Finland	$0.069_{\pm 0.018}$	0.149	0.112	0.112	**0.307**	$0.273_{\pm 0.009}$	0.300
		France	$0.032_{\pm 0.019}$	0.199	0.120	0.120	0.074	$0.218_{\pm 0.006}$	**0.228**
		Kazakhstan	$0.052_{\pm 0.021}$	0.161	0.137	0.124	0.229	$0.236_{\pm 0.014}$	**0.257**
		Poland	$0.010_{\pm 0.008}$	0.101	0.114	0.084	0.108	$0.230_{\pm 0.008}$	**0.248**
		Romania	$0.029_{\pm 0.021}$	0.167	0.056	0.126	0.148	$0.238_{\pm 0.021}$	**0.270**
		Russia	$0.000_{\pm 0.000}$	0.000	0.000	0.053	0.000	$0.110_{\pm 0.036}$	**0.155**
		Turkey	$0.044_{\pm 0.021}$	0.126	0.155	0.126	0.143	$0.233_{\pm 0.018}$	**0.264**
		Ukraine	$0.031_{\pm 0.023}$	0.074	0.037	0.083	0.135	$0.164_{\pm 0.006}$	**0.178**
	Scigrid	Switzerland	$0.030_{\pm 0.024}$	0.000	0.103	0.098	0.045	$0.128_{\pm 0.006}$	**0.139**
		Czech Republic	$0.038_{\pm 0.026}$	0.116	0.116	0.116	0.163	$0.242_{\pm 0.027}$	**0.284**
		United Kingdom	$0.070_{\pm 0.047}$	0.190	0.095	0.184	0.207	$0.252_{\pm 0.027}$	**0.326**
		Hungary	$0.027_{\pm 0.028}$	**0.190**	0.000	0.129	0.143	$\mathbf{0.190}_{\pm 0.000}$	**0.190**
		Ireland	$0.047_{\pm 0.023}$	0.101	0.084	0.106	0.079	$0.259_{\pm 0.011}$	**0.288**
		Sweden	$0.061_{\pm 0.021}$	0.142	0.121	0.201	0.094	$0.232_{\pm 0.008}$	**0.261**

Table 3.2: Results obtained on real-world graphs, split by graph instance. Each reported value represents the improvement in the expected Critical Fraction between the final and initial graphs.

RNet–DQN provides competitive performance, especially for longer action sequences. Across all settings tested, RNet–DQN performed significantly better than random. On synthetic graphs, the best model obtained the highest performance in 8 out of 12 settings tested, while the average performance is at least 89% of that of the best-performing configuration. For BA graphs, RNet–DQN obtained the best performance across all tasks tested. For ER graphs, ERes performed slightly better when considering $\mathcal{F}_{random}$; for $\mathcal{F}_{targeted}$, the greedy baseline performed better on shorter sequences. For real-world graphs, RNet–DQN obtained the best performance across all tasks.

Strategies for improving $\mathcal{F}_{random}$ are easier to learn. The performance gap between the trained model and the baselines is smaller for $\mathcal{F}_{random}$, suggesting it is less complex to learn. This is also supported by the evaluation losses monitored during training, which show performance improves and plateaus more quickly. For $\mathcal{F}_{random}$ the network with randomly initialised parameters already yields policies with satisfactory results, and training brings a small improvement. In contrast, the improvements for $\mathcal{F}_{targeted}$ are much more dramatic.

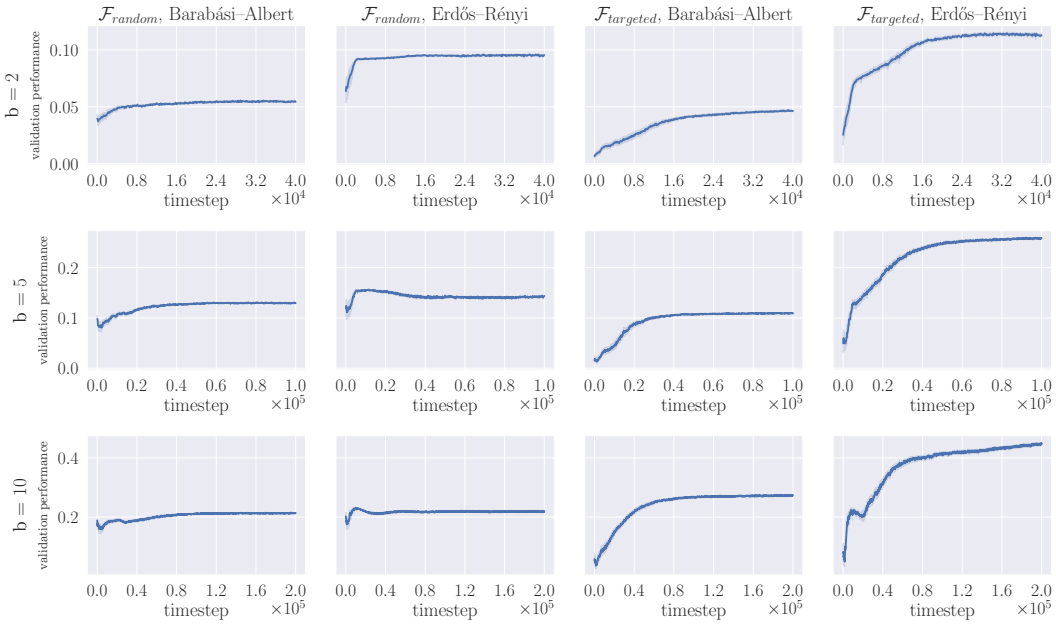


Figure 3.2: Performance on $\mathcal{G}^{\text{validate}}$ for synthetic graphs as a function of training steps. Note the different x-axes scales for each row: more training steps are typically required for longer edge addition sequences.

Out-of-distribution generalisation only occurs for $\mathcal{F}_{random}$. The performance on larger out-of-distribution graphs is preserved for the $\mathcal{F}_{random}$ objective, and especially for BA graphs we observe strong generalisation. The performance for $\mathcal{F}_{targeted}$ decays rapidly, obtaining worse performance than the baselines as the size increases. The poor performance of the greedy policy means the $Q(s, a)$ estimates are no longer accurate under distribution shift. There are several possible explanations, e.g., the inherent noise of estimating $\mathcal{F}_{random}$ makes the neural network more robust to outliers, or that central nodes impact message passing in larger graphs differently. We think investigating this phenomenon is a worthwhile direction for future work, since out-of-distribution generalisation does occur for $\mathcal{F}_{random}$ and evaluating the objective functions directly during training is prohibitively expensive for large graphs.

Performance on real-world graphs is comparatively better with respect to the baselines. This is expected since training is performed separately for each graph to be optimised.

Time Complexity. Below, we compare the time complexities of the methods.

- *RNet–DQN*: $\mathcal{O}(|V|+|E_t|)$ operations at each step: constructing node and graph-level embeddings and, based on these embeddings, performing the forward pass in the neural network to estimate $Q(s, a)$ for all valid actions.

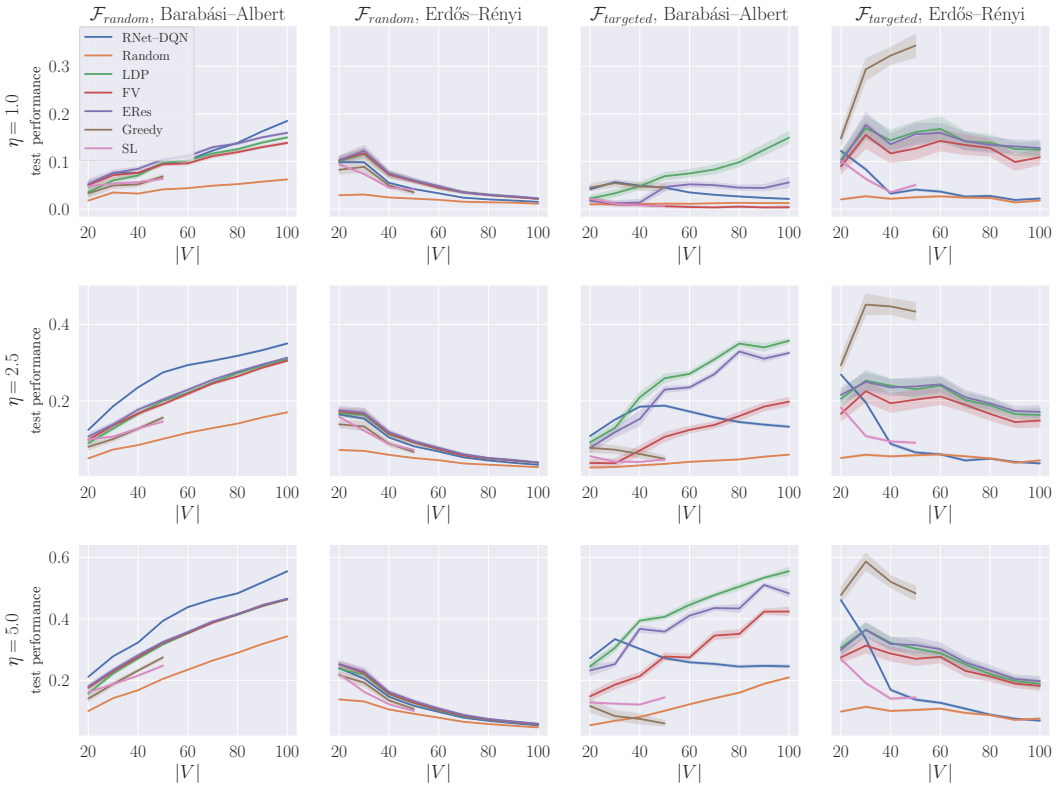


Figure 3.3: Performance on out-of-distribution synthetic graphs as a function of graph size, grouped by target problem and percentage of edge additions η. For RNet–DQN and SL, models trained on graphs with $|V| = 20$ are used.

- *Random*: $\mathcal{O}(1)$ for sampling, assuming the environment checks action validity.
- *Greedy*: $\mathcal{O}(|V|^4 \cdot (|V| + |E_t|))$. The improvement in $\mathcal{F}$ is estimated for all $\mathcal{O}(|V|^2)$ candidate edges. For each edge, this involves $\mathcal{O}(|V|)$ MC simulations, each of which has complexity $\mathcal{O}(|V| \cdot (|V| + |E_t|))$ as described in Section 3.2.1.
- *LDP*: $\mathcal{O}(|V|^2)$ for computing the product of node degrees.
- *FV, ERes*: $\mathcal{O}(|V|^3)$, since they involve computing the eigendecomposition and the Moore-Penrose pseudoinverse of the graph Laplacian respectively.
- *SL*: $\mathcal{O}(|V|^2 \cdot (|V| + |E_t|))$, since $\hat{\mathcal{F}}$ is predicted for the $\mathcal{O}(|V|^2)$ graphs that can be obtained by adding a single edge to G.

It is worth noting that the analysis above does not account for the cost of training, the complexity of which is difficult to determine as it depends on many hyperparameters and the specific characteristics of the problem at hand. The approach is thus advantageous in situations in which predictions need to be made quickly, over many

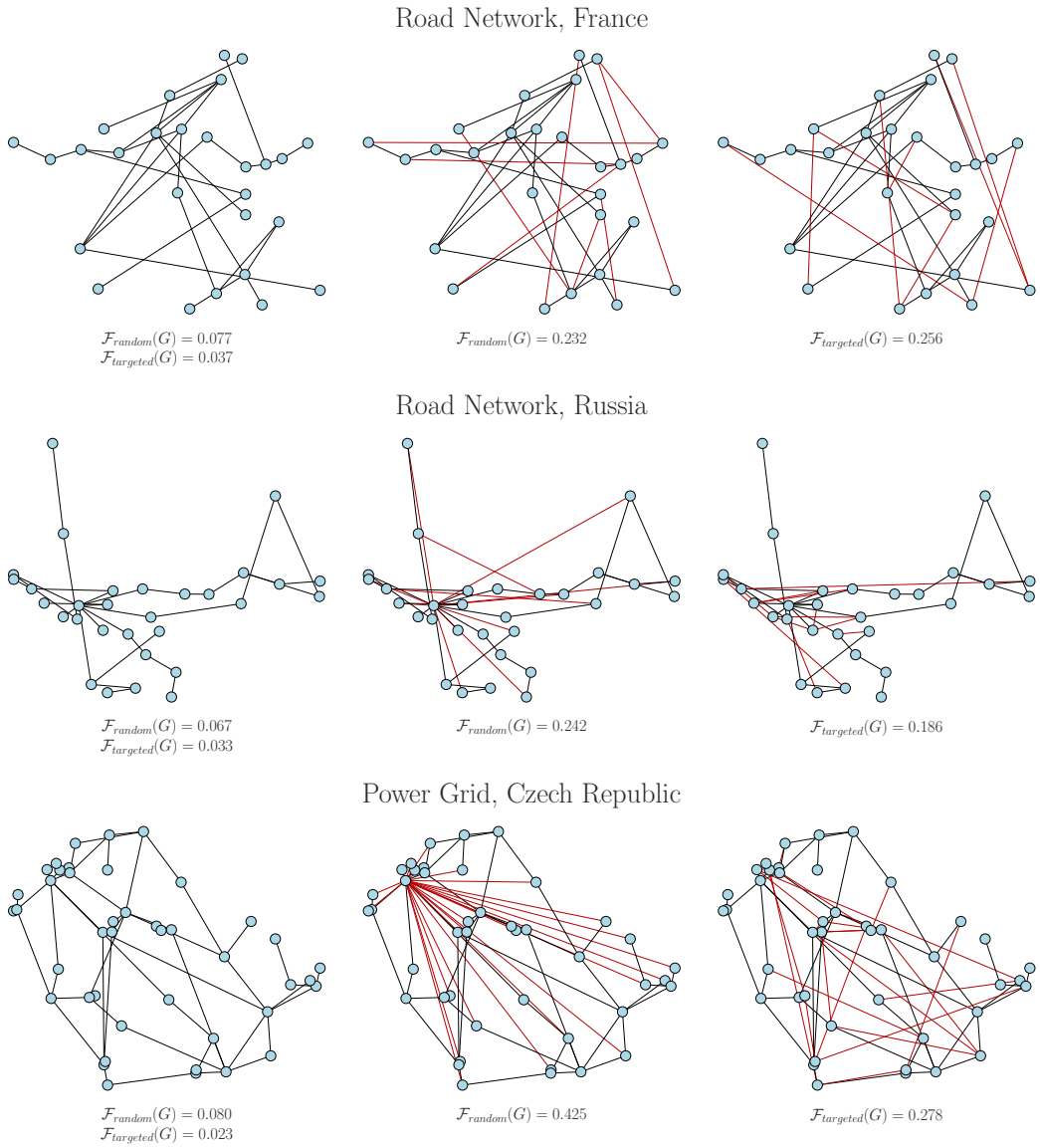


Figure 3.4: Several examples of the solutions found by RNet–DQN on real-world graphs. Each row of the illustration shows the original network on the left, while the central and right panels show the network optimised for resilience to random and targeted removals, respectively. Objective function values are shown underneath. The solutions for $\mathcal{F}_{random}$ typically assign more connections to a few central nodes, notably discovering the hub pattern in the third example. For $\mathcal{F}_{targeted}$ the added edges are spread around the network, reducing the impact of attacks. However, the algorithm might discover more complex patterns that are not directly interpretable, as shown in the solutions for the first example network.

graphs, or the model transfers well from a cheaper training regime. We also remark that, even though the method requires an upfront cost for training, this can be seen as a constant term if the number of problem instances over which we would like to obtain predictions is large. These characteristics are shared with other emergent work that tackles combinatorial optimisation with ML [203, 31].

3.5 Discussion

Relationship to RL–S2V. Our work builds on RL–S2V, a method that was applied for the construction of adversarial examples against graph classifiers [86]. However, it is worth noting that there are a series of key differences with respect to this approach. Firstly, RL–S2V is not designed to address the problem of constructing robust graphs or, more generally, learning to construct graphs according to a given goal. Secondly, there are two key algorithmic differences to RL–S2V with respect to the GC-MDP formulation: the reward function used, which in this case quantifies a global property of the graph itself, as well as the definition of the action spaces and the transition model, which account for excluding already-existing edges (RL–S2V ignores this, leading to some of the edge budget being wasted). Since the values of structural properties we consider are increasing in the number of edges (the complete graph has robustness 1), RNet–DQN generally yields strictly better performance results.

Extensions. The proposed approach can be applied to other problems based on different definitions of robustness or considering fundamentally different objective functions such as efficiency [223], path diversity [154], and assortativity [260], which are of interest in various biological, communication, and social networks. Since our formulation and algorithm are objective-agnostic, we expect they are applicable out-of-the-box for other objectives, even those for which no strong baselines are currently known. As such, this approach may be a useful tool for the discovery of new graph improvement algorithms for objectives that can be evaluated programmatically, either in closed form or via simulations. Potential limitations might be related to the complexity of the objective functions and the related computational demands. We also view the interpretability of the approach, which is less straightforward than those based on known mathematical concepts such as Fiedler Vector, to be an important research direction. Since there is an active interest in the interpretability of both GNNs and RL [371, 345], we consider that there is scope for developing techniques that are tailor-made for explaining policies learned by RL on graphs.

Operationalisation. Beyond considering other objectives, in order to operationalise the proposed algorithm, it is possible to integrate a variety of refinements, which can include capturing heterogeneous edge costs (e.g., different capacities per link in a communication network), extending the action space to support heterogeneous edge

types (e.g., addition of different types of edges with specific characteristics), and integrating domain-specific link constraints (e.g., planarity). For critical scenarios, it is also possible to verify that the resulting solutions satisfy some given formal properties and constraints [133]. Furthermore, considering multi-criteria objective functions [290] is important for cases where properties of the solutions must be balanced [199]. Various choices exist for representing this trade-off and should be captured on a case-by-case basis depending on the application: for example, a linear combination may be sufficient in certain situations, while others are characterised by economies of scale. We also remark that our formulation captures operational scenarios in which the cost of constructing a link is significantly greater than the cost of its maintenance (e.g., as with road networks).

Broader Applications. The approach described in this chapter can be used to improve the properties of a variety of human-made infrastructure systems, such as communication networks, transportation networks, and power grids. We also envisage potential applications in biological networks (e.g., hypothesis testing for understanding the characteristics of brain networks [53]), ecological networks (e.g., design of more resilient ecosystems [357]) and social networks (e.g., design of organisational structures [363]). As far as biological networks are concerned, the brain is hypothesised to optimise a trade-off between efficiency and wiring cost [53]. Our method could be used in order to test different hypotheses related to the resulting structure of brain networks over evolutionary times and also during their development. Indeed, the evolution of such networks in time has been captured (for example, Sulston et al. [325] mapped the development of the C. elegans connectome). This can be achieved by applying the optimisation procedure for different objective functions and comparing the obtained networks to the "ground truth". With respect to the potential application in ecosystem management, the graph formalisation can be used to model interactions between species in a given environment. As such, our method has potential applications to study, in simulation, the impact of introducing or removing species from an ecosystem so as to achieve a desired outcome. Specifically, in the context of robustness, we can consider optimising the resilience of an ecosystem to intrinsic or extrinsic shocks, a task of fundamental importance [9]. The dynamics of interactions between species may be modelled in simulation using well known

models of e.g., predator-prey mechanics [32]. With respect to social networks, for example, our method can be applied to derive optimal communication strategies and related team structures so as to optimise a given objective for an organisation. Finally, there are also potential applications for networks of artificial agents (i.e., robots). There is a significant body of work in the robotics literature that treats the problem of maintaining robust communication in a network of agents working together to complete a task in an environment that contains obstacles or adversaries. For instance, [323] uses properties of the graph Laplacian (namely, the Fiedler Vector and its associated eigenvalue) to ensure the underlying communication network remains robust. Since we have empirically shown superior performance to using the Fiedler Vector, our approach could also lead to gains in this deployment scenario.

3.6 Summary

In this chapter, we have addressed the problem of improving a given graph structure with the goal of maximising the value of a global objective function. We have framed it for the first time as a decision-making problem and we have formalised it as the Graph Construction MDP (GC-MDP). Our approach, developed to tackle ***RQ1***, uses RL and GNNs as key components for generalisation. As a case study, we have considered the problem of improving graph robustness to random and targeted removals of nodes. Our experimental evaluation on synthetic and real-world graphs shows that, in certain situations, this approach can deliver performance superior to existing methods, both in terms of the solutions found (i.e., the resulting robustness of the graphs) and time complexity of model evaluation (***RQ2***). Furthermore, we have shown the ability to transfer to out-of-sample graphs, as well as the potential to transfer to out-of-distribution graphs larger than those used during training as a possible technique to address ***RQ3***.

Chapter 4

Planning Spatial Networks with Monte Carlo Tree Search

In this chapter, we continue our treatment of optimising graph structure through edge additions. Unlike the method in Chapter 3, which only considers topological properties, herein we propose a more realistic model for networks positioned in physical space. Namely, it is able to capture the influence of spatial characteristics on the existence and density of links. Through the use of planning methods, we also make contributions towards improving the scalability of decision-making algorithms for graph construction by sidestepping the costly trial-and-error required for training a RL agent.

Generic planning algorithms, such as Monte Carlo Tree Search, are directly applicable given our deterministic MDP formulation but, nevertheless, may be sub-optimal. We tailor the algorithm towards this family of problems, addressing three key aspects: their single-agent nature, the rapidly growing size of the action space, and the relationship between the cost of links and their contribution to the objective. The proposed approach, SG-UCT, is rigorously evaluated for increasing the efficiency and attack resilience of synthetic spatial graphs, as well as real-world networks of Internet Service Providers and urban metro systems.

4.1 Introduction

The non-Euclidean structure of graphs has traditionally proven challenging for ML and decision-making approaches. As discussed extensively in Chapter 2, the

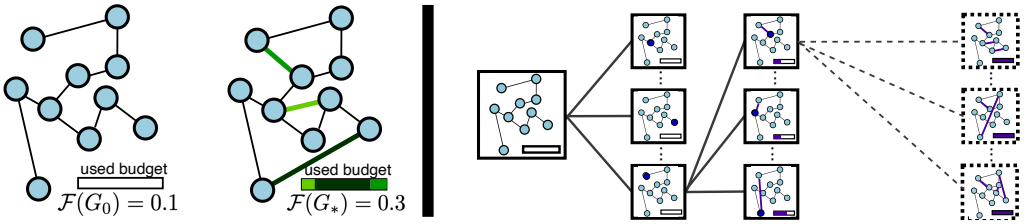


Figure 4.1: Schematic of our approach. Left: given a spatial graph G_0, an objective function $\mathcal{F}$, and a budget defined in terms of edge lengths, the goal is to add a set of edges such that the resulting graph G_* maximally increases $\mathcal{F}$. Right: we formulate this problem as a deterministic MDP, in which states are graphs, actions represent the selection of a node, transitions add an edge every two steps, and the reward is based on $\mathcal{F}$. We use Monte Carlo Tree Search to plan *the optimal set of edges to be added* using knowledge of this MDP, and propose a method (SG-UCT) that improves on standard UCT.

emergence of the GNN learning paradigm [303] and geometric deep learning more broadly [50] have brought about encouraging breakthroughs in diverse application areas for graph-structured data. Relevant examples include combinatorial optimisation [346, 29, 203], recommendation systems [254, 369] and computational chemistry [144, 190, 372, 45].

There is an increasing interest in the problem of goal-directed graph construction, in which the aim is to build or to modify the topology of a graph (i.e., add a set of edges) so as to maximise the value of a global objective function, subject to a budget constraint. As an example scenario, consider the following: given a road network and a budget of 200 kilometres of highways that can be invested, a national transportation department must decide where to place them with the goal of minimising average trip time for drivers. Another practical instance of this problem is to place 500 kilometres of tracks between stations in a train network such that, in the event of infrastructure failures or disruptions, customers have many alternative routes to their destinations. Similar network planning and design scenarios arise in a variety of other infrastructure systems including communication, power, and water distribution networks. The family of related problems unified by this framework is illustrated at a high level in the first panel of Figure 4.1 and is mathematically defined in Equation (4.1).

As this task involves an element of exploration (optimal solutions are not known *a priori*), its formulation as a decision-making process is a suitable paradigm. In Chapter 3, we formulated the optimisation of a global structural graph property as an

MDP and approached it using a variant of the RL-S2V [86] algorithm, showing that generalisable strategies for improving a global network objective can be learned, and can obtain performance superior to prior conventional approaches [35, 306, 348, 350] for certain classes of problems.

However, several real-world networks are embedded in space and this fact adds constraints and trade-offs in terms of the topologies that can be created [137, 23]. In fact, since there is a cost associated with edge length, connections tend to be local, and long-range connections must be justified by some gain (e.g., providing connectivity to a hub). Moreover, existing methods based on RL are challenging to scale, due to the sample complexity of current training algorithms, the linear increase of possible actions with the number of nodes, and the complexity of evaluating the global objectives (typically polynomial in the number of nodes). Furthermore, objective functions defined over nodes' positions, such as efficiency, are key for understanding their organisation [223], but current solutions only consider topological aspects of the problem. Additionally, training data (i.e., instances of real-world graphs) are scarce and we are typically interested in a specific starting graph (e.g., a particular infrastructure network to be improved).

In this chapter, for the first time among related works, we consider the problem of the construction of spatial graphs as a decision-making process that explicitly captures the influence of space on graph-level objectives, the realisability of links, and connection budgets. Furthermore, to address the scalability issue, we select an *optimal set of edges* to add to the graph through planning, which sidesteps the problem of sample complexity since we do not need to learn a generalisable strategy. We adopt the Monte Carlo Tree Search framework – specifically, the UCT algorithm [213] – and show it can successfully be applied in planning graph construction strategies. We illustrate our approach at a high level in Figures 4.1 and 4.2. Finally, we propose several improvements over the basic UCT method in the context of spatial networks. These relate to important characteristics of this family of problems: namely, their single-agent, deterministic nature; the inherent trade-off between the cost of edges and their contribution to the global objective; and an action space that is linear in the number of nodes in the network. Our proposed approach, Spatial Graph UCT (SG-UCT), is designed with these characteristics in mind and relies on a limited

set of assumptions. For this reason, it can be applied to a large class of networked systems positioned in physical space.

As objective functions, we consider the global properties of network efficiency and robustness to targeted attacks. While these represent a variety of practical scenarios, our approach is broadly applicable to any other structural property. We perform an evaluation on synthetic graphs generated by a spatial growth model and several real-world internet backbone networks and metro transportation systems. Our results show that SG-UCT performs best out of all methods that have been proposed in the past in all the settings we tested; moreover, the performance gain over UCT is substantial (24% on average and up to 54% over UCT on the largest networks tested in terms of a robustness metric). We also benchmark the execution time of the various approaches, showing that SG-UCT requires similar amounts of computation to other search-based methods. In addition, we conduct an ablation study that explores the impact of the individual algorithmic mechanisms, highlighting that the most significant part of the objective function gains is due to the simulation policy that prioritises lower cost edges.

4.2 Methods

In this section, we first introduce some preliminary notions regarding spatial networks and relevant objectives. We then formulate their construction towards optimising a global objective function as an MDP. Subsequently, we propose a variant of the UCT planning algorithm, termed SG-UCT, which exploits the characteristics of spatial networks.

4.2.1 Spatial Networks and Objectives

We define a *spatial network* as the tuple $G = (V, E, f, w)$. V is the set of vertices, and E is the set of edges. $f\colon V \to \Lambda$ is a function that maps nodes in the graph to a set of positions Λ. We require that Λ admits a metric d, i.e., there exists a function $d\colon \Lambda \times \Lambda \to \mathbb{R}^+$ defining a pairwise *distance* between elements in Λ. The tuple (Λ, d) defines a *space*, common examples of which include Euclidean space and spherical geometry. $w\colon E \to \mathbb{R}^+$ associates a *weight* with each edge, a positive real-valued number. For example, the weight can be interpreted as the capacity of a link in networks that carry traffic.

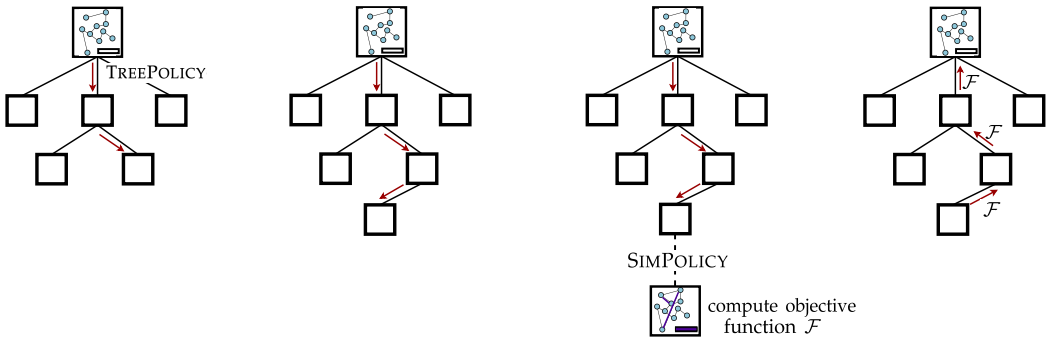


Figure 4.2: Illustration of Monte Carlo Tree Search applied to the construction of spatial networks. (i) Starting from the root node, which contains the original graph S_0, the tree is traversed until an expandable node is reached. (ii) The algorithm expands the tree by adding a child to this node. The strategy for traversing the tree and deciding which node to expand is called *tree policy*, and typically relies on statistics (such as average reward) stored by each node. (iii) From the newly added node, a trajectory is sampled using a *simulation policy* until a terminal state is reached and the algorithm cannot add more edges to the graph. A reward is received depending on the objective function $\mathcal{F}$ defined on the graph. (iv) The path is traversed back to the root node, updating the statistics of nodes along the way. After several iterations of steps (i)-(iv), the algorithm will select action A_0 corresponding to the child of the root with the highest average reward. The search then continues with the next state S_1 at the root.

We consider two global objectives for spatial networks that are representative of a wide class of properties relevant in real-world situations. Depending on the domain, there are many other global objectives for spatial networks that can be considered, to which the approach that we present is directly applicable.

Efficiency. Efficiency quantifies the optimality of information transmission in a network. It does not solely depend on topology but also on the *spatial* distances between the nodes. We adopt its definition as formalised by [223], and we indicate the global efficiency objective with $\mathcal{F}_E(G) = \frac{1}{N(N-1)} \sum_{i \neq j} \frac{1}{d_{\text{sp}}(v_i, v_j)}$, where $d_{\text{sp}}(v_i, v_j)$ is the cumulative length (i.e., the summed distances) of the shortest path between vertices v_i and v_j. To normalise, we divide by the ideal efficiency $\mathcal{F}_E^*(G) = \frac{1}{N(N-1)} \sum_{i \neq j} \frac{1}{d(f(v_i), f(v_j))}$, and possible values are thus in $[0, 1]$.

Efficiency is computable in $\mathcal{O}(|V|^3)$ by using the "weighted" version of the Floyd-Warshall shortest path algorithm[1], in which the spatial distances over the edges are given as weights (note, however, that this differs from the weight function w defined above, which is akin to capacity). The path lengths are provided raw and are not normalised by the straight line distances.

[1] In practice, this may be made faster by considering dynamic shortest path algorithms, e.g., [103].

Robustness. We consider the property of robustness, i.e., the resilience of the network in the face of removals of nodes. We adopt a robustness measure widely used in the literature [8, 56] and of practical interest and applicability based on the Largest Connected Component (LCC), i.e., the component with most nodes. In particular, we use the definition in [306], which considers the size of the LCC as nodes are removed from the network. We consider only the targeted attack case as previous work has found it is more challenging [8]. We define the robustness measure as $\mathcal{F}_R(G) = \mathbb{E}_\xi[\frac{1}{N}\sum_{i=1}^{N}\frac{|\text{LCC}(G,\xi,k)|}{N}]$, where $\text{LCC}(G,\xi,k)$ denotes the Largest Connected Component of G after the removal of the first k nodes in the permutation ξ (in which nodes appear in descending order of their degrees). Possible values are in $[\frac{1}{N}, 0.5)$. This quantity can be estimated using Monte Carlo simulations and scales as $\mathcal{O}(|V|^2 \cdot (|V| + |E|))$.

It is worth noting that the value of the objective functions is typically higher if more edges exist in the network (the complete graph has both the highest possible efficiency and robustness). However, it may be necessary to balance the contribution of an edge to the objective with its cost. The proposed method explicitly accounts for this trade-off, which is widely observed in infrastructure and brain networks [136, 53], amongst others.

4.2.2 Spatial Graph Construction as an MDP

Spatial Constraints in Network Construction. Spatial networks that can be observed in the real world typically incur a cost to edge creation. To aid intuition, consider the example of a power grid: the cost of a link depends, among another aspects, on its geographic distance as well as on its capacity. We let $c(e_{i,j})$ denote the cost of edge $e_{i,j}$ and $c(\Gamma) = \sum_{e\in\Gamma} c(e_{i,j})$ be the cost of a *set* of edges Γ. We consider $c(e_{i,j}) = w(e_{i,j}) \cdot d(f(v_i), f(v_j))$ to capture the notion that longer, higher capacity connections are more expensive. We are aware that this is just one of the possible definitions and different notions of cost may be desirable depending on the domain. However, it can be considered representative of several real-world scenarios, in which cost can be modelled in a similar manner. To ensure fair comparisons, we normalise costs $c(e_{i,j})$ to be in $[0, 1]$.

Problem Statement. Let $\mathcal{G}^{(N)}$ be the set of labelled, undirected, weighted, spatial networks with N nodes. We let $\mathcal{F}\colon \mathcal{G}^{(N)} \to [0, 1]$ be an objective function, and

$b_0 \in \mathbb{R}^+$ be a modification budget that represents an upper bound on the *summed lengths* of new edges that can be added. Note that, unlike in the previous chapter in which the budget was defined in terms of *number* of edges, herein we account for the positioning of the nodes in space. Hence, we may decide to build a higher number of short connections in case it is beneficial.

Given an initial graph $G_0 = (V, E_0, f, w) \in \mathcal{G}^{(N)}$, the aim is to add a set of edges Γ to G_0 such that the graph $G_* = (V, E_*, f, w)$ satisfies:

$$G_* = \underset{G' \in \mathcal{G}'}{\operatorname{argmax}} \mathcal{F}(G'), \tag{4.1}$$
$$\text{where} \quad \mathcal{G}' = \{G \in \mathcal{G}^{(N)} \mid E = E_0 \cup \Gamma \,.\, c(\Gamma) \leq b_0\}$$

MDP Formulation. As in the previous chapter, we formulate graph construction as a series of sequential decisions, which acts as a decomposition of the solution space. We also further decompose the edge addition into two separate node selection decisions, with the transition model adding an edge to the topology every two timesteps, while also imposing length restrictions on the new edges. When put together, these aspects yield a means of achieving scalability to larger graphs. We next define the MDP elements as below.

State: The state S_t is a 3-tuple $(G_t, \{\sigma_t\}, b_t)$ containing the spatial graph $G_t = (V, E_t, f, w)$, a singleton comprised of an *edge stub* σ_t, and the remaining budget b_t. σ_t can be either the empty set $\varnothing$ or the node $v_{k_{\text{from}}} \in V$. As previously, if the edge stub is non-empty, it means that the agent has "committed" in the previous step to creating an edge originating at the node $v_{k_{\text{from}}}$.

Action: An action A_t corresponds to the selection of the index of a node in V. We enforce spatial constraints as follows: given a node v_i, we define the set $\mathcal{K}(v_i)$ of *connectable* nodes v_j that represent realisable connections. We let:

$$\mathcal{K}(v_i) = \{v_j \in V \mid c(e_{i,j}) \leq \rho \max_{k \,.\, e_{i,k} \in E_0} c(e_{i,k})\} \tag{4.2}$$

which formalises the idea that a node can only connect as far as a proportion ρ of its longest existing connection, with $\mathcal{K}(v_i)$ fixed based on the initial graph G_0. Compared to the naïve choice of establishing a uniform connection radius for all the

nodes, this has the benefit of allowing long-range connections if they already exist in the network. Given an unspent connection budget b_t, we let the set $\mathcal{B}(v_i, b_t) = \{v_j \in \mathcal{K}(v_i) \mid c(e_{i,j}) \leq b_t\}$ consist of those connectable nodes whose cost is not more than the unspent budget. Therefore, available actions[2] are defined as:

$$\mathcal{A}(S_t) = \begin{cases} \{\text{select } k_{\text{from}} & \mid v_{k_{\text{from}}} \in V \wedge \deg(v_{k_{\text{from}}}) < |V| - 1 \\ & \wedge \ |\mathcal{B}(v_{k_{\text{from}}}, b_t)| > 0\}, \text{ if } t \bmod 2 = 0 \\ \{\text{select } k_{\text{to}} & \mid v_{k_{\text{to}}} \in V \wedge e_{k_{\text{from}},k_{\text{to}}} \notin E_t \wedge v_{k_{\text{to}}} \in \mathcal{B}(v_{k_{\text{from}}}, b_t)\}, \text{ otherwise.} \end{cases} \tag{4.3}$$

Transitions: The deterministic transition model adds an edge every two timesteps and decrements the budget accordingly. Concretely, we define it as $P(S_{t+1} = s' | S_t = s, A_t = a) = \delta_{ss'}$, where

$$s' = \begin{cases} ((V, E_t, f, w), \{v_a\}, b_t), \text{ if } t \bmod 2 = 0 \\ ((V, E_t \cup \{e_{k_{\text{from}},a}\}, f, w), \{\varnothing\}, b_t - c(e_{k_{\text{from}},a})), \text{ otherwise.} \end{cases} \tag{4.4}$$

Reward: The final reward R_T is defined as $\mathcal{F}(G_T) - \mathcal{F}(G_0)$ and intermediary rewards are 0. We do not provide intermediate rewards due to the large computational cost (at least cubic in the number of nodes) needed for calculating the objective functions. However, intermediate rewards may be provided for less computationally demanding objectives.

Episodes in this MDP proceed for an arbitrary number of steps until the budget is exhausted or no valid actions remain (concretely, $|\mathcal{A}(S_t)| = 0$). Since we are in the finite horizon case, we let $\gamma = 1$. Given the MDP definition above, the problem specified in Equation 4.1 can be reinterpreted as finding the trajectory τ_* that starts at $S_0 = (G_0, \{\varnothing\}, b_0)$ such that the final reward R_T is maximal – actions along this trajectory will define the set of edges Γ.

[2]Depending on the type of network being considered, in practice there may be different types of constraints on the connections that can be realised. For example, in transportation networks, there can be obstacles that make link creation impossible, such as prohibitive landforms or populated areas. In circuits and utility lines, planarity is a desirable characteristic as it makes circuit design cheaper. Such constraints can be captured by the definition of $\mathcal{K}(v_i)$ and enforced by the environment when providing the agent with available actions $\mathcal{A}(s)$. Conversely, defining $\mathcal{K}(v_i) = V \setminus \{v_i\}$ recovers the simplified case where no constraints are imposed.

4.2.3 Algorithm

The formulation above can, in principle, be used with any planning algorithm for MDPs in order to identify an optimal set of edges to add to the network. The UCT algorithm, discussed in Section 4.2.1, is one such algorithm that has proven very effective in a variety of settings. We refer the reader to [51] for an in-depth description of the algorithm and its various applications. However, the generic UCT algorithm assumes very little about the particulars of the problem under consideration, which, in the context of spatial network construction, may lead to sub-optimal solutions. In this section, we identify and address concerns specific to this family of problems, and formulate the Spatial Graph UCT (SG-UCT) variant of UCT in Algorithm 1. The evaluation whose results are presented in Section 4.4 compares SG-UCT to UCT and other baselines, and contains an ablation study of SG-UCT's components.

Best Trajectory Memoisation (BTM). The standard UCT algorithm is applicable in a variety of settings, including multi-agent, stochastic environments. For example, in two-player games, an agent needs to re-plan from the new state that is arrived at after the opponent executes its move. However, the single-agent, deterministic nature of the problem considered means that there is no-need to re-plan trajectories after a stochastic event: the agent can plan all its actions from the very beginning in a single step. We thus propose the following modification over UCT: memoising the trajectory with the highest reward found during the rollouts, and returning it at the end of the search. We name this Best Trajectory Memoisation, shortened BTM. This is similar in spirit (albeit much simpler) to ideas used in Reflexive and Nested MCTS for deterministic puzzles, in which the best move found at lower levels of a nested search is used to inform the upper level [59, 60].

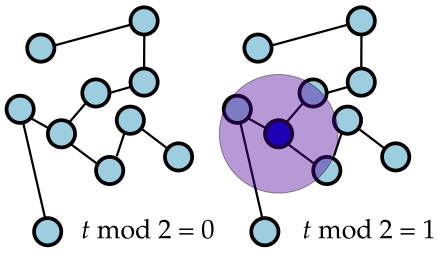


Figure 4.3: Illustration of the asymmetry in the number of actions at even (state has an empty edge stub) versus odd t (state contains a non-empty edge stub). Spatial constraints are imposed in the latter case, reducing the number of actions.

Cost-Sensitive Default Policy. The standard default policy used to perform out-of-tree actions in the UCT framework is based on random rollouts. While evaluating nodes using this approach is free from bias, rollouts can lead to high-variance estimates, which can hurt the performance of the search. Previous work has considered hand-crafted heuristics and learned policies as alternatives, although, perhaps counter-intuitively, learned policies may lead to worse results [139]. As initially discussed in Section 4.2.1, the value of the objective functions we consider grows with the number of edges of the graph. We thus propose the following default policy for spatial networks: sampling each edge with probability inversely proportional to its cost. Formally, we let the probability of edge $e_{i,j}$ being selected during rollouts be proportional to $(\max_{k,l \,.\, e_{k,l} \in E}(c(e_{k,l})) - c(e_{i,j}))^{\beta}$, where β denotes the level of bias. $\beta \to 0$ reduces to random choices, while $\beta \to \infty$ selects the minimum cost edge. This is very inexpensive computationally, as the edge selection probabilities only need to be calculated once, at the start of the search.

Action Space Reduction. In certain domains, the number of actions available to an agent is large, which can greatly affect scalability. Previous work in RL has considered decomposing actions into independent sub-actions [164], generalising across similar actions by embedding them in a continuous space [111], or learning which actions should be eliminated via supervision provided by the environment [376]. Existing approaches in planning consider progressively widening the search based on a heuristic [64] or learning a partial policy for eliminating actions in the tree [281].

Concretely, in this MDP, the action space grows linearly in the number of nodes. This is partly addressed by the imposed connectivity constraints: once an edge stub is selected (equivalently, at odd values of t), the branching factor of the search is small since only *connectable* nodes need to be considered. However, the number of actions when selecting the origin node of the edge (at even values of t) remains large, which might become detrimental to performance as the size of the network grows (as illustrated in Figure 4.3). Can this be mitigated?

We consider limiting the nodes that can initiate connections to a subset – which prunes away all branches in the search tree that are not part of this set. Concretely, let a *reduction policy* ϕ be a function that, given the initial graph G_0, outputs a strict

subset of its nodes.[3] Then, we modify our definition of allowed actions as follows. Under a reduction policy ϕ, we define

$$\mathcal{A}_\phi(S_t) = \begin{cases} \mathcal{A}(S_t) \cap \phi(G_0), \text{ if } t \bmod 2 = 0 \\ \mathcal{A}(S_t), \text{ otherwise.} \end{cases} \tag{4.5}$$

We investigate the following class of reduction policies: a node v_i is included in $\phi(G_0)$ *if and only if it is among the top nodes ranked by a local node statistic* $\lambda(v_i)$. Letting $\text{gain}(e_{i,j}) = \mathcal{F}(V, E \cup \{e_{i,j}\}, f, w) - \mathcal{F}(V, E, f, w)$, we consider the node statistics λ listed below:

- *Degree* (*DEG*): $\deg(v_i)$;
- *Inverse Degree* (*ID*): $\max_{j \,.\, i \neq j} (\deg(v_j)) - \deg(v_i)$;
- *Number of Connections* (*NC*): $|\mathcal{K}(v_i)|$;
- *Best Edge* (*BE*): $\max_{v_j \in \mathcal{K}(v_i)} \text{gain}(e_{i,j})$;
- *Best Edge Cost Sensitive* (*BECS*): $\max_{v_j \in \mathcal{K}(v_i)} \frac{\text{gain}(e_{i,j})}{c(e_{i,j})}$;
- *Average Edge* (*AE*): $\sum_{v_j \in \mathcal{K}(v_i)} \text{gain}(e_{i,j}) / |\mathcal{K}(v_i)|$;
- *Average Edge Cost Sensitive* (*AECS*): $\sum_{v_j \in \mathcal{K}(v_i)} \frac{\text{gain}(e_{i,j})}{c(e_{i,j})} / |\mathcal{K}(v_i)|$.

Since the performance of reduction strategies may depend on $\mathcal{F}$, we treat them as a tunable hyperparameter. To illustrate the impact of the reduction policy, we consider the following analysis: starting from the same initial graph, a selection of 1000 subsets of size 40% of all nodes is obtained using a uniform random reduction policy. We show the empirical distribution of the reward obtained by UCT with different sampled subsets in Figure 4.4. Since the subset that is selected has an important influence on performance, a reduction policy yielding high-reward subsets is highly desirable (effectively, we want to bias subset selection towards the upper tail of the distribution of obtained rewards).

[3] Learning a reduction policy in a data-driven way is also possible; however, obtaining the supervision signal (i.e., node rankings over multiple MCTS runs) is very expensive. Furthermore, since we prioritise performance on specific graph instances over generalisable policies, simple statistics may be sufficient. Still, a learned reduction policy that predicts an entire set at once may be able to identify better subsets than individual statistics alone. Furthermore, a possible limitation of using a reduction policy is that the optimal trajectory in the full MDP may be excluded. We consider these worthwhile directions for future investigations.

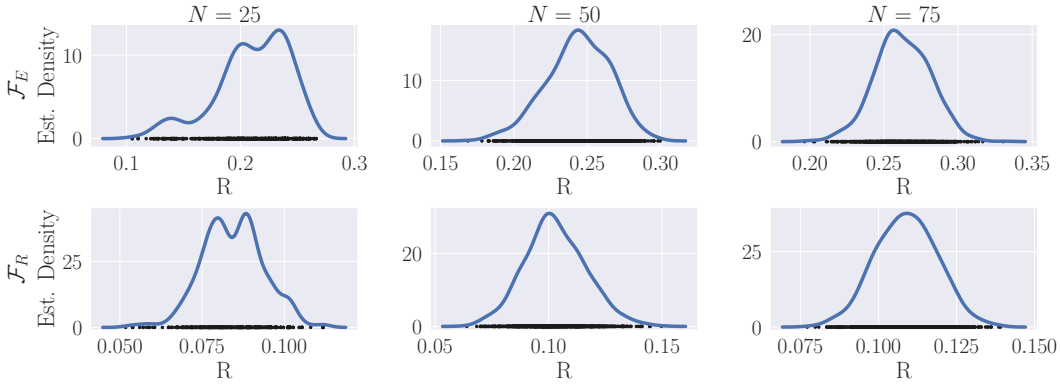


Figure 4.4: Empirical distribution of rewards obtained for subsets selected by a uniform random ϕ.

Algorithm 1 Spatial Graph UCT (SG-UCT).

Input: spatial graph $G_0 = (V, E_0, f, w)$, objective function $\mathcal{F}$,
edge budget b_0, per-action simulation budget b_{sims}, reduction policy ϕ
Output: actions $A_0, \dots A_{T-1}$
for v in V: compute $\mathcal{K}(v)$
compute $\Phi = \phi(G_0)$ ▷ apply reduction policy ϕ
$t = 0$, $b_0 = \frac{\eta}{100} \cdot c(E_0)$, $S_0 = (G_0, \{\varnothing\}, b_0)$
$r_{\text{max}} = -\infty$
bestActs = Array(), pastActs = Array()
loop
  if $|\mathcal{A}_\phi(S_t)| = 0$ **then** return bestActs
  create root node n_{root} from S_t
  for $i = 0$ to b_{sims}
    n_{border}, treeActs = TreePolicy(n_{root}, Φ) ▷ follow to border of current tree
    r, outActs = MinCostPolicy(n_{border}, Φ) ▷ cost-sensitive default policy
    Backup(n_{border}, r)
    if $r > r_{\text{max}}$ **then**
      bestActs = [pastActs, treeActs, outActs] ▷ memoisation of the best trajectory
      $r_{\text{max}} = r$
  n_{child} = MaxChild(n_{root})
  Append(pastActs, GetAction(n_{child}))
  $t+ = 1$
  S_t = GetState(n_{child})
return bestActs

4.3 Evaluation Protocol

Definitions of Space and Distance. For all experiments in this chapter, we consider the unit 2D square as our space, i.e. we let $\Lambda = [0, 1] \times [0, 1]$ and the distance d be Euclidean distance. In case the graph is defined on a spherical coordinate system (as is the case with physical networks positioned on Earth), we use the WGS84 variant of the Mercator projection to project nodes to the plane; then normalise to the unit plane. We opt to project the coordinates onto a plane, rather than onto a unit sphere,

since the networks that we consider are at the geographical scale of a city or, at most, a country, for which a plane is a reasonable local approximation. Projecting onto a unit sphere would introduce more significant distortions. For simplicity, we consider uniform weights, i.e., $w(e) = 1 \; \forall \; e \in E$. The approach can be extended to networks with heterogeneous weights by defining the action space at even timesteps as a product between the set of valid nodes and the set of possible weight values, as well as adopting an objective function that incorporates them as a drop-in replacement (e.g., the formalisation of efficiency in [33]).

Synthetic and Real-World Graphs. As a means of generating synthetic graph data, we use the popular model proposed by Kaiser and Hilgetag in [194], which simulates a process of growth for spatial networks. Related to the Waxman model [355], in this model the probability that a connection is created is inversely proportional to its distance from existing nodes. The distinguishing feature of this model is that, unlike, e.g., the random geometric graph [88], this model produces connected networks: a crucial characteristic for the types of objectives we consider. We henceforth refer to this model as Kaiser-Hilgetag (shortened KH). We use $\alpha_{\text{KH}} = 10$ and $\beta_{\text{KH}} = 10^{-3}$, which yields sparse graphs with scale-free degree distributions – a structure similar to road infrastructure networks. We also evaluate performance on networks belonging to the following real-world datasets, detailed in Table 4.1: *Internet* (a dataset of internet backbone infrastructure from a variety of ISPs [212], which display spatial characteristics due to their country-level scales) and *Metro* (a dataset of metro networks in major cities around the world [296]). Due to computational budget constraints, we limit the sizes of networks considered to $|V| = 150$.

Table 4.1: Real-world spatial graphs considered in the evaluation.

Dataset	Graph	$\lvert V\rvert$	$\lvert E\rvert$
Internet	Colt	146	178
	GtsCe	130	169
	TataNld	141	187
	UsCarrier	138	161
Metro	Barcelona	135	159
	Beijing	126	139
	Mexico	147	164
	Moscow	134	156
	Osaka	107	122

Setup. For all experiments, we allow agents a modification budget equal to a percentage η of the total cost of the edges of the original graph, i.e., $b_0 = \frac{\eta}{100} \cdot c(E_0)$. We use $\eta = 10$. The parameter ρ controlling the range of possible connections takes the value $\rho = 1$ for synthetic graphs and $\rho = 2$ for real-world graphs, respectively. This

is because the real-world networks tend to have comparatively fewer long-range connections than the synthetic ones. Moreover, a value of ρ that is too low could prohibit longer connections entirely in case they do not already exist in the seed network, severely restricting the feasible solution space. Confidence intervals are computed using results of 10 runs, each initialised using a different random seed. Rollouts are not truncated. We allow a number of node expansions per move b_{sims} equal to $20|V|$ (a larger number of expansions can improve performance, but leads to diminishing returns), and select as the move at each step the node with the maximum average value (commonly referred to as MaxChild). Full details of the hyperparameter selection methodology and the values used are provided in Appendix B.

Evaluation Metrics. We report the rewards obtained by the different approaches, i.e., the difference in $\mathcal{F}$ between the final and initial graphs. This has the advantage that the values are directly interpretable and correspond to the gain in the objective function that can be obtained with a certain budget. We note that the relatively small scale of the values reported in the tables and figures are due to the fact that they represent differences in objective function values that range between $[0, 1]$ for efficiency and $[1/N, 0.5]$ for robustness, as reported in Section 4.2.1.

Baselines. The baselines against which we compare are detailed below and represent several prior methods discussed in Sections 2.3.1 and 2.3.2. All of the techniques, including our proposed algorithm, solve the problem approximately, since no methods currently exist to solve the problem exactly beyond the smallest of graphs. We do not consider previous RL-based methods such as that proposed in Chapter 3 directly, since they are unsuitable for training at the scale of the largest graphs taken into consideration.

- *Random* ($\mathcal{F}_E, \mathcal{F}_R$): Randomly selects an available action.
- *Greedy* ($\mathcal{F}_E, \mathcal{F}_R$): A local search that selects the edge that gives the biggest improvement in $\mathcal{F}$: formally, it adds the edge e that satisfies $\operatorname{argmax}_e \text{gain}(e)$ to the graph structure. This approach builds a shallow search tree of depth two, evaluating the objective function for all leaf nodes.
- *GreedyCS* ($\mathcal{F}_E, \mathcal{F}_R$): We also consider the cost-sensitive variant of the greedy local search, for which the gain is offset by the cost: $\operatorname{argmax}_e \frac{\text{gain}(e)}{c(e)}$.

- *MinCost* $(\mathcal{F}_E, \mathcal{F}_R)$: Selects edge e that satisfies $\operatorname{argmin}_e c(e)$.

- *LBHB* $(\mathcal{F}_E)$: Adds an edge between the node with Lowest Betweenness and the node with Highest Betweeness; formally, letting the betweeness centrality of node v be $\mathrm{bw}(v)$, this strategy adds an edge between nodes $\operatorname{argmin}_i \mathrm{bw}(v_i)$ and $\operatorname{argmax}_j \mathrm{bw}(v_j)$, with $i \neq j$.

- *LDP* $(\mathcal{F}_R)$: Adds an edge between the vertices with the *Lowest Degree Product*, i.e., vertices v_i, v_j that satisfy $\operatorname{argmin}_{i,j \,.\, i \neq j} \deg(v_i) \cdot \deg(v_j)$.

- *FV* $(\mathcal{F}_R)$: Adds an edge between vertices v_i, v_j satisfying $\operatorname{argmax}_{i,j \,.\, i \neq j} |y_i - y_j|$, where $\mathbf{y}$ is the Fiedler Vector [122, 348], and y_i denotes the i-th element of $\mathbf{y}$.

- *ERes* $(\mathcal{F}_R)$: Adds an edge between vertices with the highest pairwise Effective Graph Resistance, i.e., nodes v_i, v_j that satisfy $\operatorname{argmax}_{i,j \,.\, i \neq j} \Omega_{i,j}$. $\Omega_{i,j}$ is defined as $(\tilde{\mathbf{L}}^{-1})_{i,i} + (\tilde{\mathbf{L}}^{-1})_{j,j} - 2(\tilde{\mathbf{L}}^{-1})_{i,j}$, where $\tilde{\mathbf{L}}^{-1}$ is the pseudoinverse of the graph Laplacian $\mathbf{L}$ [350].

4.4 Evaluation Results

Our evaluation considers two dimensions of interest. In the first half of this section, we examine the degree to which the proposed method and the baselines can optimise the target objective. In the second half, we benchmark the computational time that the various approaches need for finding a solution, a highly relevant practical concern.

4.4.1 Optimising Graph Structure

Synthetic Graph Results. In this experiment, we consider 50 KH graphs each of sizes $\{25, 50, 75\}$. The results obtained are shown in the top half of Table 4.2. We summarise our findings as follows: SG-UCT outperforms UCT and all other methods in all the settings tested, obtaining 13% and 32% better performance than UCT on the largest synthetic graphs for the efficiency and robustness measures respectively. For $\mathcal{F}_R$, UCT outperforms all baselines, while for $\mathcal{F}_E$ the performance of the Greedy baselines is superior to UCT. Interestingly, MinCost yields solutions that are superior to all other heuristics and comparable to search-based methods while being very cheap to evaluate. Furthermore, UCT performance decays in comparison to the baselines as the size of the graph increases.

Table 4.2: Gains in the values of the objective function $\mathcal{F}$ between the optimised and the original graphs obtained by baselines, UCT, and SG-UCT on synthetic graphs.

Objective	$\mathcal{F}_E$			$\mathcal{F}_R$		
$\lvert V\rvert$	25	50	75	25	50	75
Random	0.128$\pm$0.008	0.089$\pm$0.005	0.077$\pm$0.004	0.031$\pm$0.002	0.033$\pm$0.002	0.035$\pm$0.002
Greedy	0.298	0.335	0.339	0.064	0.078	0.074
Greedy$_{\text{CS}}$	0.281	0.311	0.319	0.083	0.102	0.115
LDP	—	—	—	0.049	0.044	0.040
FV	—	—	—	0.051	0.049	0.049
ERes	—	—	—	0.054	0.057	0.052
MinCost	0.270	0.303	0.315	0.065	0.082	0.099
LBHB	0.119	0.081	0.072	—	—	—
UCT	0.288$\pm$0.003	0.307$\pm$0.003	0.311$\pm$0.003	0.092$\pm$0.001	0.112$\pm$0.002	0.120$\pm$0.001
SG-UCT (ours)	**0.305**$\pm$0.000	**0.341**$\pm$0.000	**0.352**$\pm$0.001	**0.107**$\pm$0.001	**0.140**$\pm$0.001	**0.158**$\pm$0.000

Table 4.3: Gains in the values of the objective function $\mathcal{F}$ between the optimised and the original graphs obtained by baselines, UCT, and SG-UCT on real-world graphs.

			Random	Greedy	Greedy$_{\text{CS}}$	LDP	FV	ERes	MinCost	LBHB	UCT	SG-UCT (ours)
$\mathcal{F}$	$\mathcal{G}$	Graph										
$\mathcal{F}_E$	Internet	Colt	0.081$\pm$0.003	0.180	0.127	—	—	—	0.127	0.098	0.164$\pm$0.003	**0.199**$\pm$0.000
		GtsCe	0.017$\pm$0.007	0.123	0.089	—	—	—	0.082	0.014	0.110$\pm$0.004	**0.125**$\pm$0.002
		TataNld	0.020$\pm$0.004	0.106	0.082	—	—	—	0.078	0.015	0.102$\pm$0.003	**0.110**$\pm$0.002
		UsCarrier	0.026$\pm$0.014	**0.178**	0.097	—	—	—	0.097	0.026	0.171$\pm$0.005	**0.178**$\pm$0.002
	Metro	Barcelona	0.020$\pm$0.005	0.071	0.063	—	—	—	0.063	0.003	0.067$\pm$0.002	**0.076**$\pm$0.000
		Beijing	0.008$\pm$0.003	0.036	0.033	—	—	—	0.028	0.003	0.041$\pm$0.001	**0.046**$\pm$0.001
		Mexico	0.007$\pm$0.002	0.037	0.035	—	—	—	0.032	0.011	0.037$\pm$0.001	**0.041**$\pm$0.000
		Moscow	0.011$\pm$0.003	0.052	0.042	—	—	—	0.038	0.007	0.043$\pm$0.001	**0.053**$\pm$0.001
		Osaka	0.017$\pm$0.005	0.097	0.082	—	—	—	0.082	0.010	0.093$\pm$0.003	**0.102**$\pm$0.000
$\mathcal{F}_R$	Internet	Colt	0.007$\pm$0.004	0.034	0.075	0.005	0.006	0.009	0.075	—	0.055$\pm$0.003	**0.089**$\pm$0.000
		GtsCe	0.023$\pm$0.011	0.064	0.101	0.048	0.017	0.031	0.099	—	0.098$\pm$0.005	**0.155**$\pm$0.003
		TataNld	0.017$\pm$0.010	0.043	0.083	0.011	-0.002	0.013	0.074	—	0.093$\pm$0.006	**0.119**$\pm$0.002
		UsCarrier	0.010$\pm$0.004	0.078	0.060	0.035	0.038	0.033	0.041	—	0.085$\pm$0.007	**0.125**$\pm$0.003
	Metro	Barcelona	0.020$\pm$0.007	0.057	0.073	0.010	0.009	0.036	0.071	—	0.076$\pm$0.004	**0.115**$\pm$0.002
		Beijing	0.004$\pm$0.004	0.014	0.054	0.003	0.002	0.001	0.037	—	0.055$\pm$0.003	**0.062**$\pm$0.001
		Mexico	0.007$\pm$0.004	0.040	0.043	0.003	0.005	0.011	0.038	—	0.051$\pm$0.002	**0.068**$\pm$0.001
		Moscow	0.013$\pm$0.005	0.072	0.057	0.033	0.042	0.034	0.031	—	0.090$\pm$0.003	**0.109**$\pm$0.002
		Osaka	0.003$\pm$0.006	0.042	0.064	0.008	0.011	0.015	0.064	—	0.066$\pm$0.004	**0.072**$\pm$0.001

Real-world Graph Results. The results obtained for real-world graphs are shown in Table 4.3. As with synthetic graphs, we find that SG-UCT performs better than UCT and all other methods in all settings tested. The aggregated differences in performance between SG-UCT and UCT are 10% and 39% for $\mathcal{F}_E$ and $\mathcal{F}_R$ respectively.

Ablation Study. Since SG-UCT comprises three components that are active at the same time as defined in Section 4.2.3, we conduct an ablation study on synthetic graphs in order to assess their impact. To achieve this, we consider standard UCT and enable each of the three components in turn, measuring the performance of the algorithm. The obtained results are shown in Table 4.4, in which the results for each component are separated by the horizontal lines. SG-UCT$_{\text{BTM}}$ denotes UCT with Best

Table 4.4: Ablation study that examines the impact of the three components of SG-UCT by enabling each of them separately in standard UCT. The components are Best Trajectory Memoisation, the MINCOST simulation policy, and the various choices of a reduction policy. The results are separated by horizontal lines. Each value represents the gain in the value of the objective function $\mathcal{F}$ between the optimised and the original graphs.

Objective	$\mathcal{F}_E$			$\mathcal{F}_R$		
$\lvert V\rvert$	25	50	75	25	50	75
UCT	$0.288_{\pm 0.003}$	$0.307_{\pm 0.003}$	$0.311_{\pm 0.003}$	$0.092_{\pm 0.001}$	$0.112_{\pm 0.002}$	$0.120_{\pm 0.001}$
SG-UCT$_{\text{BTM}}$	$0.304_{\pm 0.001}$	$0.324_{\pm 0.002}$	$0.324_{\pm 0.002}$	$0.106_{\pm 0.001}$	$0.123_{\pm 0.001}$	$0.128_{\pm 0.001}$
SG-UCT$_{\text{MINCOST}}$	$0.299_{\pm 0.001}$	$0.327_{\pm 0.001}$	$0.333_{\pm 0.001}$	$0.105_{\pm 0.001}$	$0.131_{\pm 0.001}$	$0.153_{\pm 0.001}$
SG-UCT$_{\text{RAND-80}}$	$0.284_{\pm 0.005}$	$0.305_{\pm 0.003}$	$0.303_{\pm 0.003}$	$0.091_{\pm 0.001}$	$0.111_{\pm 0.001}$	$0.119_{\pm 0.001}$
SG-UCT$_{\text{RAND-60}}$	$0.271_{\pm 0.007}$	$0.288_{\pm 0.004}$	$0.288_{\pm 0.003}$	$0.089_{\pm 0.003}$	$0.107_{\pm 0.002}$	$0.115_{\pm 0.002}$
SG-UCT$_{\text{RAND-40}}$	$0.238_{\pm 0.009}$	$0.263_{\pm 0.005}$	$0.271_{\pm 0.003}$	$0.083_{\pm 0.001}$	$0.102_{\pm 0.002}$	$0.110_{\pm 0.002}$
SG-UCT$_{\text{DEG-40}}$	$0.237_{\pm 0.003}$	$0.262_{\pm 0.003}$	$0.255_{\pm 0.002}$	$0.069_{\pm 0.001}$	$0.086_{\pm 0.001}$	$0.092_{\pm 0.001}$
SG-UCT$_{\text{ID-40}}$	$0.235_{\pm 0.002}$	$0.268_{\pm 0.001}$	$0.283_{\pm 0.001}$	$0.094_{\pm 0.001}$	$0.114_{\pm 0.001}$	$0.124_{\pm 0.001}$
SG-UCT$_{\text{NC-40}}$	$0.234_{\pm 0.003}$	$0.268_{\pm 0.002}$	$0.262_{\pm 0.003}$	$0.071_{\pm 0.001}$	$0.087_{\pm 0.002}$	$0.092_{\pm 0.001}$
SG-UCT$_{\text{BE-40}}$	$0.286_{\pm 0.002}$	$0.304_{\pm 0.002}$	$0.297_{\pm 0.001}$	$0.088_{\pm 0.001}$	$0.108_{\pm 0.001}$	$0.115_{\pm 0.001}$
SG-UCT$_{\text{BECS-40}}$	$0.290_{\pm 0.001}$	$0.316_{\pm 0.001}$	$0.319_{\pm 0.002}$	$0.097_{\pm 0.001}$	$0.115_{\pm 0.001}$	$0.121_{\pm 0.001}$
SG-UCT$_{\text{AE-40}}$	$0.286_{\pm 0.002}$	$0.302_{\pm 0.003}$	$0.297_{\pm 0.002}$	$0.088_{\pm 0.001}$	$0.103_{\pm 0.001}$	$0.114_{\pm 0.001}$
SG-UCT$_{\text{AECS-40}}$	$0.289_{\pm 0.001}$	$0.317_{\pm 0.001}$	$0.319_{\pm 0.002}$	$0.098_{\pm 0.001}$	$0.117_{\pm 0.001}$	$0.126_{\pm 0.001}$

Trajectory Memoisation, SG-UCT$_{\text{MINCOST}}$ denotes UCT with the cost-based default policy, SG-UCT$_{\phi-q}$ for q in $\{40, 60, 80\}$ denotes UCT with a particular reduction policy ϕ, and q represents the percentage of original nodes that are selected by ϕ.

We find that BTM indeed brings a net improvement in performance: on average, 5% for $\mathcal{F}_E$ and 11% for $\mathcal{F}_R$. The benefit of the cost-based default policy is substantial (especially for $\mathcal{F}_R$), ranging from 4% on small graphs to 27% on the largest graphs considered, and increases the higher the level of bias. This is further evidenced in Figure 4.5, which shows the average reward obtained as a function of β. This illustrates that the cost-based simulation policy yields the largest gains in the value of the objective function out of all the three algorithmic components. In terms of reduction policies, even for a random selection of nodes, we find that the performance penalty paid is comparatively small: a 60% reduction in ac-

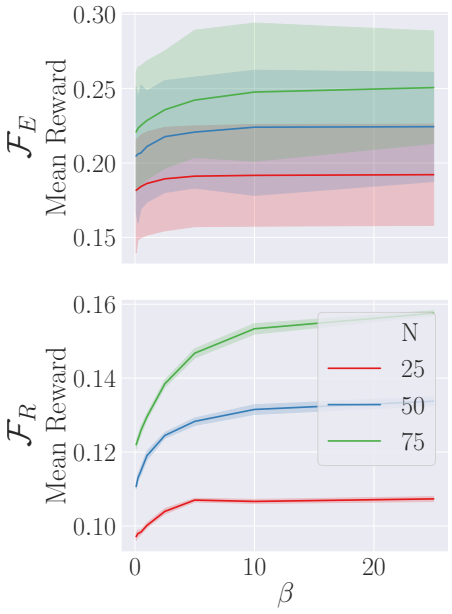


Figure 4.5: Average reward for SG-UCT$_{\text{MINCOST}}$ as a function of β, which suggests that a bias towards low-cost edges is beneficial.

Table 4.5: Representative wall clock time for the algorithms measured in hours, minutes, and seconds on real-world graphs.

$\mathcal{F}$	$\mathcal{G}$	Graph	Random	Greedy	GreedyCS	LDP	FV	ERes	MinCost	LBHB	UCT	SG-UCT (ours)
$\mathcal{F}_E$	Internet	Colt	<00:01	0:20:50	1:19:05	—	—	—	0:00:02	0:00:02	0:51:10	1:30:15
		GtsCe	<00:01	0:18:06	0:37:23	—	—	—	0:00:01	0:00:01	0:33:35	1:01:22
		TataNld	<00:01	0:11:26	0:34:25	—	—	—	0:00:01	0:00:01	0:42:12	0:53:47
		UsCarrier	<00:01	0:06:08	0:14:45	—	—	—	<00:01	<00:01	0:26:20	0:38:11
	Metro	Barcelona	<00:01	0:03:22	0:07:11	—	—	—	<00:01	0:00:01	0:23:05	0:26:45
		Beijing	<00:01	0:01:52	0:04:09	—	—	—	<00:01	<00:01	0:15:32	0:18:44
		Mexico	<00:01	0:01:58	0:03:08	—	—	—	<00:01	0:00:02	0:16:44	0:24:08
		Moscow	<00:01	0:03:00	0:05:44	—	—	—	<00:01	0:00:01	0:15:03	0:22:57
		Osaka	<00:01	0:01:38	0:02:32	—	—	—	<00:01	<00:01	0:13:23	0:14:49
$\mathcal{F}_R$	Internet	Colt	<00:01	0:11:39	2:35:10	<00:01	<00:01	<00:01	0:00:03	—	0:37:19	1:27:04
		GtsCe	<00:01	0:08:16	1:11:32	<00:01	<00:01	<00:01	0:00:01	—	0:30:51	1:43:39
		TataNld	<00:01	0:07:24	1:06:12	<00:01	<00:01	<00:01	0:00:01	—	1:05:55	1:41:43
		UsCarrier	<00:01	0:06:50	0:37:12	<00:01	<00:01	<00:01	0:00:01	—	0:31:19	1:11:30
	Metro	Barcelona	<00:01	0:04:00	0:16:59	<00:01	<00:01	<00:01	<00:01	—	0:30:40	0:50:20
		Beijing	<00:01	0:01:03	0:09:55	<00:01	<00:01	<00:01	<00:01	—	0:26:54	0:33:05
		Mexico	<00:01	0:02:53	0:09:08	<00:01	<00:01	<00:01	<00:01	—	0:31:12	0:59:55
		Moscow	<00:01	0:03:23	0:13:37	<00:01	<00:01	<00:01	<00:01	—	0:30:11	0:31:49
		Osaka	<00:01	0:01:42	0:06:30	<00:01	<00:01	<00:01	<00:01	—	0:14:46	0:22:22

tions translates to at most 15% reduction in performance, and as little as 5%; the impact of random action reduction becomes smaller as the size of the network grows. The best-performing reduction policies are those based on a node's gains, with BECS and AECS outperforming UCT with no action reduction. For the $\mathcal{F}_R$ objective, a poor choice of bias can be harmful: prioritising nodes with high degrees leads to a 32% reduction in performance compared to UCT, while a bias towards lower-degree nodes is beneficial.

4.4.2 Running Time and Scalability

An important aspect to consider is the computational time needed to decide which edges to add. The wall clock time for a complete run of the algorithms on the real-world graphs is shown in Table 4.5 and is measured on a single core of an Intel Xeon E5-2630 v3 (2014) processor.

A distinction between heuristic and search-based methods is immediately apparent. The former category of methods, which do not evaluate the objective function directly but instead rely on local node statistics or spectral properties, result in timings that do not exceed a few seconds and often complete their evaluation within a second. On the other hand, the latter category, comprising the greedy and tree search variants, require larger timescales. Within this category, the cost-sensitive methods (GreedyCS and SG-UCT) typically require more time with respect to their cost-agnostic counterpart. This is due to longer action sequences resulting in more

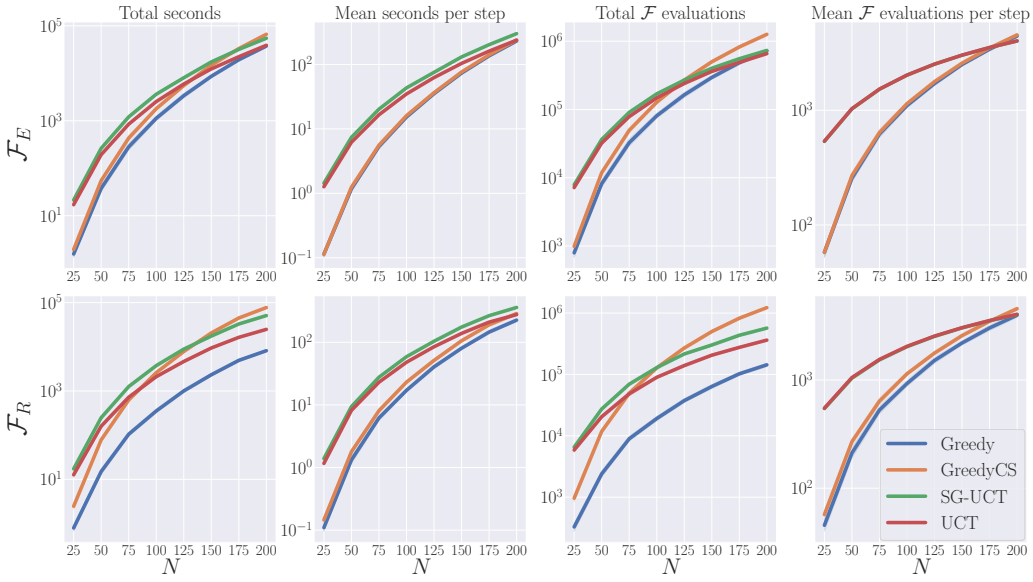


Figure 4.6: Wall clock time (leftmost two columns) and number of objective function evaluations (rightmost two) used by the different algorithms on synthetic graphs.

edges being added and hence more simulations being performed, as well as the overheads needed for computing distances and using them to weight edge choices.

In order to further probe the scaling behaviour of the search-based approaches, we carry out an additional experiment using synthetic KH graphs of sizes between $N = 25$ and $N = 200$. Each column in Figure 4.6 represents a different metric for the Greedy and UCT variants; namely: the total and mean wall clock time per search step, as well as the total and mean number of objective function evaluations. As shown in the third panel, it is indeed the case that the cost-sensitive searches typically perform more objective function evaluations, an aspect that is reflected in their total wall clock timings (first panel). Furthermore, the UCT variants are noticeably slower on the smaller graphs, an aspect that is explained by the increased number of simulations compared to the greedy searches. Nevertheless, an advantage of UCT is that the user is able to specify a budget in terms of number of simulations that suits their requirements, different from the $20N$ simulations used throughout our experiments as reported in Section 4.3. On the largest graphs, the greedy approaches begin performing more objective evaluations per step (fourth panel), and hence the wall clock timings per search step are also smaller (second panel), becoming nearly identical to the UCT variants on the largest of graphs. This is due to the fact that the Greedy approaches need to consider $\mathcal{O}(|V|)^2$ actions at each step compared to

the $\mathcal{O}(|V|)$ required by UCT and SG-UCT, and this difference begins to manifest at larger scales.

Let us now revisit the claim in Section 4.1 regarding superior scalability with respect to prior RL methods, which require training a model before it can be used to improve a particular network. Namely, in Chapter 3, we have reported a wall clock time that is equivalent to 56 hours of a single core of a comparable CPU to train a model on graphs of size $N = 20$. Due to the complexity of the problem, we did not train models directly beyond graphs with $N = 50$. In contrast, on similar computational infrastructure, the SG-UCT method proposed in this chapter requires 11 hours on average to optimise a much larger graph with $N = 200$ nodes. Hence, in cases in which we are interested in improving a particular network, our proposed method yields an important improvement in scalability by sidestepping the cost of model training altogether.

Given the timings above, we expect our method to be applicable out-of-the-box to graphs with several hundreds of nodes. Practically speaking, further improvements in the speed and performance of the algorithm can be obtained by implementing it fully in a low-level programming language (in the reference implementation, the core algorithm is implemented in Python and the objective functions, the main speed bottleneck, are implemented in C++). It is possible to speed up the computation of the objective functions by exploiting their incremental structure (see Footnote 2 in this chapter), using cheaper proxy quantities (i.e., the various spectral indicators of resilience, which can circumvent needing to run simulations), or learning an approximate model of such global processes. Root and leaf parallelisation in Monte Carlo Tree Search [51] can also be employed to speed up individual runs. Beyond this scale, it may be necessary to consider different levels of abstraction (for example, by treating the problem hierarchically). Due to the inherent difficulty of combinatorial optimisation problems, current solutions are generally limited to this scale.

We also remark that there naturally exists a trade-off between the computational cost of the methods and their ability to improve the values of the given objective function. Namely, the heuristic approaches can be evaluated very quickly but yield smaller performance improvements (in terms of the values of the objective function) compared to the search-based methods, which require more computational time.

However, the type of infrastructure networks (such as road, rail or wired communication networks) motivating our work are very expensive to build in the real world. We argue that finding a solution as close as possible to optimality is worthwhile, since the cost of additional simulations would be negligible compared to that of laying down roads or tracks characterised by sub-optimal layouts in the real world.

4.5 Discussion

Given our formulation of goal-directed graph construction in spatial networks as a combinatorial optimisation problem, we note that other generic optimisation algorithms can in principle be used. The Greedy baselines are local searches over a horizon of two actions, in effect constructing a shallow search tree in which the objective function is evaluated for all the leaf nodes. Approaches such as branch-and-bound share some similarities since they also construct a tree of the solution space. However, to the best of our knowledge, no meaningful bounding criteria exist for the considered objectives. In such cases, branch-and-bound degenerates to an Exhaustive Search, which is infeasible beyond the smallest of graphs.

Our work is also related to a large body of prior work in network design and optimisation [7] that were previously discussed [273, 364, 105]. Another important class of approaches for network design is based on the concept of *spanner of a graph*, i.e., a subgraph of the given graph in which the length of any path does not exceed a given threshold. Such methods have found applications in designing communication networks [239, 163]. However, none of the methods are applicable in our setting given their focus on very specific problem definitions.

More broadly, works in the operations research literature formulate the problem under consideration as a mathematical program on a case-by-case basis, for which highly optimised combinatorial solvers can be applied. In contrast, our method is generic and makes no assumptions about the objective at hand. In principle, any objective function defined on a spatial graph is admissible, allowing our method to optimise for complex, non-linear objectives that may arise in the real world (the resilience of the network to targeted attacks is one example of such a non-linear objective). Hence, our approach has the potential to be used for many more real-world problems, and can be applied to objectives for which no exact algorithms are currently known.

Necessarily, the generic nature of our method means that it may perform worse on certain specific problems, for example those with linear constraints and objectives. The trade-offs arising from the use of generic ML and decision-making algorithms are a topic of ongoing debate in the combinatorial optimisation community [31], and we view these classes of methods as complementary.

4.6 Summary

In this chapter, we have addressed the problem of optimising the structure of spatial graphs, proposing a positive answer to ***RQ1*** that is complementary to the contributions of Chapter 3. Namely, given an initial spatial graph, a budget defined in terms of edge lengths, and a global objective, we have aimed to find a set of edges to be added to the graph such that the value of the objective is maximised. For the first time, we have formulated this task as a deterministic Markov Decision Process that accounts for how the spatial geometry influences the connections and organisational principles of real-world networks. Building on the UCT framework, we have considered several aspects that characterise this problem space and proposed the Spatial Graph UCT (SG-UCT) algorithm to address them.

Our approach brings an important improvement in scalability with respect to the approach in Chapter 3 for goal-directed graph construction, both in terms of network size that the method can operate on, as well as reduced computational cost. The much-desired gains in scalability expressed in ***RQ3*** are obtained through the use of planning methods that do not require training, alongside realistic feasible reductions in the action space. Our evaluation results show performance that is substantially better than UCT (24% on average and up to 54% in terms of a robustness measure) and all existing baselines taken into consideration, while requiring a computational budget similar to other search-based methods (***RQ2***).

We hope that our work will be used by infrastructure designers and urban planners as a basis for the design of more cost-effective infrastructure systems such as communication, power, transportation, and water distribution networks. Even though we have treated an idealised version of such network design scenarios, we consider that the flexibility of decision-making frameworks to accommodate realistic constraints and objectives holds great promise in this problem space.

Chapter 5

Solving Graph-based Public Goods Games with Tree Search and Imitation Learning

In this chapter, we turn our attention beyond graph structure to finding a Maximal Independent Set of nodes that maximises an objective function. In pursuit of a data-driven approach, we define an MDP that incrementally generates an mIS, and adopt a planning method to search this solution space, outperforming existing methods. Furthermore, we devise a graph Imitation Learning technique that uses search demonstrations to obtain a GNN parametrised policy, which generalises to unseen problem instances while being substantially faster to evaluate.

As an application scenario, we consider a class of Public Goods Games on networks. They represent insightful settings for studying incentives for individual agents to make contributions that, while costly for each of them, benefit the wider society. We adopt the perspective of a central planner with a global view of a network of self-interested agents and the goal of maximising some desired property for a Nash Equilibrium of the networked best-shot Public Goods Game. By exploiting the correspondence between equilibria and mISs, we are able to find desirable configurations efficiently.

5.1 Introduction

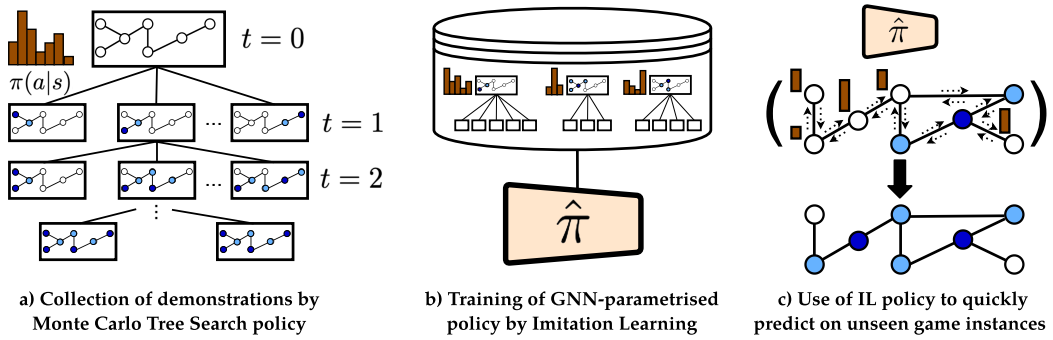
In a *Public Goods Game*, individuals can choose to invest in an expensive good (paying a cost), with benefits being shared by wider society [226]. It is a form of N-party social dilemma that has been used to study the tension between decisions that

benefit the individual and the common good [215]. Aspects characteristic to public goods are observed in many important societal problems such as meeting climate change targets [201, 249], the dynamics of research and innovation [186], the design of effective vaccination programs [127], and, more generally, situations in which contributions are non-excludable. The analysis of this class of games is related to ongoing efforts to study cooperation in multi-agent systems as a means of driving progress on societal challenges [84].

The *best-shot* PGG is a variant in which investment decisions are binary and agents benefit if either they or a neighbour own the good [167]. Since patterns of connections along social and geographical dimensions in networks are known to shape individual decision-making [46, 147], a natural restriction is to limit the impact of contributions to an agent's neighbours. Graph-based best-shot Public Goods Games exhibit multiple Pure Strategy Nash Equilibria [90]. Given this multiplicity, a natural question that arises is how to compute equilibria that satisfy some properties, a task known to be NP-complete in general for multiplayer games [143, 78]. Examples of desirable equilibria are those that maximise the social welfare (total utility) of agents or those with a high degree of fairness in terms of contributions. For graph-based best-shot PGGs, it has been shown that each equilibrium corresponds to an mIS [46]: a set of vertices of maximal size in which no two nodes are adjacent. Since enumerating mISs to identify desirable equilibria quickly becomes unfeasible for non-trivially sized graphs, practical alternatives are needed for larger graphs.

Towards this goal, Dall'Asta et al. [90] proposed a centralised algorithm based on Best-Response dynamics that converges to the optimal equilibrium (with respect to social welfare) in the limit of infinite time, and suggested a tractable simulated annealing alternative. Levit et al. [230] proved that the general version of the best-shot PGG is a potential game and derived an algorithm for finding equilibria based on *side payments*, which are used by agents that are unhappy with their outcome to convince neighbours to switch. While superior results were obtained over Best-Response dynamics, there is still a wide gap between the equilibria found by this approach and optimal equilibria as found by Exhaustive Search on small graphs. Furthermore, current methods cannot optimise for criteria other than social welfare.

The contributions of this chapter are two-fold and can be summarised as follows.



a) Collection of demonstrations by Monte Carlo Tree Search policy

b) Training of GNN-parametrised policy by Imitation Learning

c) Use of IL policy to quickly predict on unseen game instances

Figure 5.1: Schematic of our approach for finding desirable equilibria in the graph-based best-shot game. (a) We exploit the correspondence between agents acquiring the public good in equilibria (pictured in dark blue above) and the mIS substructure of graphs. We define an MDP that incrementally grows an independent set until it is maximal, and use MCTS to plan in this MDP in order to find desirable equilibrium configurations of the game. (b) We propose a Graph Imitation Learning method which uses demonstrations of the MCTS policy π to learn a policy $\hat{\pi}$ parametrised by a GNN. (c) We use $\hat{\pi}$ to find optimal equilibrium configurations on unseen instances of the game.

Firstly, we propose to take advantage of the connection between equilibria in this class of games and mISs. This relationship allows us to define an MDP that incrementally generates an mIS to optimise a desired property; thus, every configuration found is, by construction, an equilibrium of the game.[1] We adopt a variant of the Monte Carlo Tree Search algorithm for planning in this MDP. On small graphs, where an exhaustive enumeration of equilibria can be performed, the best outcomes found by this method are matched in most settings. On larger graphs, existing methods are outperformed, especially in cases in which the costs for acquiring the public good differ among players.

Secondly, we devise a way to learn the structure of the solutions found by the planning algorithm based on Imitation Learning, such that predictions can be obtained on unseen game instances without repeating the search process. Specifically, we use a dataset of demonstrations of the search in order to learn a GNN parametrised policy through Imitation Learning, a procedure we call *Graph Imitation Learning* (*GIL*). The resulting policy is able to achieve 99.5% of the performance of the search method while being approximately three orders of magnitude quicker to evaluate and even exceeding the performance of the original search in some cases. This method is

[1] We highlight the difference between *Maximal* Independent Set and *Maximum* Independent Set. A *Maximal* IS (mIS) is an IS that is not a proper subset of another IS. A *Maximum* IS (MIS) is an mIS of the largest possible size. In PGGs, an MIS may not be a desirable equilibrium, since it involves many players expending the cost.

applicable beyond this class of networked Public Goods Games, i.e., to a variety of graph-based decision-making problems in which a model of the MDP is available and the goal is to maximise a graph-level objective function.

5.2 Methods

In this section, we first discuss some preliminary notions regarding network games, and formalise the problem we set out to address. Subsequently, we introduce our proposed method which, in contrast with previous approaches, directly exploits the relationship between equilibria of this game and the mIS property. We assume that a central planner, with a global view of the graph, seeks to find optimal outcomes of this game. To this end, we formulate the construction of a mIS as an MDP, in which an agent incrementally builds an independent set, receiving a reward signal based on the objective $\mathcal{F}$ once the independent set is maximal. To plan in this MDP, we use the UCT algorithm. We also describe an Imitation Learning procedure based on Behavioural Cloning [284] that can be used to learn a generalisable model of equilibrium structure by mimicking the moves of the search. This policy can be evaluated rapidly on new instances of the game without performing a new search. Our method is illustrated in Figure 5.1 and detailed below.

5.2.1 Preliminaries and Problem Statement

Game Definition. A networked, best-shot Public Goods Game takes place over an undirected, unweighted graph $G = (V, E)$ with no self-loops. Each vertex in $V = \{v_1, v_2, \dots v_N\}$ represents one of the players, while edges E capture the interactions between agents in the game. Each player chooses an action $a^v \in A^v$, where $A^{v_i} = \{0, 1\}$ denotes the action space of player v_i.[2] We let action 1 denote investment in the public good by the agent and 0 denote non-investment. An action profile $\mathbf{a} = [a^{v_1}, \dots, a^{v_N}]$ captures the choices of all players, and $A^{v_1} \times A^{v_2} \times \cdots \times A^{v_N}$ is the set of all possible action profiles. We use $\mathbf{a}^{-v}$ to refer to actions of all other players except v, and $\mathbb{I}(\mathbf{a})$ to denote the set of all players that play action 1 in action profile $\mathbf{a}$. Investment in the public good carries a cost $c(v) \in (0, 1)$ for each player v. We use the terms *identical cost* (IC) to refer to the setting in which costs are the same for all players, and *heterogeneous cost* (HC) to refer to that in which costs are different

[2]Note that we use superscripts to indicate the actions of the players, and the subscript remains reserved for the timestep used in the single-agent RL notation.

between players. We let $\mathbf{c} = [c(v_1), \ldots, c(v_N)]$.

We also define, for each player v_i, the *neighbourhood* $\mathcal{N}(v_i)$ that contains all of its adjacent players, i.e., $\mathcal{N}(v_i) = \{v_j \in V | e_{i,j} \in E\}$. The utility function $\mathcal{U}(v_i, \mathbf{a})$ for player v_i under an action profile $\mathbf{a}$ is defined as:

$$\mathcal{U}(v_i, \mathbf{a}) = \begin{cases} 1 - c(v_i), \text{ if } a^{v_i} = 1 \\ 1, \text{ if } a^{v_i} = 0 \ \wedge \ \exists v_j \in \mathcal{N}(v_i) \,.\, a^{v_j} = 1 \\ 0, \text{ if } a^{v_i} = 0 \ \wedge \ \forall v_j \in \mathcal{N}(v_i) \,.\, a^{v_j} = 0 \end{cases} \tag{5.1}$$

We are interested in Pure Strategy Nash Equilibrium solutions, since there are no mixed strategy equilibria in this game [46]. A pure strategy is a complete, deterministic description of how a player will play the game. An action profile is a PSNE if all players would not gain higher utility by changing their action choice, given the actions of the other players. Formally, $\mathbf{a} \in A^{v_1} \times A^{v_2} \times \cdots \times A^{v_N}$ is a PSNE if and only if $\mathcal{U}(v_i, a^{v_i}, \mathbf{a}^{-v_i}) \geq \mathcal{U}(v_i, a', \mathbf{a}^{-v_i}) \ \forall v_i \in V, \ a' \in A^{v_i}$. The tuple $(G, \mathbf{c})$ defines an instance of this game. We let the set $\mathcal{E}$ denote all Pure Strategy Nash Equilibria of a game instance.

Problem Statement. We formulate our problem as follows: given a game instance $(G, \mathbf{c})$ and an objective function $\mathcal{F}\colon \mathcal{E} \to [0, 1]$, the goal is to find the PSNE for which $\mathcal{F}$ is maximised; concretely, finding $\mathbf{a}$ that satisfies $\operatorname{argmax}_{\mathbf{a} \in \mathcal{E}} \mathcal{F}(\mathbf{a})$.

What constitutes a desirable equilibrium in this game? From a utilitarian perspective, a *desirable* equilibrium is one that *maximises the social welfare* of agents. We define the $\mathcal{F}_{SW}(\mathbf{a})$ objective as below, normalising by the number of players N so that games of different sizes are comparable:

$$\mathcal{F}_{SW}(\mathbf{a}) = \frac{\sum_i \mathcal{U}(v_i, \mathbf{a})}{N} \tag{5.2}$$

A further desirable characteristic is *global fairness*, or equality between players' utilities. Letting abs() denote absolute value, we define the fairness objective $\mathcal{F}_{GF}(\mathbf{a})$ as the complement of the Gini coefficient, a measure of inequality:

$$\mathcal{F}_{GF}(\mathbf{a}) = 1 - \frac{\sum_i \sum_j \text{abs}(\mathcal{U}(v_i, \mathbf{a}) - \mathcal{U}(v_j, \mathbf{a}))}{2N \sum_j \mathcal{U}(v_j, \mathbf{a})} \tag{5.3}$$

Maximal Independent Sets. A subset of vertices $I \subseteq V$ is an independent set of the graph $G = (V, E)$ if none of the vertices in the set are adjacent to each other. Formally, $\forall v_i, v_j \in I$ s.t. $i \neq j \ . e_{i,j} \notin E$. An mIS is an independent set that is not a proper subset of any other independent set. Bramoullé and Kranton [46] have proven a bidirectional correspondence between the set of players playing 1 in equilibria of the networked best-shot PGG and mISs. Thus, one way of finding desirable equilibria that is faster than considering all 2^N action profiles could be to enumerate all mISs. However, the best known family of algorithms [49] for this task has worst-case running time $\mathcal{O}(3^{N/3})$, which makes this impractical beyond very small graphs.

5.2.2 MDP Definition

The MDP we propose is defined below and illustrated in Figure 5.2. Note that it is formulated from the perspective of a central planner that acts on the agent society. In particular, the actions a that are mentioned in the remainder of this section refer to the actions of this principal agent.

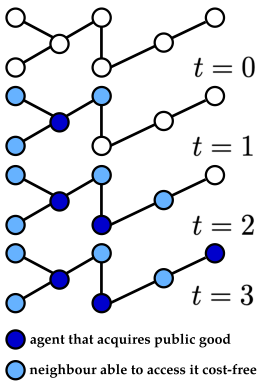


Figure 5.2: Illustration of the proposed MDP.

States $\mathcal{S}$. A state S_t is a tuple (G, I_t) formed of the graph and independent set I_t, with $I_0 = \varnothing$.

Actions $\mathcal{A}$. Actions correspond to selecting the index of a node that is not currently in the independent set. We define available actions as $\mathcal{A}(S_t) = \{\text{select } i \mid v_i \in V \setminus \bigcup_{v_j \in I_t} \mathcal{N}(v_j)\}$, i.e., nodes currently in the independent set and all their neighbours are excluded.

Transition function P. Transitions are deterministic and correspond to the addition of a node to the independent set. Concretely, given that the agent selects node v_a at time $t-1$, the next state is defined as (G, I_t), where $I_t = I_{t-1} \cup \{v_a\}$.

Reward function R. Rewards depend on the objective function $\mathcal{F}$ considered (concretely, $\mathcal{F}_{SW}$ or $\mathcal{F}_{GF}$), and are provided once an mIS is constructed, with all other intermediate rewards being 0.

Terminal. Episodes proceed until the agent has finished constructing an mIS.

5.2.3 Collection of Demonstrations by Monte Carlo Tree Search

Since the MDP formulation above fully describes the transition and reward functions of this MDP, we may use model-based planning algorithms in order to plan an mIS that maximises the desired objective. Concretely, we opt for the UCT [213] variant of the Monte Carlo Tree Search algorithm, which has proven to be an effective framework in a wide variety of decision-making problems. Recall that, as described in Section 2.5.5, the UCT algorithm selects the child node corresponding to action a that maximises

$$UCT(s,a) = \bar{r_a} + 2\epsilon_{\text{UCT}}\sqrt{\frac{2\ln C(s)}{C(s,a)}}, \tag{5.4}$$

where $\bar{r_a}$ is the mean reward observed when taking action a in state s, $C(s)$ is the visit count for the parent node, $C(s,a)$ is the number of child visits, and ϵ_{UCT} is a constant that controls the level of exploration [213]. We use the UCT policy π to collect demonstrations using a set of training game instances $\mathcal{G}^{\text{train}}$, each of which contains N players. This process is illustrated in Figure 5.1a. We let $\mathcal{D}_N$ denote this dataset of demonstrations. Each demonstration is effectively a datapoint and is composed of a tuple $(S_t, \mathcal{A}(S_t), N, \mathbf{v})$, where:

- S_t represents the state from which the move is executed (i.e., for the mIS problem, the underlying graph G and the current independent set I_t);
- $\mathcal{A}(S_t)$ represents the valid actions available at the state (i.e., it contains nodes that are not currently in I_t or are a neighbour of any node in I_t);
- N is the number of nodes (players);
- $\mathbf{v}$ is a vector of size $|\mathcal{A}(S_t)|$, where each entry is equal to $C(S_t,a)$, for each valid action $a \in \mathcal{A}(S_t)$. This represents the number of visits of the search policy from the root state S_t to each of the possible actions a. Given that $C(S_t) = \sum_{a\in\mathcal{A}(S_t)} C(S_t,a)$, the empirical policy is then estimated as $\pi(a|S_t) = \frac{C(S_t,a)}{C(S_t)}$.

5.2.4 Graph Imitation Learning

Policy parametrisation. The main disadvantage in using planning algorithms is that predictions are expensive to obtain for new game instances. To mitigate this, we explore the possibility of learning a policy $\hat{\pi}$ parametrised by a GNN – specifically,

structure2vec [85]. Recall, as described in Section 2.4.3, that it produces an embedding vector $\mathbf{h}_{v_i}$ for each vertex $v_i \in V$ that captures the structure of the graph as well as interactions between neighbours. This is achieved in several rounds of combining the features of a node with an aggregation of its neighbours' features according to weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, to which an activation function is applied. Permutation-invariant embeddings for the state S_t can be derived by summing the node-level embeddings: $h(S_t) = \sum_{v_i \in V} \mathbf{h}_{v_i}$. We use initial node features $\mathbf{x}_{v_i}$ corresponding to a one-hot encoding that captures whether the node is in the independent set, i.e., $\mathbf{x}_{v_i} = [1, 0]$ if $v_i \in I_t$ and $\mathbf{x}_{v_i} = [0, 1]$ otherwise.

An important challenge in this setting is that, at any timestep, a significant number of actions are unavailable. Thus, choosing to have the output layer consist of a softmax layer with one unit per vertex in V is wasteful. In addition, such an architecture is sensitive to node relabelling and not transferable to other graphs. We thus consider a different approach: we make the final layer of the policy network output a *proto-action* $\psi(S_t) = \mathbf{W}_3 \text{ ReLU } (\mathbf{W}_4 \, h(S_t))$. Then, in order to obtain probabilities for each action a, we measure the Euclidean distances $d(a, \psi(S_t))$ between the proto-action and the embeddings of all available actions, normalised using a softmax with temperature τ_{GIL}. This allows us to compute probabilities for all possible actions in a single forward pass. Let $\Theta = \{\mathbf{W}_i\}_{i=1}^4$ denote the full set of parameters for the policy $\hat{\pi}_\Theta$. Formally:

$$\hat{\pi}_\Theta(A_t|S_t) = \frac{\exp(d(\mathbf{x}_{v_{A_t}}, \psi(S_t))/\tau_{\text{GIL}})}{\sum_{a \in \mathcal{A}(S_t)} \exp(d(\mathbf{x}_{v_a}, \psi(S_t))/\tau_{\text{GIL}})} \tag{5.5}$$

Loss Term. For training the policy network, we minimise the KL divergence between the distribution of the policy network and the empirical distribution formed by the number of child visits [11], i.e.,

$$\mathcal{L}(\Theta) = -\sum_{a \in \mathcal{A}(s)} \frac{C(s,a)}{C(s)} \log(\hat{\pi}_\Theta(a|s)) \tag{5.6}$$

Training Strategies. We consider several training strategies for obtaining a model for a target size N of game instances (i.e., games in which there are N players). Since the structure of equilibria for increasingly large number of players are potentially

more complex (i.e., we have to deal with larger graphs), we also consider whether training additionally on smaller graphs (thus simpler examples) brings a generalisation benefit. We employ *curriculum learning*, a methodology successful in a variety of ML settings [30, 377], and compare against training only on the target size as well as mixing the examples of different sizes instead of constructing a curriculum. Concretely, we consider the following training strategies, noting that validation is performed on the target size N:

- **separate**: train only on examples from $\mathcal{D}_N$;
- **mixed**: train on examples from $\bigcup_{k \leq N} \mathcal{D}_k$;
- **curriculum**: carry out the training in several epochs, at each epoch considering only examples from $\mathcal{D}_k$, with each value $k \leq N$ considered in ascending order.

We learn the parameters Θ as well as the softmax temperature τ_{GIL} in an end-to-end fashion. When evaluating this policy, we use greedy action selection. Our method, which we refer to as *Graph Imitation Learning*, is illustrated in Figure 5.1b. A pseudocode description is given in Algorithm 2. We note that the presentation of the algorithm is independent of the specific MDP model used.

5.3 Evaluation Protocol

Game Instances. To evaluate our approach and all baselines, we create instances of network games over graphs with a number of players $N \in \{15, 25, 50, 75, 100\}$. For each size and each underlying graph model, we generate: 10^3 training instances $\mathcal{G}^{\text{train}}$; 10^2 validation instances $\mathcal{G}^{\text{validate}}$ used for hyperparameter optimisation; and 10^2 test instances $\mathcal{G}^{\text{test}}$. To set costs $\mathbf{c}$, in the IC case we fix $c(v_i) = 1/2,\ \forall v_i \in V$, while for HC we consider costs uniformly sampled in $(0, 1)$. To create the underlying graphs G over which the game is played, we use the following synthetic models:

- *Erdős–Rényi* (*ER*): A graph sampled uniformly out of all graphs with N nodes and m_{ER} edges [114]. We use $m_{\text{ER}} = \frac{20}{100} \cdot \frac{N \cdot (N-1)}{2}$, which represents 20% of all possible edges.
- *Barabási–Albert* (*BA*): A growth model where N nodes each attach preferentially to m_{AB} existing nodes [19]. We use $m_{\text{AB}} = 2$.

Algorithm 2 Graph Imitation Learning (GIL).

procedure GIL($\mathcal{G}^{\text{train}}, \mathcal{G}^{\text{validate}}$)
  $\Theta = \text{INITPARAMS}()$ ▷ Randomly initialise policy $\hat{\pi}$ and underlying GNN.
  $\mathcal{D} = \text{COLLECTDATASET}(\mathcal{G}^{\text{train}})$
  while true **do**
    batch $= \text{SAMPLEBATCH}(\mathcal{D})$
    update Θ by SGD(batch, $\mathcal{L}$) ▷ See Equation 5.6 for loss term.
    $\text{CHECKSTOPPINGCRITERION}(\hat{\pi}, \mathcal{G}^{\text{validate}})$ ▷ Evaluate policy on held-out set.
  return policy $\hat{\pi}$

procedure COLLECTDATASET($\mathcal{G}^{\text{train}}$)
  for G in $\mathcal{G}^{\text{train}}$ **do**
    $t = 0$
    $S_t = \text{INITSTATE}(G)$
    while $|\mathcal{A}(S_t)| > 0$ **do**
      $S_{t+1}, A_t, \text{dem} = \text{RUNMCTS}(S_t)$ ▷ Collect demonstration & plan action.
      $\mathcal{D} = \mathcal{D} \cup \{\text{dem}\}$ ▷ Add demonstration to dataset.
      $t+ = 1$
  return $\mathcal{D}$

procedure RUNMCTS(s)
  create root node n_{root} from s
  for $i = 0$ to b_{sims} **do**
    $n_{\text{border}} = \text{TREEPOLICY}(n_{\text{root}})$
    $r = \text{RANDOMDEFAULTPOLICY}(n_{\text{border}})$
    $\text{BACKUP}(n_{\text{border}}, r)$ ▷ Backup reward (proportional to objective $\mathcal{F}$).
  $\mathbf{v} = \text{ARRAY}()$
  for a in $\mathcal{A}(s)$ **do** ▷ Construct vector of visit counts for all actions.
    $\text{APPEND}(\mathbf{v}, C(s, a))$
  $n_{\text{child}} = \text{ROBUSTCHILD}(n_{\text{root}})$
  $s_{\text{next}}, a_{\text{next}} = \text{GETSTATE}(n_{\text{child}}), \text{GETACTION}(n_{\text{child}})$
  dem $= (s, \mathcal{A}(s), N, \mathbf{v})$
  return $s_{\text{next}}, a_{\text{next}}, \text{dem}$

- *Watts–Strogatz* (*WS*): A model designed to capture the small-world property found in many social and biological networks, which generates networks with high clustering coefficient [354]. Starting with a regular ring lattice with N vertices with k_{WS} edges each, edges are rewired to a random node with probability p_{WS}. We use $k_{\text{WS}} = 2$ and $p_{\text{WS}} = 0.1$.

Baselines. We compare to the following baselines:

- *Exhaustive Search* (*ES*): Select the PSNE that maximises the objective out of 2^N possible action profiles. Only applicable on very small graphs due to its computational complexity.

- *Best-Response* (*BR*): In the graph-based best-shot PGG, Best-Response converges to a PSNE [230]. We start from a randomly selected action profile, allow the agents to iteratively play Best-Response, and measure $\mathcal{F}$ once a PSNE is reached.

- *Payoff Transfer* (*PT*): The method of Levit et al. [230] modifies the definition of utilities in this game to include an additional term that represents a payoff. The distributed procedure they propose enables agents to convince their neighbours to switch their action by providing a payoff. As with BR, we start from a randomly selected action profile, and measure $\mathcal{F}$ once a PSNE is reached. Even though the PSNEs reached do not necessarily correspond to mISs, we evaluate the objective functions by considering the utilities of the players.

- *Simulated Annealing* (*SA*): The method proposed by Dall'Asta et al. [90] works by randomly selecting an agent playing 0, incentivising them to switch their action to 1, then iterating on the BR rule. The new PSNE reached is either accepted or rejected based on a simulated annealing rule with a certain temperature parameter τ_{SA}. In the limit of infinite time, this method converges to the optimal Nash Equilibrium in terms of social welfare in the IC case.

We also consider the additional baselines listed below, which exploit the mIS connection. We note that TH and TLC have not been considered in prior work but are potentially effective heuristics.

- *Random*: Incrementally construct an mIS by randomly picking, at each step, a node that is not in the independent set and is not adjacent to any nodes in the independent set.

Table 5.1: Mean rewards obtained by the methods split by cost setting, graph model, and objective function.

c	$\mathcal{G}$	$\mathcal{F}$	Random	TH	TLC	BR	PT	SA	UCT	GIL (ours)
HC	BA	$\mathcal{F}_{GF}$	0.745$\pm$0.005	0.802	0.774	0.742$\pm$0.004	0.791$\pm$0.015	0.815$\pm$0.000	**0.837**$\pm$0.000	0.834$\pm$0.001
		$\mathcal{F}_{SW}$	0.697$\pm$0.007	0.779	0.727	0.691$\pm$0.006	0.760$\pm$0.019	0.795$\pm$0.000	**0.815**$\pm$0.000	0.813$\pm$0.000
	ER	$\mathcal{F}_{GF}$	0.877$\pm$0.001	0.896	0.920	0.877$\pm$0.000	0.911$\pm$0.002	0.908$\pm$0.001	**0.945**$\pm$0.000	0.940$\pm$0.003
		$\mathcal{F}_{SW}$	0.868$\pm$0.001	0.890	0.912	0.867$\pm$0.000	0.903$\pm$0.002	0.903$\pm$0.001	**0.940**$\pm$0.000	0.935$\pm$0.001
	WS	$\mathcal{F}_{GF}$	0.803$\pm$0.002	0.806	0.865	0.804$\pm$0.002	0.821$\pm$0.003	0.832$\pm$0.001	**0.892**$\pm$0.000	**0.892**$\pm$0.000
		$\mathcal{F}_{SW}$	0.781$\pm$0.002	0.785	0.846	0.782$\pm$0.003	0.800$\pm$0.004	0.817$\pm$0.001	**0.876**$\pm$0.000	**0.876**$\pm$0.000
IC	BA	$\mathcal{F}_{GF}$	0.833$\pm$0.000	0.844	—	0.834$\pm$0.000	0.841$\pm$0.005	**0.849**$\pm$0.000	0.847$\pm$0.000	0.847$\pm$0.000
		$\mathcal{F}_{SW}$	0.697$\pm$0.007	0.779	—	0.691$\pm$0.006	0.757$\pm$0.019	0.794$\pm$0.000	**0.795**$\pm$0.000	**0.795**$\pm$0.000
	ER	$\mathcal{F}_{GF}$	0.893$\pm$0.000	0.906	—	0.892$\pm$0.000	0.907$\pm$0.001	0.916$\pm$0.000	**0.922**$\pm$0.000	0.919$\pm$0.002
		$\mathcal{F}_{SW}$	0.867$\pm$0.000	0.889	—	0.866$\pm$0.001	0.889$\pm$0.002	0.903$\pm$0.000	**0.910**$\pm$0.000	0.908$\pm$0.001
	WS	$\mathcal{F}_{GF}$	0.842$\pm$0.001	0.843	—	0.842$\pm$0.001	0.847$\pm$0.001	0.856$\pm$0.000	0.862$\pm$0.000	**0.864**$\pm$0.000
		$\mathcal{F}_{SW}$	0.777$\pm$0.002	0.782	—	0.779$\pm$0.003	0.791$\pm$0.004	0.813$\pm$0.001	0.824$\pm$0.000	**0.828**$\pm$0.000

- *Target Hubs* (*TH*): Pick the highest-degree node available that is not yet included in the independent set. While this strategy is not guaranteed to find a global maximum, placing the public good on nodes with many connections means that many others can access it.

- *Target Lowest Cost* (*TLC*): Place the public good on nodes for which the cost $c(v)$ is lowest (only applicable in HC case). This may result in equilibria with high social welfare and fairness since the good is acquired only by those players for which it costs little to do so.

Training and Evaluation Protocol. Evaluation (and training, where applicable) is performed separately for each N, graph model, objective $\mathcal{F}$, and cost setting (IC or HC). We aggregate results from 10 different random seeds for stochastic approaches. Further details are provided in Appendix C.

5.4 Evaluation Results

We show the main results obtained in Table 5.1, in which entries are aggregated across games with a number of players $N \in \{15, 25, 50, 75, 100\}$. Each reported value corresponds to the average objective function value of an equilibrium of the graph-based best-shot Public Goods Game. In Appendix C, extended versions of these results are shown in Table C.1. The reported values are separated by the number of players N. We also include an additional analysis that evaluates the win rate percentage for each of the methods instead of average reward, since the average reward metric may be sensitive to outliers (i.e., game instances with abnormally

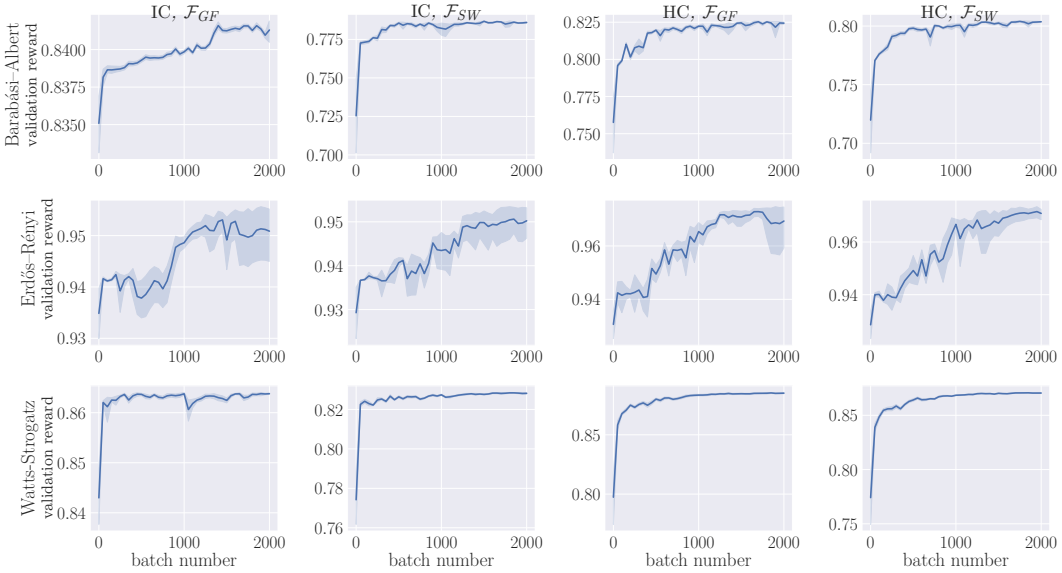


Figure 5.3: Training curves for GIL, showing performance on the held-out validation set.

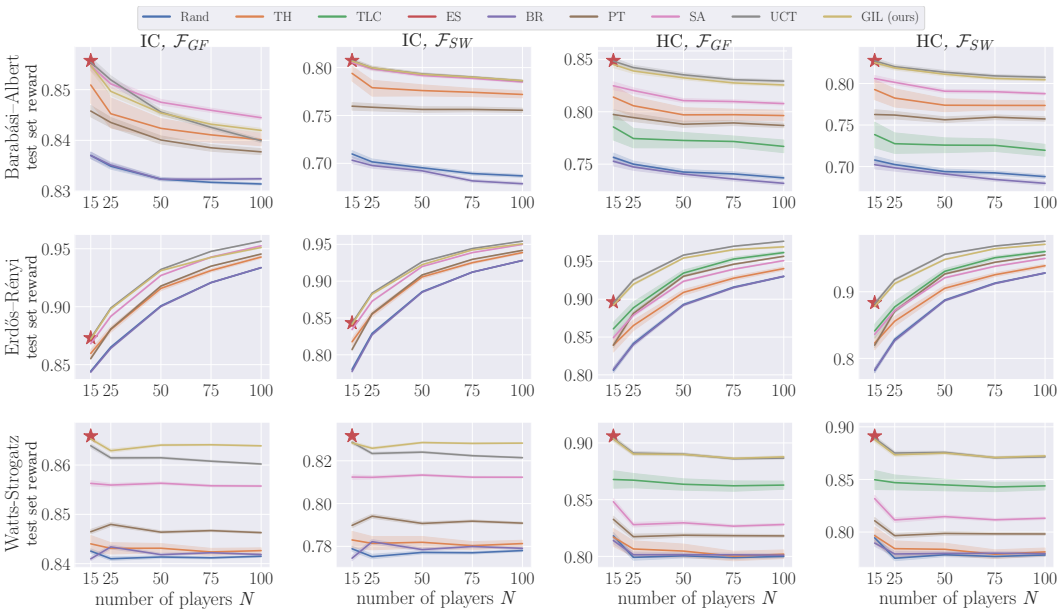


Figure 5.4: Mean rewards obtained by the methods as a function of the number of players N.

large or small objective function values). This is shown in Table C.2, with ties being broken randomly in case there is more than one winner.

Performance obtained during training on the held-out validation set of instances with $N = 100$ is shown in Figure 5.3. The performance on the test set is shown in Figure 5.4, in which the x-axes represent the number of players. We also show a comparison of the runtime per episode in milliseconds used by the different methods in Figure 5.6. In the figures, the stars at $N = 15$ represent Exhaustive Search.

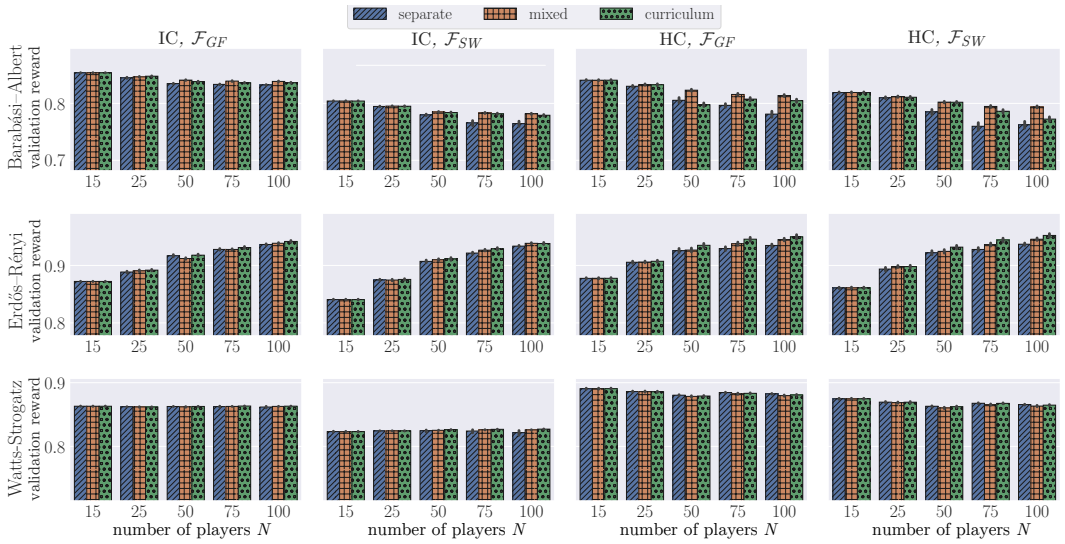


Figure 5.5: Mean rewards obtained on the validation set by GIL using different training procedures. The choice of an appropriate procedure depends on the underlying graph structure. We observe that it is more important in the HC case and it increases in importance with graph size.

We find that the UCT planning method outperforms previous methods in all cases except for $IC, \mathcal{F}_{GF}$ in which the SA baseline does better as game size increases. For the smallest graphs, UCT nearly performs on par with Exhaustive Search. The Random and BR baselines consistently perform the poorest, as expected. The gap between the methods enabled by our approach (UCT and GIL) are higher in the HC settings where costs for acquiring the public good differ between players.

Impact of Training Strategies. Additionally, we also explore the impact of the training strategies used for the Imitation Learning phase in Figure 5.5. Since the choice of which examples to present to the model as well as their order could have an impact on the performance of the policy, together with the fact there is no *a priori* knowledge of which strategy is optimal, we treat the training strategy as a tunable hyperparameter. We find that no training strategy is better across the board; rather, their rankings are consistent depending on the graph model on which the method is trained. For BA graphs, the mixed strategy performs best; in the case of ER graphs the curriculum strategy achieves the highest reward; for WS the differences between the averages are very small. Additionally, for BA and ER graphs, the relative differences between the methods are higher in the HC setting than in IC. In all cases, the difference in mean reward between the training strategies is insignificant on small graphs, and increases in importance the larger the size.

Imitation Learned Policy. We find that the performance of the policy learned with GIL is within 99.5% of that of the planning method, even substantially exceeding it in certain cases (e.g., in the IC setting for Watts-Strogatz graphs and both objectives). Given the timings in Figure 5.6, this closeness in performance is even more remarkable since the GIL policy is approximately three orders of magnitude cheaper to evaluate than the planning method on the largest graphs tested.

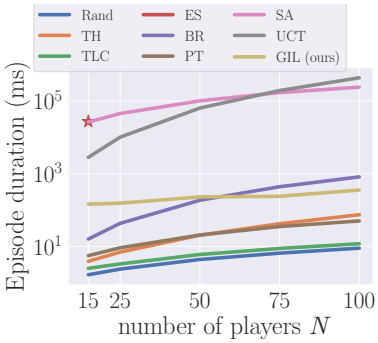


Figure 5.6: Mean milliseconds needed to complete an episode (i.e., construct an mIS) as a function of the number of players.

5.5 Discussion

Limitations. Since the proposed method exploits the connection with the mIS substructure, it is only applicable for the class of games for which it holds, and cannot be directly applied to a wider class of networked Public Goods Games. In addition, given that this method assumes a centralised perspective, it is only applicable in situations in which the network structure is known or can be reasonably inferred. Even though the work in this chapter answers the question of what outcomes *could* be reached by self-interested players, the design of mechanisms to move towards such configurations remains a challenging problem. Possible incentivisation mechanisms that we aim to explore in future work include incentivising individual players as well as modifying the network structure itself. Since the impact of such interventions on players' actions can be captured by Best-Response dynamics, this can be formulated as a search problem in which the goal is to find the set of interventions that brings us arbitrarily close to the goal state. At a high level, an approach similar to the proposed method, which leverages a different deterministic MDP model capturing the impact of interventions, may be used.

Societal Impact and Implications. The direct implication of this chapter is that our approach enables a social planner to find beneficial outcomes that can be achieved and maintained by self-interested players for situations that can be modelled by networked best-shot games. This class of games is relevant for a variety of scenarios in which the agents forming a society can choose to contribute effort to a public good, and our work is motivated by positive societal impact. We cannot foresee

situations in which this method can be directly misused. This relies, however, on the assumption that the social planner aims to improve outcomes. Furthermore, this type of modelling is necessarily abstract and makes simplifying assumptions that may not hold in the real world.

5.6 Summary

In this chapter, we have endeavoured to find desirable equilibria of the networked best-shot Public Goods Game, an application of the general problem of finding an mIS that optimises an objective function, as expressed in ***RQ4***. We have approached this from the perspective of a principal agent with global knowledge of the game that aims to find optimal Pure Strategy Nash Equilibria in terms of social welfare and fairness of outcomes. We have defined an MDP to find desirable mIS substructures in graphs, and shown that using the UCT algorithm to plan in this MDP yields better results than existing approaches, especially in the case where the costs of acquiring the public good differ between players. To address ***RQ5***, we have also proposed a Graph Imitation Learning method which is able to learn the structure of these equilibria, yielding performance within 99.5% of the planning method while generalising to different game instances and generating predictions approximately three orders of magnitude quicker than UCT on the largest graphs tested.

The proposed method is directly applicable to other settings in which mIS are of interest (see, e.g., [89]). More broadly, the method for performing planning and Imitation Learning presented in this chapter can be used for a variety of problems that can be formulated as a decision-making process on a graph with the goal of maximising a given objective function. Areas in which this may be of interest include combinatorial optimisation and algorithmic reasoning over graphs [31, 58], provided that the horizon for the task is manageable by a search procedure. While we have focussed on constructing generalisable models, if predictions need to be made for a single graph instance, one could also consider combining the planning and Imitation Learning steps similarly to the ExIt algorithm [11, 315]. This chapter is related to other recent work that considers learning in network games (e.g., [337] treats network *emergence* games) as well as more broadly to ongoing efforts in the area of cooperation in multi-agent systems and its impact on societal problems [84].

Chapter 6

Graph Neural Modelling of Network Flows

In this chapter, we depart from the model pursued so far of formulating combinatorial optimisation problems as MDPs. Instead, we focus on *learning representations*, an important component of data-driven methods for solving them. To isolate concerns, we adopt a Supervised Learning setting, which involves predicting the objective function outcome when applying known algorithms. Particularly, we focus on Multi-Commodity Network Flow problems, which involve distributing traffic over a network such that the infrastructure is used efficiently.

Due to their ubiquity in transportation and logistics, together with the appeal of data-driven optimisation, MCNF scenarios have increasingly been approached using graph learning methods. However, we hypothesise that the global message function typically used in such works constrain the routing unnecessarily. To address this issue, we propose a learning representation based on GNNs that uses distinctly parametrised message functions along each link, akin to a relational model where all edge types are unique. We extensively evaluate the proposed approach through an Internet routing case study using 17 Service Provider topologies and two flow routing schemes.

6.1 Introduction

Flow routing represents a fundamental problem that captures a variety of optimisation scenarios that arise in real-world networks [6, Chapter 17]. One classic example is the maximum flow problem, which seeks to find the best (in terms of maximum capacity) path between a source node and a sink node. The more general Multi-Commodity Network Flow problem allows for multiple flows of different sizes between several sources and sinks that share the same distribution network. Amongst other things, it serves as a formalisation of the distribution of packets in a computer network, of goods in a logistics network, or cars in a rail network [180]. To aid understanding, we illustrate the MCNF family of problems in Figure 6.1.

For maximum flow problems, efficient algorithms have been developed [80, Chapter 24], including a recent near-linear time approach [66]. For the more complex MCNF problems, Linear Programming solutions can be leveraged in order to compute, in polynomial time, the optimal routes given knowledge of pairwise demands between the nodes in the graph [124, 331]. At the other end of the spectrum, oblivious routing methods derive routing strategies with partial or no knowledge of traffic demands, optimising for "worst-case" performance [285]. As recognised by existing works, *a priori* knowledge of the full demand matrix is an unrealistic assumption, as loads in real systems continuously change. Instead, ML techniques may enable a middle ground [339]: learning a model trained on past loads that can perform well in a variety of traffic scenarios, without requiring a disruptive redeployment of the routing strategy [125].

From a more practical point of view, this shift towards data-driven approaches is illustrated by the concepts of data-driven computer networking [189] and self-driving networks [120]. Early works in this area were based on MLP architectures [339, 287]. More recently, models purposely designed to operate on graphs, including variants of the expressive Message Passing Neural Networks [300, 10] and Graph Nets [24], have been adopted.

In particular, we focus on the prediction of the maximum utilisation among all links in a network, given traffic demands and a predefined routing strategy. The requirements are quantified using a matrix that specifies the amount of traffic between pairs of nodes in the graph. The underlying network infrastructure is

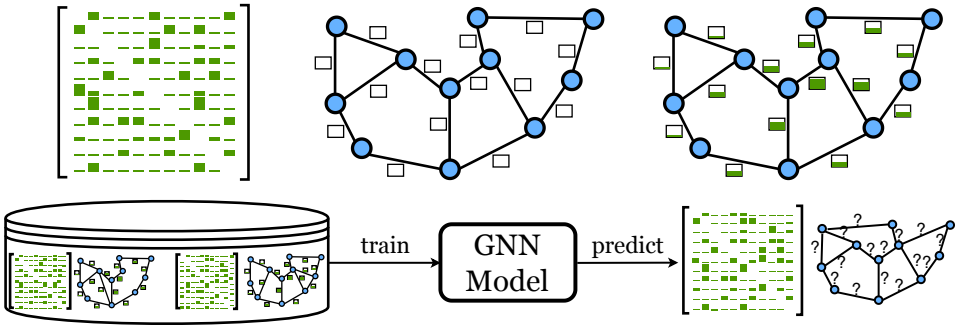


Figure 6.1: Top. An illustration of the Multi-Commodity Network Flow family of problems. The requirements of the routing problem are defined using a matrix that specifies the total amount of traffic that has to be routed between each pair of nodes in a graph. We are also given a graph topology in which links are equipped with capacities. All flows have an entry and exit node and share the same underlying transportation infrastructure. Under a particular routing scheme, such as shortest path routing, the links are loaded by the total amount of traffic passing over them. **Bottom.** A model is trained using a dataset of the link utilisations for certain demand matrices and graph topologies. This model is then used to predict the Maximum Link Utilisation for an unseen demand matrix.

given using a weighted graph, where each weight represents the capacity of a link. Under a given routing scheme, such as shortest path routing, the links are loaded by the total amount of traffic passing over them. The learning task is to predict, given a routing scheme and a dataset of demand matrices and loads, the Maximum Link Utilisation for an unseen demand matrix. While the link utilisations can be computed analytically given a particular demand matrix, the goal is to obtain one model that can predict them accurately for several demand matrices with a single set of parameters.

In this chapter, we propose a novel approach based on GNNs for the MCNF problem. The intuition is that a GNN may perform better than structure-agnostic learning architectures such as the MLP due to the *algorithmic alignment* [367] between the computational mechanism of GNNs and the task itself. In the context of this problem, the message-passing mechanism is akin to receiving the flows from neighbouring nodes in one round, then deciding how to split the flows among the neighbours in the subsequent round. Despite the fact that graph learning methods show promise in this space, current works nevertheless adopt schemes that aggregate messages along neighbouring edges using the same weight vectors. In the context of routing flows over graphs, this constrains the model unnecessarily. Instead, we argue that nodes should be able to weight flows along each link separately, so that

each node may independently update its state given incoming and outgoing traffic. We illustrate this in Figure 6.2.

Furthermore, the ways in which prior works encode the demands as node features varies between the full demand matrix [339, 380] and a node-wise summation [175], and it is unclear when either is beneficial. Besides the learning representation aspects, existing approaches in this area are evaluated using very few graph topologies (typically 1 or 2) of small sizes (typically below 20 nodes). This makes it difficult to assess the gain that graph learning solutions bring over vanilla architectures such as the MLP. Additionally, a critical point that has not been considered is the impact of the underlying graph topology on the effectiveness of the learning process. To address these shortcomings, we make a series of contributions along the following axes:

- **Learning representations for data-driven flow routing.** We propose a novel mechanism for aggregating messages along each link with a different parametrisation, which we refer to as *Per-Edge Weights* (*PEW*). This is equivalent to a relational method such as R-GCN in which every edge is given a different relation type and is compatible with any relational architecture. Despite its simplicity, we show that this mechanism yields substantial predictive gains over architectures that use the same message function for all neighbours. We also find that equipping a GNN with PEW can exploit the complete demand matrix as node features, while standard methods perform better with the lossy node-wise sum used in prior work.

- **Rigorous evaluation and systematic comparison of existing approaches.** Whereas existing works test on few, small-scale topologies, we evaluate the proposed method and related baselines on 17 real-world Internet Service Provider topologies and 2 different routing schemes in the context of a case study in computer networks. Perhaps surprisingly, we find that a well-tuned MLP often outperforms a vanilla GNN architecture without PEW when the methods are given equal hyperparameter and training budgets.

- **Understanding the impact of topology.** The range of experiments we carry out allows us to establish that a strong link exists between topology and the

Figure 6.2: Left. An illustration of the MPNN used in previous flow routing works, which uses the same message function $M^{(l)}$ for aggregating neighbour messages. **Right.** An illustration of our proposed Per-Edge Weights (PEW), which uses uniquely parametrised per-edge message functions.

difficulty of the prediction task, which is consistent across routing schemes. Generally, the predictive performance decreases with the size of the graph (in terms of number of nodes, diameter and edge density), but it increases the more heterogeneous the graph is in terms of capacities, degrees and betweenness of the nodes. Moreover, we find that, when graph structure varies through the presence of different subsets of nodes, the predictive performance of the GNN-based methods increases compared to simpler, structure-agnostic methods, such as MLP.

6.2 Methods

6.2.1 Routing Formalisation and Learning Task

Flow routing formalisation. We assume the splittable-flow routing formalisation proposed by Fortz and Thorup [126]. We let $G = (V, E)$ be a directed graph, with V representing the set of nodes and E the set of edges. We use $N = |V|$ and $m = |E|$ as shorthands, as well as v_i and $e_{i,j}$ to denote specific nodes and edges, respectively. Each edge has an associated *capacity* $\kappa(e_{i,j}) \in \mathbb{R}^+$. We also define a *demand matrix* $\mathbf{D} \in \mathbb{R}^{N \times N}$ where entry $D_{src,dst}$ is the traffic that source node src sends to destination dst. With each tuple $(src, dst, e_{i,j}) \in V \times V \times E$ we associate the quantity $f_{e_{i,j}}^{(src,dst)} \geq 0$, which specifies the amount of traffic flow from src to dst that goes over the edge $e_{i,j}$. The *load* of edge $e_{i,j}$, load$(e_{i,j})$, is the total traffic flow traversing it, i.e., load$(e_{i,j}) = \sum_{(src,dst) \in V \times V} f_{e_{i,j}}^{(src,dst)}$. Furthermore, the quantities $f_{e_{i,j}}^{(src,dst)}$ must obey the following flow conservation constraints:

$$\sum_{e \in \delta^+(v_i)} f_e^{(src,dst)} - \sum_{e \in \delta^-(v_i)} f_e^{(src,dst)} = \begin{cases} D_{src,dst} & \text{if } v_i = src, \\ -D_{src,dst} & \text{if } v_i = dst, \\ 0 & \text{otherwise.} \end{cases} \tag{6.1}$$

where the sets $\delta^+(v_i), \delta^-(v_i)$ are node v_i's outgoing and incoming edges respectively. Intuitively, these constraints capture the fact that traffic sent from src to dst originates at the source (first clause), must be absorbed at the target (second clause), and ingress equals egress for all other nodes (final clause).

Routing schemes. A routing scheme $\mathscr{R}$ specifies how to distribute the traffic flows. Specifically, we consider two well-known routing schemes. The first is the *Standard Shortest Paths* (SSP) scheme in which, for a given node, the full flow quantity with destination dst is sent to the neighbour on the shortest path to dst. The widely used *ECMP* scheme [177] instead splits outgoing traffic among all the neighbours on the shortest path to dst if multiple such neighbours exist.

Prediction target. A common way of evaluating a routing strategy $\mathscr{R}$ is *Maximum Link Utilisation* (*MLU*), i.e., the maximal ratio between link load and capacity. Formally, given a demand matrix $\mathbf{D}$, we denote it as $\text{MLU}(\mathbf{D}) = \max_{e_{i,j} \in E} \frac{\text{load}(e_{i,j})}{\kappa(e_{i,j})}$. This target metric has been extensively studied in prior work [195] and is often used by ISPs to gauge when the underlying infrastructure needs to be upgraded [151].

Supervised learning setup. We assume that we are provided with a dataset of traffic matrices $\mathcal{D} = \cup_k \{\mathbf{D}^{(k)}, \text{MLU}(\mathbf{D}^{(k)})\}$. Given that our model produces an approximation $\widehat{\text{MLU}}(\mathbf{D}^{(k)})$ of the true Maximum Link Utilisation, the goal is to minimise the Mean Squared Error $\frac{\sum_k (\text{MLU}(\mathbf{D}^{(k)}) - \widehat{\text{MLU}}(\mathbf{D}^{(k)}))^2}{|\mathcal{D}|}$.

6.2.2 Per-Edge Weights

We propose a simple mechanism to increase the expressivity of models for data-driven flow routing. As previously mentioned, several works in recent years have begun adopting various graph learning methods for flow routing problems such as variants of Message Passing Neural Networks [142, 300, 10] or Graph Networks [175]. Recall, as described in Section 2.4.3, that MPNNs derive hidden features $\mathbf{h}_{v_i}^{(l)}$ for node v_i in layer $l+1$ by computing messages $\mathbf{m}^{(l+1)}$ and applying updates of the form:

$$\begin{aligned}\mathbf{m}_{v_i}^{(l+1)} &= \sum_{v_j \in \mathcal{N}(v_i)} M^{(l)}\left(\mathbf{h}_{v_i}^{(l)}, \mathbf{h}_{v_j}^{(l)}, \mathbf{x}_{e_{i,j}}\right) \\ \mathbf{h}_{v_i}^{(l+1)} &= U^{(l)}\left(\mathbf{h}_{v_i}^{(l)}, \mathbf{m}_{v_i}^{(l+1)}\right)\end{aligned} \tag{6.2}$$

where $\mathcal{N}(v_i)$ is the neighbourhood of node v_i, $\mathbf{x}_{e_{i,j}}$ are edge features for edge $e_{i,j}$, and $M^{(l)}$ and $U^{(l)}$ are the differentiable message (sometimes also called edge) and vertex update functions in layer l. Typically, $M^{(l)}$ is some form of MLP that is applied in parallel when computing the update for each node in the graph. An advantage of applying the same message function $M^{(l)}$ across the entire graph is that the number of parameters remains fixed in the size of the graph, enabling a form of combinatorial generalisation [24]. However, while this approach has been very successful in many graph learning tasks such as graph classification, we argue that it is not best suited for flow routing problems.

Instead, for this family of problems, the edges do not have uniform semantics. Each of them plays a different role when the flows are routed over the graph and, as shown in Figure 6.1, each will take on varying levels of load. Equivalently, from a node-centric perspective, each node should be able to decide flexibly how to distribute several flows of traffic over its neighbouring edges. This intuition can be captured by using a different message function $M_{i,j}^{(l)}$ when aggregating messages received along each edge $e_{i,j}$. We call this mechanism *Per-Edge Weights*, or *PEW*. We illustrate the difference between PEW and a typical MPNN in Figure 6.2.

Even though the applications are unrelated, similar graph learning techniques have been applied for modelling knowledge bases, another area in which the different semantics of edges play an important role. Knowledge bases are often modelled as graphs in which edges between nodes (entities) are labelled with types out of a set of possible relations $\mathcal{X}$. Relational graph learning architectures, such as the RGCN, use a different message function $M_{\chi}^{(l)}$ for each relation type χ. Hence, we can draw a correspondence with relational models: our proposed approach is equivalent to a relational graph learning model in which there exists a relation type for each edge, i.e., $\mathcal{X} \equiv E$. This correspondence lets us exploit existing relational architectures and implementations. For completeness, let us now describe a possible realisation of PEW by drawing an analogy to the RGCN relational architecture, noting that PEW is compatible with any other relational model (and, indeed, in our experiments we

use the more complex RGAT since it supports edge features). Each layer applies the update rule:

$$\mathbf{h}_{v_i}^{(l+1)} = \text{ReLU}\left(\sum_{v_j \in \mathcal{N}(v_i)} \frac{1}{\mu_{i,j}} \mathbf{W}_{i,j}^{(l)} \mathbf{h}_{v_j}^{(l)} + \mathbf{W}_0^{(l)} \mathbf{h}_{v_i}^{(l)}\right) \tag{6.3}$$

where $\mathcal{N}(v_i)$ denotes the set of neighbours of v_i, $\mathbf{W}_{i,j}^{(l)}$ is an edge-specific weight matrix, $\mu_{i,j}$ is a problem-specific normalisation constant that can be fixed or learned, and ReLU may be replaced by another non-linearity.

A possible disadvantage of PEW is that the number of parameters grows linearly with the edge count. However, given the relatively small scale of the typical network considered in such problems (several hundreds of nodes), in practice the impact in terms of memory usage or execution time has not been significant in our experiments: the largest RGAT+PEW model, used for the Uninett2011 graph, has approximately $800,000$ parameters. Furthermore, solutions have already been developed and validated for the much larger-scale relational graphs with millions of nodes and thousands of relations, such as the basis and block-diagonal decompositions proposed in [305], which can help in keeping the number of parameters low. Other routing-specific options that may be investigated in future work could be the "clustering" of the edges depending on the structural roles that they play (such as peripheral or backbone links) or the use of differently parametrised neighbourhoods for the regions of the graph, which may perform well in case a significant proportion of the traffic is local.

6.3 Evaluation Protocol

This section describes the experimental setup we use for our evaluation. We focussed on a case study on routing flows in computer networks to demonstrate its effectiveness in real-world scenarios, which can be considered representative of a variety of settings in which we wish to predict the properties of a routing scheme from an underlying network topology and a set of observed demand matrices.

Model architectures. Due to the importance of the link capacities for the prediction objective, we opt for the RGAT [55] architecture, which supports edge features. To make sure that implementations are aligned, we compare RGAT+PEW with the standard RGAT with a single relation type, which is equivalent to a GAT [343].

We also compare against a standard MLP architecture made up of fully-connected layers followed by ReLU activations. The features provided as input to the three methods are the same: for the GNN methods, the node features are the demands $\mathbf{D}$ in accordance with the demand input representations defined later in this section, while the edge features are the capacities κ, and the adjacency matrix $\mathbf{A}$ governs the message passing. For the MLP, we unroll and concatenate the demand input representation derived from $\mathbf{D}$, the adjacency matrix $\mathbf{A}$, and all edge capacities κ in the input layer. We note that other non-ML baselines, such as Linear Programming, are not directly applicable for this task: while they can be used to derive a routing strategy, in this chapter the goal is to predict a property of an existing routing strategy (SSP or ECMP, as defined in Section 6.2.1).

Traffic generation. In order to generate synthetic flows of traffic, we use the "gravity" approach proposed by Roughan [297]. Akin to Newton's law of universal gravitation, the traffic $D_{i,j}$ between nodes v_i and v_j is proportional to the amount of traffic, D_i^{in}, that enters the network via v_i and D_j^{out}, the amount that exits the network at v_j. The values D_i^{in} and D_j^{out} are random variables that are identically and independently distributed according to an exponential distribution. Despite its simplicity in terms of number of parameters, this approach has been shown to synthesise traffic matrices that correspond closely to those observed in real-world networks [297, 162]. We additionally apply a rescaling of the volume by the MLU (defined in Section 6.2.1) under the LP solution of the MCNF formulation, as recommended in the networking literature [157, 154].

Network topologies. We consider real-world network topologies that are part of the Repetita and Internet Topology Zoo repositories [138, 212]. In case there are multiple snapshots of the same network topology, we only use the most recent so as not to bias the results towards these graphs. We limit the size of the considered topologies to between $[20, 100]$ nodes, which we note is still substantially larger than topologies considered in prior work on ML for routing flows. Furthermore, we only consider heterogeneous topologies with at least two different link capacities. Given the traffic model above, for some topologies the MLU dependent variable is nearly always identical regardless of the demand matrix, making it trivial to devise a good predictor. Out of the 39 resulting topologies, we filter out those for which

the minimum MLU is equal to the 90th percentile MLU over 100 demand matrices, leaving 17 unique topologies. The properties of these topologies are summarised in Table 6.1 in the Appendix. For the experiments in Section 6.4.2, we use variations in the original topology. These variations are generated as follows: a number of nodes to be removed from the graph is chosen uniformly at random in the range $[1, \frac{N}{5}]$, subject to the constraint that the graph does not become disconnected. Demand matrices are generated starting from this modified topology.

Table 6.1: Properties of the topologies.

Graph	N	m	Diameter	$\frac{m}{N}$	Flows in $\mathcal{D}$
Aconet	23	62	4	2.70	1587000
Agis	25	60	7	2.40	1875000
Arnes	34	92	7	2.71	3468000
Cernet	41	116	5	2.83	5043000
Cesnet201006	52	126	6	2.42	8112000
Grnet	37	84	8	2.27	4107000
Iij	37	130	5	3.51	4107000
Internode	66	154	6	2.33	13068000
Janetlense	20	68	4	3.40	1200000
Karen	25	56	7	2.24	1875000
Marnet	20	54	3	2.70	1200000
Niif	36	82	7	2.28	3888000
PionierL3	38	90	10	2.37	4332000
Sinet	74	152	7	2.05	16428000
SwitchL3	42	126	6	3.00	5292000
Ulaknet	82	164	4	2.00	20172000
Uninett2011	69	192	9	2.78	14283000

Datasets. The datasets $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{validate}}$, $\mathcal{D}^{\text{test}}$ of demand matrices are disjoint and contain 10^3 demand matrices each. Both the demands and capacities are standardised by dividing them by the maximum value across the union of the datasets.

Demand input representation. We also consider two different demand input representations that appear in prior work, which we term *raw* and *sum*. In the former, the feature vector $\mathbf{x}_{v_i}^{\text{raw}} \in \mathbb{R}^{2N}$ for node v_i is $[D_{1,i}, \ldots, D_{N,i}, D_{i,1}, \ldots, D_{i,N}]$, which corresponds to the concatenated outgoing and incoming demands respectively. The latter is an aggregated version $\mathbf{x}_{v_i}^{\text{sum}} \in \mathbb{R}^2$ equal to $[\sum_j D_{i,j}, \sum_i D_{j,i}]$, i.e., it contains the summed demands.

Training and evaluation protocol. Training and evaluation are performed separately for each graph topology and routing scheme. To compute means and confidence intervals, we repeat training and evaluation across 10 different random seeds. Train-

Figure 6.3: Normalised MSE obtained by the predictors on different topologies for the SSP (top) and ECMP (bottom) routing schemes. Lower values are better. PEW improves performance over vanilla GAT substantially. An MLP outperforms the graph learning method without PEW in 80% of cases.

ing is done by mini-batch SGD using the Adam optimiser [207] and proceeds for 3000 epochs with a batch size of 16. We perform early stopping if the validation performance does not improve after 1500 epochs, also referred to as "patience" in other graph learning works [343, 116]. Since the absolute value of the MLUs varies significantly in datapoints generated for different topologies, we apply a normalisation when reporting results such that they are comparable. Namely, the MSE of the predictors is normalised by the MSE of a simple baseline that outputs the average MLU for all DMs in the provided dataset. We refer to this as Normalised MSE (NMSE).

6.4 Evaluation Results

6.4.1 Benefits of PEW for Flow Routing

The primary results are shown in Figure 6.3, which compares the normalised MSE obtained by the three architectures (RGAT+PEW, RGAT and MLP) for the 17 topologies. The two rows correspond to the SSP and ECMP schemes respectively.

We find that PEW improves the predictive performance over a vanilla RGAT in nearly all (88%) of the settings tested, and that it performs the best out of all predictors in 76% of topologies. As later shown in Table 6.2, the latter figure rises to 82% when different subsets of the nodes are present and generating demands.

Hence, this highlights the importance of parametrising links differently, suggesting that it is an effective inductive bias for this family of problems. Interestingly, when excluding PEW, the MLP performs better than RGAT in 80% of the considered cases. This echoes findings in other graph learning works [116], i.e., the fact that a well-tuned MLP can be competitive with GNN architectures and even outperform them. Furthermore, both the relative differences between predictors and their absolute normalised MSEs are fairly consistent across the different topologies.

6.4.2 Varying Graph Structure

Next, we investigate the impact of variations in topology on the predictive performance of the models. In this experiment, the sole difference to the setup described above is that the datasets contain 10^3 demand matrices that are instead distributed on 25 variations in topology of the original graph (i.e., we have 40 DMs per variation making up each dataset). To evaluate how the different methods rank, we use two metrics: the Win Rate (WR) is the percentage of topologies for which the method obtains the lowest normalised MSE, and the Mean Reciprocal Rank (MRR) is the arithmetic average of the complements of the ranks of the three predictors. For both metrics, higher values are better. Results are shown in Table 6.2. We find that the relative performance of the GNN-based methods increases while that of the MLP decreases when varying subsets of the nodes in the original graph are present. This suggests that GNN-based approaches are more resilient to changes in graph structure (e.g., nodes joining and leaving the network), a common phenomenon in practice.

Table 6.2: Mean Reciprocal Rank and Win Rates for the different predictors. The performance of the GNN-based approaches increases relative to the MLP when the graph structure varies by means of different subsets of nodes being present and generating demands.

			MLP	RGAT	RGAT+PEW
$\mathscr{R}$	metric	Graph			
SSP	MRR ↑	Original	0.588	0.382	**0.863**
		Variations	0.520	0.412	**0.902**
	WR ↑	Original	23.529	0.000	**76.471**
		Variations	11.765	5.882	**82.353**
ECMP	MRR ↑	Original	0.578	0.392	**0.863**
		Variations	0.500	0.422	**0.912**
	WR ↑	Original	23.529	0.000	**76.471**
		Variations	11.765	5.882	**82.353**

6.4.3 Best Demand Input Representation

To compare the two demand input representations, we additionally train the model architectures on subsets of $5\%, 10\%, 25\%$ and 50% of the datasets. Recall that the *raw*

representation contains the full demand matrix while the *sum* representation is a lossy aggregation of the same information. The latter may nevertheless help to avoid overfitting and, given that the distribution of the demands is exponential, the largest flows will dominate the features.

Results are shown in Figure 6.4. The x-axis indicates the number of demand matrices used for training and evaluation, while the y-axis displays the difference in normalised MSE between the *raw* and *sum* representations, averaged across all topologies. As marked in the figure, $y > 0$ means that the raw representation performs better, while the reverse is true for $y < 0$. With very few datapoints, the two input representations yield similar errors for both RGAT+PEW and GAT. Beyond this, two interesting trends emerge: as the number of datapoints increases, RGAT+PEW performs better with the raw demands, while the vanilla GAT performs better with the lossy representation. This suggests that, while the RGAT+PEW model is able to exploit the granular information in the raw demands, they instead cause the standard RGAT to overfit and obtain worse generalisation performance.

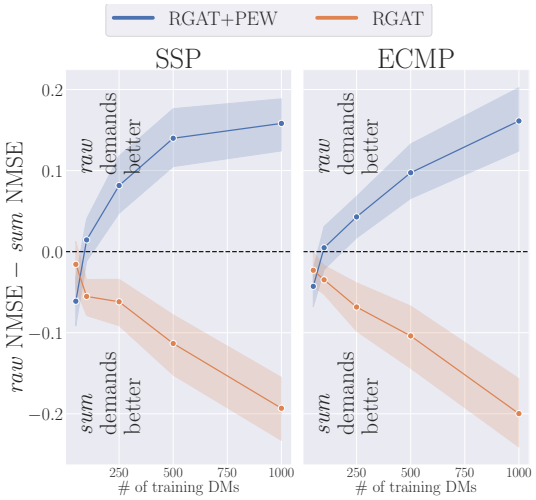


Figure 6.4: Difference in normalised MSE between the *raw* and *sum* demand input representations as a function of the number of training datapoints for RGAT+PEW and RGAT for the SSP (left) and ECMP routing schemes (right). As the dataset size increases, RGAT+PEW is able to exploit the granular demand information, while RGAT performs better with a lossy aggregation of the demand information.

6.4.4 Impact of Topology

Our final set of experiments examines the relationship between the topological characteristics of graphs and the relative performance of our proposed model architecture. The six properties that we examine are defined as follows, noting that the first three are global properties while the final three measure the variance in local node and edge properties.

- **Number of nodes**: the cardinality N of the node set V;
- **Diameter**: the maximum length among all pairwise shortest paths;

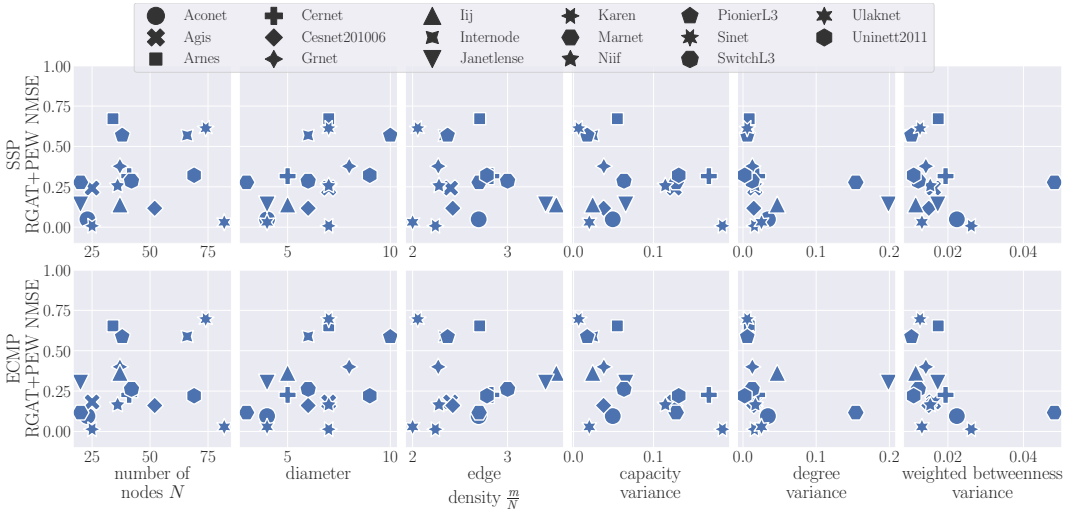


Figure 6.5: Impact of topological characteristics on the predictive performance of RGAT+PEW. Performance degrades as the graph size increases (first 3 columns), but improves with higher levels of heterogeneity of the graph structure (last 3 columns).

- **Edge density**: the ratio of links to nodes $\frac{m}{N}$;
- **Capacity variance**: the variance in the normalised capacities $\kappa(e_{i,j})$;
- **Degree variance**: the variance in the degree centralities $\frac{\deg(v_i)}{N}$;
- **Weighted betweenness variance**: the variance in the values of a weighted version of betweenness centrality [47], which measures the fraction of shortest paths between all pairs of nodes in the network passing through each node.

The results of this analysis are shown in Figure 6.5. As previously, the normalised MSE of the RGAT+PEW model is shown on the y-axis, while the x-axis measures properties of the graphs. Each datapoint represents one of the 17 topologies. Additionally, to complement the absolute values shown, Figures 6.7 and 6.8 relate topological characteristics to the *percentage changes* of RGAT+PEW to RGAT and RGAT to MLP, so that the relative benefits of the model compared to standard architectures can be understood.

We find that topological characteristics do not fully determine model performance but, nevertheless, it is possible to make a series of observations related to them. Generally, the performance of the method decreases as the size of the graph grows in number of nodes, diameter, and edge density (metrics that are themselves correlated). This result can be explained by the fact that our experimental protocol

relies on a fixed number of demand matrices, which represent a smaller sample of the distribution of demand matrices as the graph increases in size. Hence, this can lead to a model with worse generalisation from the training to the test phase, despite the larger parameter count. This is corroborated by the analysis in Figure 6.7. On the other hand, the performance of the method typically improves with increasing heterogeneity in node and link-level properties (namely, variance in the capacities and degree / weighted betweenness centralities). The relationship between the NMSE and some properties (e.g., weighted betweenness) may be non-linear. Regarding the benefits of a standard RGAT with respect to an MLP, the only clear relationship emerging in Figure 6.8 is that the RGAT performs better in graphs with a high edge density. Such graphs contain more alternative paths that the flows can take to reach the destinations, which can make the routing outcome more tied to graph structure.

6.4.5 Learning Curves

Representative learning curves for the Uninett2011 graph (the largest in terms of edge count) are shown in Figure 6.6. The remaining learning curves are delegated to Appendix D.5. For their generation, we report the MSE on the held-out validation set of the best-performing hyperparameter combination for each architecture and demand input representation. To smoothen the curves, we apply exponential weighting with an $\alpha_{\text{EW}} = 0.92$. We also skip the validation losses for the first 5 epochs since their values are on a significantly larger scale and would distort the plots. As large spikes sometimes arise, validation losses are truncated to be at most the value obtained after the 5 epochs. An interesting trend shown by the learning curves is that the models consistently require more epochs to reach a low validation loss in the ECMP case than for SSP, reflecting its increased complexity.

6.5 Discussion

A key assumption behind the PEW approach is that a set of labels that identify the nodes is known, so that when topologies vary, the mapping to a particular weight parametrisation is kept consistent. This is a suitable assumption for a variety of real-world networks. For example, backbone networks at the scale of an Internet Service Provider are characterised by infrequent upgrades in infrastructure. The structure of large-scale logistics and transportation networks also tends to evolve

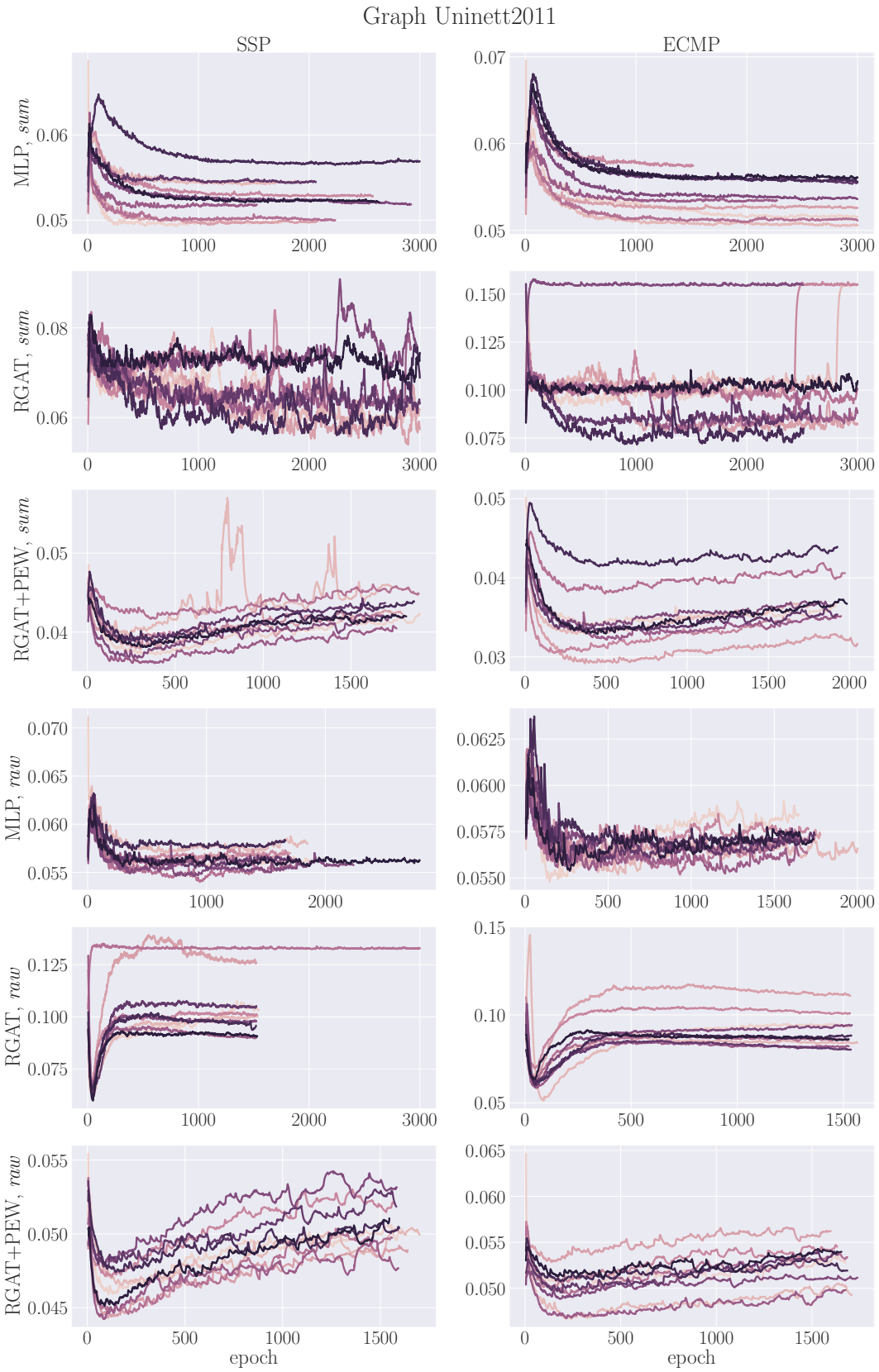


Figure 6.6: Learning curves for Uninett2011.

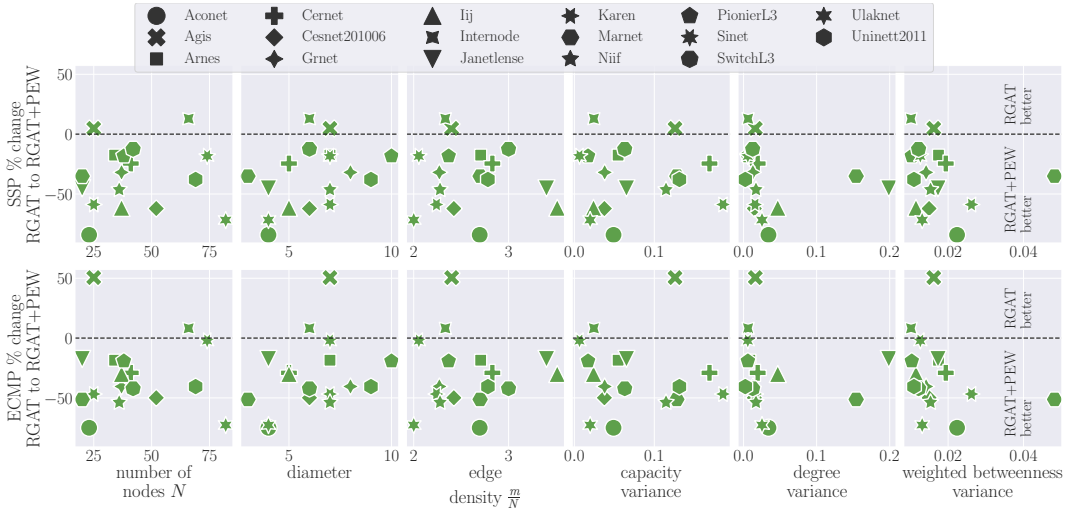


Figure 6.7: Relationship between the percentage changes in NMSE from RGAT to RGAT+PEW and the topological characteristics of the considered graphs.

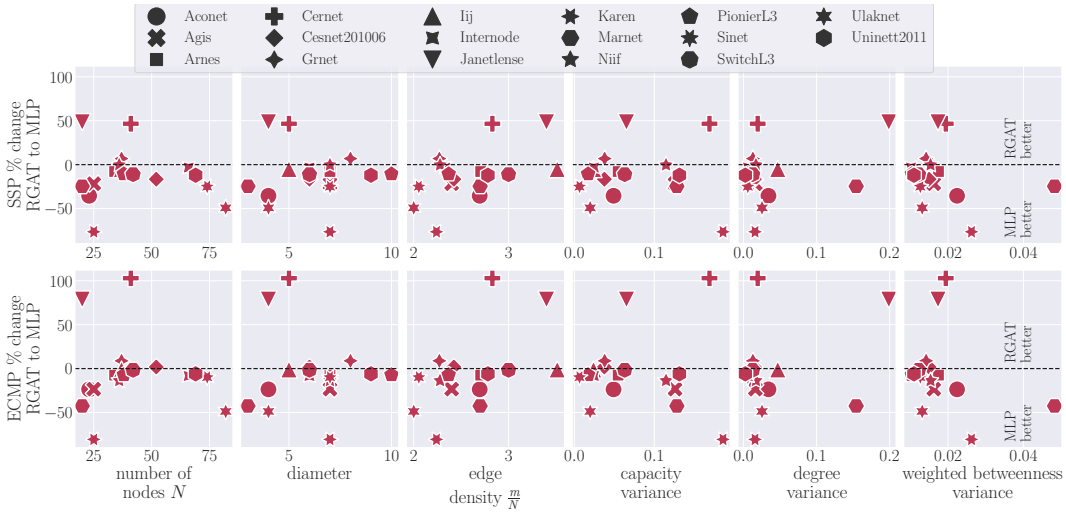


Figure 6.8: Relationship between the percentage changes in NMSE from RGAT to MLP and the topological characteristics of the considered graphs.

relatively slowly and, in any case, the computational time is negligible compared to the timescale and resource cost of these changes. However, performance may degrade in highly dynamic networks, where the timescale of the structural changes is substantially lower than the time needed to recalculate link utilisations and the subsequent adaptations of the systems making use of such a predictive model.

While this chapter has focussed on learning the properties of *existing* routing protocols in order to isolate concerns, in future work we aim to pursue learning *new* routing protocols given the proposed learning representation and broader insights in this problem space that we have obtained.

6.6 Summary

In this chapter, we have addressed the problem of data-driven routing of flows across a graph, which has several applications of practical relevance in areas as diverse as logistics and computer networks. We have proposed Per-Edge Weights, an effective model architecture for predicting link loads in a network based on historical observations, given a traffic demand matrix and a routing strategy. The novelty of our approach resides in the use of weight parametrisations for aggregating messages that are unique for each edge of the graph.

We have demonstrated that the proposed solution brings substantial gains in predictive performance over standard graph learning and MLP approaches, besides having the added benefit of simplicity. Furthermore, in our exploration of other facets of ***RQ6***, we have shown that this architecture is able to exploit the full demand matrix, unlike previous methods for which a lossy aggregation of features is preferable. Our findings also highlight the importance of topology for data-driven routing. We have shown that performance typically decreases when the graph grows in size, but increases with higher levels of heterogeneity of local properties. Additionally, the gain in predictive performance of GNN-based methods over a simple MLP increases when the observed graph structures are heterogeneous due to different subsets of nodes being present in the network.

Chapter 7

Conclusion

In this section, we first summarise the contributions of this thesis. Then, we discuss its limitations, notable directions for future work, as well as possible areas of impact for the proposed techniques. We conclude with high-level closing thoughts.

7.1 Summary and Contributions

In this thesis, we proposed and evaluated learning-based approaches for optimising the outcomes of processes taking place on graphs. The core insight is that viewing such combinatorial optimisation problems as decision-making processes can bring a host of advantages over prior methods, which include: a greater level of flexibility with respect to the objectives to be optimised; the ability to discover algorithms that are more effective than prior methods; and the possibility of achieving fast evaluation times after having undergone training.

These characteristics were consistently demonstrated throughout the previous chapters, in which we discussed computationally challenging problems of practical interest. The thesis contributed *formulations* of these problems as decision-making processes on graphs; *solution methods* for MDPs that extend standard algorithms and are particularly tailored for network-structured systems; and *learning representations* that encode appropriate inductive biases.

We began, in Chapter 3, by considering the problem of optimising graph structural properties through edge additions. We presented a formulation of this problem as a decision-making process, and proposed an approach based on RL and GNNs to

discover heuristics for graph construction, answering ***RQ1*** positively. We demonstrated the ability of the proposed method to optimise the structure of networks in the presence of random node failures and targeted attacks in a way that outperforms prior hand-designed heuristics while being faster to evaluate after training (***RQ2***). Furthermore, we showed that the approach is able to effectively scale, in some cases, to larger graphs than encountered during training (***RQ3***).

Chapter 4 focussed on optimising graph structure with RL (***RQ1***). We demonstrated the scalability and practicality of such approaches for real-world problems (***RQ3***). Firstly, in case a stakeholder is interested in the optimisation of a particular spatial network, rather than a generalisable predictive model, one can use planning approaches to sidestep the cost of model training entirely. Secondly, we proposed an MDP formulation that goes beyond raw topology, encapsulating several realistic traits of networks positioned in physical space, such as restrictions on connection densities and lengths. The proposed algorithm was shown to substantially outperform prior search-based methods for the optimisation of the efficiency and attack resilience of networks, while using a similar computational budget to them (***RQ2***).

In Chapter 5, we treated the problem of finding a Maximal Independent Set of nodes that maximises a given objective function. Formulating this task as an MDP to address ***RQ4***, we showed that one can outperform prior heuristics by using a decision-time planning technique. Furthermore, in our quest to answer ***RQ5***, we proposed a method for performing Imitation Learning of a policy defined on graphs, which gives "the best of both worlds": the ability to maintain much of the solid performance of the original algorithm while being orders of magnitude faster to evaluate. As motivating case study, the approach was applied to finding desirable equilibria of the networked best-shot Public Goods Game, a social dilemma scenario. In this relevant application for ***RQ4***, the approach was able to find equilibrium configurations of high fairness and social welfare, which a social planner might incentivise a society to move towards.

Finally, Chapter 6 focussed on learning representations, an important component of solutions that use ML for combinatorial optimisation. It called into question the use of GNN architectures with a single, identical global message function for treating Multi-Commodity Network Flow problems – answering the first part of ***RQ6***

negatively through experimental evidence. We proposed a GNN model with per-edge parametrised message functions, which is akin to allowing nodes to distribute flow quantities along outgoing edges in a flexible fashion. Our experimental evaluation, which addresses the latter parts of ***RQ6***, showed that the proposed method is able to outperform existing architectures in terms of predictive performance in a Supervised Learning setting; it can utilise the full matrix of demands as input; and it performs comparatively better in graphs with heterogeneous characteristics.

7.2 Limitations and Future Work

We now discuss the high-level limitations of the proposed methods, which are shared with other works that apply ML to combinatorial optimisation problems.

Firstly, an important issue is the fact that one cannot guarantee that the learned models will generalise well when encountering instances outside of the training distribution – this is a fundamental limitation of ML approaches and a direct consequence of the No Free Lunch theorems [362]. In online decision-making scenarios, one may want to have the tools in place to detect distribution shift, and possibly use a fall-back approach whose properties and expected performance are well-understood.

Secondly, ML-based approaches typically require spending an overhead in terms of experimentation and computational resources for the various stages of the pipeline, such as setting up the datasets, performing feature engineering, training the model, and selecting the values of the hyperparameters. However, once such a model is trained, it can typically be used to perform predictions whose computational cost is negligible in comparison. Costs may be mitigated in the future by collective efforts to train generalist "foundation models" that can be shared among researchers working on similar problems, akin to the current dynamics of sharing large language and protein folding models. However, the unique nature of different graph combinatorial optimisation problems may prove challenging in this sense.

Lastly, an important limitation that was alluded to, but not addressed in the present thesis, is the interpretability of the learned models and algorithms. We have seen that the proposed approaches can optimise the given objectives remarkably well, but we are not necessarily able to identify the mechanisms that lead to this observed performance. This is an important part of our future research agenda. Much like physicists would simulate a process of interest then work backwards to

try to derive physical laws, interpretability of an algorithm learned through RL may help us in formulating it a traditional way, potentially allowing for optimisations that can dramatically speed up performance.

Existing interpretability techniques are not directly applicable in this scenario given the graph-structured data and our framing of the problems in the RL setting. Interpretability of both GNNs and RL are areas of active interest in the ML community (e.g., [371, 345]) but there remains significant work to be done, especially at their intersection. Notably, a recent work [141] adapts concept-based explainability methods to GNNs in the Supervised Learning setting, showing that logical rules for several classic graph algorithms, such as Breadth-First Search and Kruskal's method for finding a minimal spanning tree, can be extracted. Akin to work that tackles the explainability of RL in visual domains (e.g., [256] treats Atari games), we consider that there is scope for developing techniques that are tailor-made for explaining policies learned by RL on graphs for solving combinatorial optimisation problems.

Let us move on to discussing some areas for future work. Opportunities for direct extension, which may be seen as "low-hanging fruit", were discussed throughout the thesis chapters. To briefly summarise, the proposed methods can be directly applied to the optimisation of another objective function by simply substituting it with the quantity of interest. As discussed in Chapter 3, such methods can also be applied for optimising a weighted combination of objectives, since in the real world there may be different, competing metrics. To encapsulate real-world constraints, one can also define restrictions on the action space or disallowed states, in a similar fashion to the method presented in Chapter 4.

Even though the thesis has made contributions to improving the scalability of the proposed methods, we believe that this is a fundamental aspect which a future research agenda should be focussed on. Another possible path to improving scalability is to consider actions that, instead of depending on the node labels, execute certain predefined transformations that have a high chance of improving the solution (e.g., swapping two components of the solution based on a greedy criterion). This would aid scalability by decreasing the size of the state and action spaces, with the downside of a possible loss of generality as a result. This type of approach is common in metaheuristics such as the ALNS [291] and is currently being pursued in some

of our further work [191] with applications to the VRP [335]. Another avenue for improving scalability is treating the problem hierarchically, for example by grouping nodes based on geographical regions, community structure, or roles that they play in the system.

This thesis has focussed on optimising the outcome of processes from the point of view of a single, central, planner. This has some appealing characteristics such as providing full observability and control over the construction of the solution. For certain systems, a decentralised, multi-agent formulation in which an agent only has local observability and control may be a viable approach. If the setting is not fully cooperative, potential issues with agents maximising their own gain to the detriment of the common good may arise. Nevertheless, multi-agent modelling can be a viable tool for obtaining complex, emergent behaviour from local decisions, especially when the size of the problem exceeds the scope and capacity of a single central planner.

7.3 Applications and Impact

Given the generality of graph representations, the work herein may find applications in diverse fields, in ways that are difficult to anticipate. Below, we make an attempt to summarise the areas in which we foresee a path to impact.

Overall, the present thesis is highly relevant to the operations research discipline (which significantly overlaps with industrial engineering and management science), since it contributes flexible approaches for decision-making on networks such that resources and budgets are efficiently used. Possible application areas include the optimisation of supply chains, logistics and transportation systems, and network engineering. Given the practical relevance of these fields of study, we expect impact might occur not only in an academic setting, but that technologies based on the proposed methods may have an impact in practice. We envision that such technologies can lead to reduced operational costs and usage of resources (natural or human).

The work on graph construction presented in Chapters 3 and 4 is highly relevant to engineering and the physical sciences. There are possible applications of the proposed techniques in designing structures, both at a macroscopic and microscopic level. With respect to the former, we mention structural engineering for the stability and resilience of human-made buildings, as well as the robustness of transportation

networks. Regarding the latter, similar techniques may be used to discover materials and compounds with desirable properties.

Potentially, the proposed work may also find applications in decision-making and policy design of local authorities and governments. Approaches similar to the work in Chapter 5 can be used to run simulations that assess the right means of targeting interventions in a networked system such as to encourage certain desirable outcomes. As an example scenario, important health factors such as smoking and obesity are known to be linked to individuals' social networks [71, 72], and hence a policy-maker could be interested in the best way of targeting a public health campaign. Furthermore, this type of technique may assist authorities in deciding how to best invest and spend available resources to, for example, extend a transportation network or invest in public infrastructure. Such methods may help in making decisions that have a well-specified, objective, target outcome.

We note that the type of mathematical modelling used in this thesis necessarily makes simplifying assumptions about the real world and cannot capture all of its complexity. Due to this, we highlight the importance of involving stakeholders and domain experts in the modelling process when considering practical applications, so that assumptions, risks, and benefits are thoroughly analysed and specified. By doing so, negative impacts may be foreseen and mitigated.

7.4 Closing Thoughts

More broadly, when taken together, the chapters of this thesis give a blueprint for approaching graph combinatorial optimisation problems in a data-driven way. One needs to specify:

1. The elements that make up the state of the world and are visible to the decision-making agent. Typically, the state will contain both static elements (out of the control of the agent) and dynamic parts (may be modified through the agent's decisions). The constituents of a state can take the form of a subset of nodes or edges, subgraphs, as well as features and attributes that are global or attached to nodes and edges.

2. What are the levers that the agent can use to exert change in the world and modify part of the state. More complex operations can be defined using com-

positionality (e.g., analogously to the decomposition of edge additions in the selection of two nodes used in Chapters 3 and 4), which, in keeping with the spirit of the Markov assumption, is important for achieving scalability.

3. The ways in which the world changes as a result of the actions and/or outside interference. While we have treated deterministic cases in this thesis, one can also consider situations with stochastic properties. These are manageable as-is by model-free RL techniques, while planning methods can be extended to stochastic settings, for example by "averaging out" several outcomes [51].

4. Finally, the quantity that one cares about, and seeks to optimise. This would typically take the form of an objective function for which the set of world states is the domain.

To give an example, in a follow-up work [107], we have extended the method in Chapter 3 to the problem of rewiring a computer network such that the navigation of an attacker is impeded, a common scenario in cybersecurity. Even though the applications and technical details differ substantially, at a high level the work follows this blueprint quite faithfully.

Given the ubiquity of optimisation problems and the wide applicability of the methodology described above, it can be worthwhile to reduce the barrier to entry. Currently, we expect it is high due to needing to master several deeply technical areas. Looking further ahead, it should be possible to allow an end user to specify the system states, actions, and constraints – while letting a software package take care of aspects such as learning mechanics, feature design, and model training. It may even be possible to develop a concise language for expressing these elements, through which the necessary learning environment can be generated automatically. Doing so would advance its level of maturity in the technological cycle.

Can we, therefore, collectively hang up our boots, leaving the machines to discover how to solve these problems? We argue that this not the case. Generic decision-making algorithms and learning representations are clearly not a silver bullet since they do not necessarily exploit problem structure efficiently. Examples of this were shown in various parts of the thesis, such as the clear gain obtained by using a cost-sensitive simulation policy in the SG-UCT algorithm proposed in

Chapter 4, or the harm in using the wrong inductive bias for a learning representation in Chapter 6. There are substantial improvements to be made by encoding knowledge and understanding about the problem into these solution approaches. We envisage the future of algorithm development as leveraging both deep problem insights and highly efficient machine-learned components.

Appendix A

Appendix for Chapter 3: *Goal-directed Graph Construction using Reinforcement Learning*

A.1 Implementation

Source code is available at `https://github.com/VictorDarvariu/graph-construction-rl`. A version of the source code without the GPU dependency is available at `https://github.com/VictorDarvariu/graph-construction-rl-lite`. For full details about how to set up the experimental infrastructure, run the experiments, and reproduce the results, please see the instructions in the repository. The RNet–DQN implementation uses PyTorch and is bootstrapped from the RL-S2V implementation provided by Dai et al.[1], which is based on the authors' implementation of the S2V GNN[2]. We implement the performance-critical robustness simulations in a custom C++ module. For completeness, details of how the robustness objective functions are calculated are shown in Algorithm 1.

A.2 Data Availability

The original real-world datasets used in this research (Scigrid, Euroroad) are publicly available and were retrieved via the Scigrid [247] project website (`https://www.power.scigrid.de/pages/downloads.html`) and KONECT [218] (`http://konect.cc/`) respectively. They can be downloaded without registration. Scigrid is licensed under the Open Database License (ODbL) v1.0, while Euroroad is license-free. The scripts

[1] `https://github.com/Hanjun-Dai/graph_adversarial_attack`
[2] `https://github.com/Hanjun-Dai/pytorch_structure2vec`

Algorithm 3 Pseudocode of algorithm for estimating the robustness of graphs to targeted and random removals of nodes.

Input: undirected graph G
Parameters: node removal strategy (random or targeted),
number of Monte Carlo simulations b_{MC}
Output: estimated robustness $\mathcal{F}$

```
fracs = Array(b_MC)
for simulation i = 1 to b_MC do                          ▷ Trivially parallelisable
    ξ ← GeneratePermutation(G)                           ▷ Depends on node removal strategy
    for permutation index j = 1 to N do
        v ← ξ[j]                                         ▷ Select next node in permutation
        RemoveNode(G, v)                                 ▷ Remove the node and all its edges
        ncc ← NumConnectedComponents(G)
        if ncc > 1 then
            fracs[i] ← j/N                               ▷ Critical Fraction reached
            break
return mean(fracs)
```

and instructions used to extract the subgraphs corresponding to individual countries in the infrastructure networks are available in the code repository.

A.3 Parameters

General Parameters. We train for $4 \times 10^4 \eta$ steps using the Adam optimiser and a batch size of 50. For RNet–DQN, we use an experience replay buffer of size equal to the number of steps. We let $\gamma = 1$, since we are in the finite horizon case. Target network weights are updated every 50 steps. For SL, we perform early stopping if the validation loss does not improve after 10^4 steps. We do not use any weight regularisation or normalisation. We do not perform any gradient clipping when computing the DQN or MSE losses. Weights are initialised using Glorot initialisation. We use a learning rate $\alpha = 0.0001$. During training, we scale the rewards linearly by a factor of 100 in order to improve numerical stability.

Parameters for Synthetic Graphs. For synthetic graphs, we use a number of message passing rounds $L = 3$. The MLP layer has 128 hidden units. For RNet–DQN, we decay the exploration rate ϵ linearly from $\epsilon = 1$ to $\epsilon = 0.1$ for the first half of training steps, then fix $\epsilon = 0.1$ for the rest of the training. To estimate the values of the objective functions we use $2|V|$ Monte Carlo simulations.

Parameters for Real-World Graphs. We use $L = 5$ as these graphs are larger in size and diameter. The MLP layer has 32 hidden units. For RNet–DQN, we decay ϵ

linearly from $\epsilon = 1$ to $\epsilon = 0.1$ in the first 10% of training steps, then fix $\epsilon = 0.1$ for the rest of the training. To estimate the objective functions, we use 40 MC simulations.

A.4 Runtime Details

For the computational experiments presented in this chapter, we used a machine with 2 Intel Xeon E5-2637 v4 processors, 64GB RAM, and a NVIDIA Tesla P100 GPU. For synthetic graphs, the time per RNet–DQN training run for a specific robustness objective function and graph family is approximately 3.5 hours with $\eta = 5$. The computational time for the synthetic graph experiments is approximately 1350 hours (56 days), while the experiments on real-world graphs took approximately 1240 hours (51 days). Since most of the cost is due to considering a large number of random initialisations in order to provide statistically robust evaluations, the experiments can be trivially parallelised.

Appendix B

Appendix for Chapter 4: *Planning Spatial Networks with Monte Carlo Tree Search*

B.1 Implementation

We implement all approaches and baselines in Python using a variety of numerical and scientific computing packages [181, 158, 246, 351], while the calculations of the objective functions (efficiency and robustness) are performed in a custom C++ module as they are the main speed bottleneck. The implementation is provided as Docker containers together with instructions that enable reproducing (up to hardware differences) all the results reported in the chapter, including tables and figures. The implementation is available at `https://github.com/VictorDarvariu/planning-spatial-networks-mcts`.

B.2 Data Availability

The *Internet* dataset is publicly available without any restrictions and can be downloaded via the Internet Topology Zoo website, `http://www.topology-zoo.org/dataset.html`. The *Metro* dataset was originally used in [296], and was licensed to us by the authors for the purposes of this chapter. A copy of the *Metro* dataset can be obtained by others by contacting its original authors for licensing (see `https://www.quanturb.com/data`).

B.3 Parameters

Hyperparameter optimisation for UCT and SG-UCT is performed separately for each objective function and synthetic graph model / real-world network dataset. For synthetic graphs, hyperparameters are tuned over a disjoint set of graphs. For

real-world graphs, hyperparameters are optimised separately for each graph. We consider an exploration constant $\epsilon_{\text{UCT}} \in \{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$. Since the ranges of the rewards may vary in different settings, we further employ two means of standardisation: during the tree search we instead use $\mathcal{F}(G_T)$ as the final reward R_T, and further standardise ϵ_{UCT} by multiplying with the average reward observed at the root in the previous timestep – ensuring consistent levels of exploration. The hyperparameters for the ablation study are bootstrapped from those of standard UCT, while $\beta \in \{0.1, 0.25, 0.5, 1, 2.5, 5, 10\}$ for the SG-UCT$_{\text{MINCOST}}$ variant is optimised separately. These results are used to reduce the hyperparameter search space for SG-UCT for both synthetic and real-world graphs. Values of hyperparameters used are shown in Table B.1. For estimating $\mathcal{F}_R$ we use $|V|/4$ Monte Carlo simulations.

B.4 Runtime Details

Experiments were carried out on an internal cluster of 8 machines, each equipped with 2 Intel Xeon E5-2630 v3 (2014) processors and 128GB RAM. On this infrastructure, all experiments reported in this chapter took approximately 21 days to complete.

Table B.1: Hyperparameters used for UCT and SG-UCT.

Experiment	Graph	Objective Agent	ϵ_{UCT} $\mathcal{F}_E$	$\mathcal{F}_R$	ϕ $\mathcal{F}_E$	$\mathcal{F}_R$	β $\mathcal{F}_E$	$\mathcal{F}_R$
Internet	Colt	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.1	0.1	—	—	—	—
	GtsCe	SG-UCT	0.1	0.05	AECS-40	AECS-40	25	25
		UCT	0.25	0.1	—	—	—	—
	TataNld	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.1	0.1	—	—	—	—
	UsCarrier	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.05	0.1	—	—	—	—
Metro	Barcelona	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.05	0.05	—	—	—	—
	Beijing	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.05	0.05	—	—	—	—
	Mexico	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.25	0.05	—	—	—	—
	Moscow	SG-UCT	0.25	0.1	AECS-40	AECS-40	25	25
		UCT	0.05	0.05	—	—	—	—
	Osaka	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.05	0.25	—	—	—	—
KH-25	—	SG-UCT	0.05	0.25	AECS-40	AECS-40	25	25
		UCT	0.1	0.1	—	—	—	—
		$\text{SG-UCT}_{\text{MINCOST}}$	0.1	0.1	—	—	25	25
KH-50	—	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.05	0.25	—	—	—	—
		$\text{SG-UCT}_{\text{MINCOST}}$	0.05	0.25	—	—	10	25
KH-75	—	SG-UCT	0.05	0.05	AECS-40	AECS-40	25	25
		UCT	0.05	0.1	—	—	—	—
		$\text{SG-UCT}_{\text{MINCOST}}$	0.05	0.1	—	—	25	25

Appendix C

Appendix for Chapter 5: *Solving Graph-based Public Goods Games with Monte Carlo Tree Search and Imitation Learning*

C.1 Implementation

Our implementation is available at `https://github.com/VictorDarvariu/solving-graph-pgg` as Docker containers together with instructions that enable reproducing (up to hardware differences) all the results reported in the chapter, including tables and figures. For complete instructions to reproduce the results, please consult the `README.md` file in the repository. We implement all approaches and baselines in Python using a variety of numerical and scientific computing packages [181, 158, 246, 270, 351]. For GIL, we use the PyTorch implementation of structure2vec provided by the original authors [85].

C.2 Data Availability

The provided implementation contains the necessary code and instructions to generate the synthetic data on which the experiments are carried out. Please consult the `README.md` file in the root of the repository.

C.3 Parameters

Hyperparameters. We optimise hyperparameters for UCT, GIL, and the SA methods; the other methods are hyperparameter-free. For UCT, we use a random simulation

policy and a number of node expansions per move $b_{\text{sims}} = 20N$ (larger values provide diminishing returns). At each step, once simulations are completed, we select the child node with the largest number of visits as the action (RobustChild). We treat the exploration parameter ϵ_{UCT} as a hyperparameter to be optimised, and for each problem instance we consider $\epsilon_{\text{UCT}} \in \{0.05, 0.1, 0.25, 0.5, 1, 2.5\}$. Since the ranges of the rewards may vary in different settings, we further standardise ϵ_{UCT} by multiplying with the average reward observed at the root in the previous timestep – ensuring consistent levels of exploration. For GIL, the learning rate $\alpha \in \{10^{-2}, 10^{-3}, 10^{-4}\}$, the number of S2V message passing rounds $L \in \{3, 4, 5, 6\}$, and the training strategy (separate, mixed, or curriculum) are optimised using a grid search. For SA, we consider $\tau_{\text{SA}} \in \{10^1, 10^2, 10^3, 10^4\}$, stop the optimisation after 10^4 steps without an improvement, and use a cut-off of 10^7 steps. For SA, the lowest tested value $\tau_{\text{SA}} = 10$ of the simulated annealing temperature was optimal across all settings tested (we did not explore lower values since the method would become significantly more expensive to run, and is already slow as shown in Figure 5.6).

GIL Training. GIL is the only method that requires training. The datasets on which this method is trained correspond to demonstrations of the hyperparameter-optimised UCT. We carry out the Imitation Learning procedure as described in Section 5.2.4 and evaluate performance on the validation instances every 50 steps. We train using the Adam [207] optimiser for $2 \cdot 10^3$ steps and use a batch size of 5 in all cases (larger batch sizes proved harmful). The dimension of the proto-action vector ψ and the number of S2V latent variables are both 64. The temperature τ_{GIL} is initialised to 10.

C.4 Runtime Details

Experiments were carried out on an internal cluster of 8 machines, each equipped with 2 Intel Xeon E5-2630 v3 processors and 128GB RAM. On this infrastructure, the experiments reported in this chapter took approximately 14 days to complete.

Table C.1: Mean rewards obtained by the methods split by cost setting, graph model, objective function, and number of players.

$\mathbf{c}$	$\mathcal{G}$	$\mathcal{F}$	N	Rand	TH	TLC	BR	PT	SA	UCT	GIL (ours)
HC	BA	$\mathcal{F}_{GF}$	15	$0.757_{\pm 0.005}$	0.814	0.785	$0.753_{\pm 0.006}$	$0.797_{\pm 0.016}$	$0.825_{\pm 0.001}$	$\mathbf{0.848}_{\pm 0.000}$	$0.845_{\pm 0.001}$
			25	$0.750_{\pm 0.005}$	0.806	0.774	$0.747_{\pm 0.007}$	$0.794_{\pm 0.017}$	$0.820_{\pm 0.002}$	$\mathbf{0.842}_{\pm 0.000}$	$0.839_{\pm 0.000}$
			50	$0.742_{\pm 0.007}$	0.797	0.773	$0.741_{\pm 0.005}$	$0.788_{\pm 0.016}$	$0.811_{\pm 0.001}$	$\mathbf{0.835}_{\pm 0.000}$	$0.832_{\pm 0.001}$
			75	$0.741_{\pm 0.006}$	0.797	0.772	$0.736_{\pm 0.004}$	$0.789_{\pm 0.014}$	$0.810_{\pm 0.001}$	$\mathbf{0.831}_{\pm 0.000}$	$0.828_{\pm 0.002}$
			100	$0.737_{\pm 0.005}$	0.796	0.767	$0.732_{\pm 0.004}$	$0.787_{\pm 0.012}$	$0.808_{\pm 0.001}$	$\mathbf{0.829}_{\pm 0.000}$	$0.826_{\pm 0.001}$
		$\mathcal{F}_{SW}$	15	$0.708_{\pm 0.007}$	0.793	0.738	$0.702_{\pm 0.009}$	$0.763_{\pm 0.020}$	$0.806_{\pm 0.001}$	$\mathbf{0.827}_{\pm 0.000}$	$0.825_{\pm 0.000}$
			25	$0.702_{\pm 0.008}$	0.782	0.728	$0.698_{\pm 0.011}$	$0.762_{\pm 0.023}$	$0.801_{\pm 0.002}$	$\mathbf{0.820}_{\pm 0.000}$	$0.819_{\pm 0.000}$
			50	$0.694_{\pm 0.010}$	0.774	0.726	$0.691_{\pm 0.007}$	$0.756_{\pm 0.021}$	$0.791_{\pm 0.001}$	$\mathbf{0.813}_{\pm 0.000}$	$0.811_{\pm 0.000}$
			75	$0.692_{\pm 0.009}$	0.774	0.726	$0.684_{\pm 0.007}$	$0.759_{\pm 0.018}$	$0.790_{\pm 0.001}$	$\mathbf{0.809}_{\pm 0.000}$	$0.806_{\pm 0.001}$
			100	$0.688_{\pm 0.008}$	0.774	0.720	$0.680_{\pm 0.007}$	$0.757_{\pm 0.016}$	$0.788_{\pm 0.001}$	$\mathbf{0.807}_{\pm 0.001}$	$0.804_{\pm 0.001}$
	ER	$\mathcal{F}_{GF}$	15	$0.806_{\pm 0.004}$	0.839	0.861	$0.807_{\pm 0.002}$	$0.839_{\pm 0.007}$	$0.849_{\pm 0.002}$	$\mathbf{0.895}_{\pm 0.000}$	$0.892_{\pm 0.001}$
			25	$0.841_{\pm 0.002}$	0.865	0.889	$0.840_{\pm 0.001}$	$0.881_{\pm 0.004}$	$0.879_{\pm 0.002}$	$\mathbf{0.925}_{\pm 0.000}$	$0.920_{\pm 0.001}$
			50	$0.893_{\pm 0.001}$	0.909	0.934	$0.892_{\pm 0.001}$	$0.930_{\pm 0.001}$	$0.924_{\pm 0.001}$	$\mathbf{0.958}_{\pm 0.000}$	$0.954_{\pm 0.000}$
			75	$0.916_{\pm 0.001}$	0.928	0.953	$0.915_{\pm 0.001}$	$0.946_{\pm 0.001}$	$0.940_{\pm 0.000}$	$\mathbf{0.970}_{\pm 0.000}$	$0.965_{\pm 0.006}$
			100	$0.930_{\pm 0.001}$	0.940	0.962	$0.930_{\pm 0.001}$	$0.957_{\pm 0.001}$	$0.951_{\pm 0.001}$	$\mathbf{0.977}_{\pm 0.000}$	$0.969_{\pm 0.011}$
		$\mathcal{F}_{SW}$	15	$0.782_{\pm 0.004}$	0.823	0.841	$0.782_{\pm 0.001}$	$0.820_{\pm 0.008}$	$0.836_{\pm 0.002}$	$\mathbf{0.882}_{\pm 0.000}$	$0.878_{\pm 0.001}$
			25	$0.829_{\pm 0.002}$	0.856	0.877	$0.827_{\pm 0.001}$	$0.871_{\pm 0.005}$	$0.872_{\pm 0.002}$	$\mathbf{0.918}_{\pm 0.000}$	$0.912_{\pm 0.000}$
			50	$0.887_{\pm 0.001}$	0.905	0.931	$0.887_{\pm 0.001}$	$0.927_{\pm 0.002}$	$0.921_{\pm 0.001}$	$\mathbf{0.956}_{\pm 0.000}$	$0.948_{\pm 0.002}$
			75	$0.913_{\pm 0.001}$	0.925	0.951	$0.912_{\pm 0.001}$	$0.944_{\pm 0.001}$	$0.938_{\pm 0.000}$	$\mathbf{0.969}_{\pm 0.000}$	$0.965_{\pm 0.001}$
			100	$0.928_{\pm 0.001}$	0.939	0.960	$0.928_{\pm 0.001}$	$0.955_{\pm 0.001}$	$0.950_{\pm 0.001}$	$\mathbf{0.976}_{\pm 0.000}$	$0.971_{\pm 0.002}$
	WS	$\mathcal{F}_{GF}$	15	$0.818_{\pm 0.007}$	0.817	0.868	$0.814_{\pm 0.005}$	$0.833_{\pm 0.007}$	$0.848_{\pm 0.006}$	$\mathbf{0.904}_{\pm 0.002}$	$0.903_{\pm 0.000}$
			25	$0.799_{\pm 0.006}$	0.807	0.867	$0.801_{\pm 0.004}$	$0.817_{\pm 0.006}$	$0.828_{\pm 0.003}$	$\mathbf{0.891}_{\pm 0.000}$	$0.890_{\pm 0.000}$
			50	$0.801_{\pm 0.002}$	0.805	0.864	$0.801_{\pm 0.003}$	$0.819_{\pm 0.005}$	$0.830_{\pm 0.001}$	$\mathbf{0.890}_{\pm 0.000}$	$\mathbf{0.890}_{\pm 0.000}$
			75	$0.799_{\pm 0.002}$	0.800	0.862	$0.801_{\pm 0.003}$	$0.818_{\pm 0.003}$	$0.827_{\pm 0.001}$	$\mathbf{0.886}_{\pm 0.000}$	$\mathbf{0.886}_{\pm 0.000}$
			100	$0.800_{\pm 0.001}$	0.802	0.863	$0.801_{\pm 0.002}$	$0.818_{\pm 0.004}$	$0.828_{\pm 0.001}$	$0.887_{\pm 0.000}$	$\mathbf{0.888}_{\pm 0.000}$
		$\mathcal{F}_{SW}$	15	$0.795_{\pm 0.009}$	0.797	0.850	$0.790_{\pm 0.006}$	$0.811_{\pm 0.008}$	$0.832_{\pm 0.008}$	$\mathbf{0.889}_{\pm 0.002}$	$0.888_{\pm 0.000}$
			25	$0.775_{\pm 0.008}$	0.784	0.847	$0.779_{\pm 0.005}$	$0.797_{\pm 0.007}$	$0.812_{\pm 0.003}$	$\mathbf{0.875}_{\pm 0.000}$	$0.873_{\pm 0.001}$
			50	$0.778_{\pm 0.003}$	0.784	0.845	$0.780_{\pm 0.005}$	$0.799_{\pm 0.005}$	$0.815_{\pm 0.001}$	$\mathbf{0.876}_{\pm 0.000}$	$0.875_{\pm 0.000}$
			75	$0.777_{\pm 0.002}$	0.779	0.843	$0.780_{\pm 0.003}$	$0.798_{\pm 0.003}$	$0.812_{\pm 0.001}$	$\mathbf{0.871}_{\pm 0.000}$	$\mathbf{0.871}_{\pm 0.000}$
			100	$0.778_{\pm 0.001}$	0.781	0.844	$0.780_{\pm 0.003}$	$0.798_{\pm 0.004}$	$0.813_{\pm 0.001}$	$0.871_{\pm 0.000}$	$\mathbf{0.872}_{\pm 0.000}$
IC	BA	$\mathcal{F}_{GF}$	15	$0.837_{\pm 0.001}$	0.851	—	$0.837_{\pm 0.001}$	$0.846_{\pm 0.005}$	$\mathbf{0.855}_{\pm 0.000}$	$\mathbf{0.855}_{\pm 0.000}$	$\mathbf{0.855}_{\pm 0.000}$
			25	$0.835_{\pm 0.001}$	0.845	—	$0.835_{\pm 0.000}$	$0.844_{\pm 0.006}$	$0.851_{\pm 0.000}$	$\mathbf{0.852}_{\pm 0.000}$	$0.850_{\pm 0.000}$
			50	$0.832_{\pm 0.000}$	0.842	—	$0.832_{\pm 0.000}$	$0.840_{\pm 0.006}$	$\mathbf{0.848}_{\pm 0.000}$	$0.846_{\pm 0.000}$	$0.845_{\pm 0.001}$
			75	$0.832_{\pm 0.001}$	0.841	—	$0.832_{\pm 0.001}$	$0.839_{\pm 0.005}$	$\mathbf{0.846}_{\pm 0.000}$	$0.843_{\pm 0.000}$	$0.843_{\pm 0.001}$
			100	$0.831_{\pm 0.001}$	0.840	—	$0.832_{\pm 0.001}$	$0.838_{\pm 0.004}$	$\mathbf{0.844}_{\pm 0.000}$	$0.840_{\pm 0.000}$	$0.842_{\pm 0.001}$
		$\mathcal{F}_{SW}$	15	$0.710_{\pm 0.008}$	0.794	—	$0.703_{\pm 0.010}$	$0.760_{\pm 0.021}$	$0.805_{\pm 0.001}$	$\mathbf{0.807}_{\pm 0.000}$	$\mathbf{0.807}_{\pm 0.000}$
			25	$0.702_{\pm 0.009}$	0.779	—	$0.698_{\pm 0.010}$	$0.759_{\pm 0.023}$	$0.799_{\pm 0.001}$	$\mathbf{0.800}_{\pm 0.000}$	$\mathbf{0.800}_{\pm 0.000}$
			50	$0.695_{\pm 0.008}$	0.776	—	$0.692_{\pm 0.007}$	$0.756_{\pm 0.020}$	$0.792_{\pm 0.001}$	$\mathbf{0.793}_{\pm 0.000}$	$\mathbf{0.793}_{\pm 0.001}$
			75	$0.689_{\pm 0.009}$	0.774	—	$0.682_{\pm 0.006}$	$0.756_{\pm 0.017}$	$0.789_{\pm 0.001}$	$\mathbf{0.790}_{\pm 0.000}$	$\mathbf{0.790}_{\pm 0.001}$
			100	$0.687_{\pm 0.008}$	0.772	—	$0.679_{\pm 0.006}$	$0.755_{\pm 0.016}$	$0.785_{\pm 0.001}$	$\mathbf{0.786}_{\pm 0.000}$	$\mathbf{0.786}_{\pm 0.001}$
	ER	$\mathcal{F}_{GF}$	15	$0.844_{\pm 0.001}$	0.860	—	$0.844_{\pm 0.001}$	$0.855_{\pm 0.003}$	$0.868_{\pm 0.001}$	$\mathbf{0.873}_{\pm 0.000}$	$0.872_{\pm 0.000}$
			25	$0.865_{\pm 0.001}$	0.880	—	$0.864_{\pm 0.001}$	$0.881_{\pm 0.001}$	$0.892_{\pm 0.001}$	$\mathbf{0.899}_{\pm 0.000}$	$0.898_{\pm 0.001}$
			50	$0.901_{\pm 0.001}$	0.916	—	$0.900_{\pm 0.001}$	$0.918_{\pm 0.000}$	$0.927_{\pm 0.000}$	$\mathbf{0.932}_{\pm 0.000}$	$0.931_{\pm 0.000}$
			75	$0.921_{\pm 0.000}$	0.931	—	$0.921_{\pm 0.000}$	$0.935_{\pm 0.000}$	$0.943_{\pm 0.000}$	$\mathbf{0.948}_{\pm 0.000}$	$0.943_{\pm 0.006}$
			100	$0.934_{\pm 0.001}$	0.943	—	$0.934_{\pm 0.001}$	$0.945_{\pm 0.000}$	$0.953_{\pm 0.000}$	$\mathbf{0.957}_{\pm 0.000}$	$0.951_{\pm 0.006}$
		$\mathcal{F}_{SW}$	15	$0.780_{\pm 0.002}$	0.818	—	$0.777_{\pm 0.002}$	$0.807_{\pm 0.007}$	$0.834_{\pm 0.001}$	$\mathbf{0.843}_{\pm 0.000}$	$0.841_{\pm 0.000}$
			25	$0.829_{\pm 0.002}$	0.855	—	$0.827_{\pm 0.001}$	$0.856_{\pm 0.002}$	$0.873_{\pm 0.001}$	$\mathbf{0.884}_{\pm 0.000}$	$0.882_{\pm 0.001}$
			50	$0.886_{\pm 0.001}$	0.906	—	$0.885_{\pm 0.001}$	$0.909_{\pm 0.001}$	$0.920_{\pm 0.000}$	$\mathbf{0.926}_{\pm 0.000}$	$0.922_{\pm 0.002}$
			75	$0.912_{\pm 0.001}$	0.925	—	$0.912_{\pm 0.001}$	$0.930_{\pm 0.000}$	$0.939_{\pm 0.000}$	$\mathbf{0.944}_{\pm 0.000}$	$0.942_{\pm 0.002}$
			100	$0.928_{\pm 0.001}$	0.939	—	$0.928_{\pm 0.001}$	$0.942_{\pm 0.000}$	$0.950_{\pm 0.000}$	$\mathbf{0.954}_{\pm 0.000}$	$0.951_{\pm 0.004}$
	WS	$\mathcal{F}_{GF}$	15	$0.843_{\pm 0.003}$	0.844	—	$0.841_{\pm 0.003}$	$0.846_{\pm 0.003}$	$0.856_{\pm 0.002}$	$0.864_{\pm 0.001}$	$\mathbf{0.865}_{\pm 0.001}$
			25	$0.841_{\pm 0.002}$	0.843	—	$0.843_{\pm 0.002}$	$0.848_{\pm 0.002}$	$0.856_{\pm 0.001}$	$0.861_{\pm 0.000}$	$\mathbf{0.863}_{\pm 0.000}$
			50	$0.841_{\pm 0.001}$	0.843	—	$0.842_{\pm 0.002}$	$0.846_{\pm 0.002}$	$0.856_{\pm 0.000}$	$0.861_{\pm 0.000}$	$\mathbf{0.864}_{\pm 0.000}$
			75	$0.841_{\pm 0.000}$	0.842	—	$0.842_{\pm 0.001}$	$0.847_{\pm 0.001}$	$0.856_{\pm 0.000}$	$0.861_{\pm 0.000}$	$\mathbf{0.864}_{\pm 0.000}$
			100	$0.842_{\pm 0.001}$	0.843	—	$0.842_{\pm 0.001}$	$0.846_{\pm 0.001}$	$0.856_{\pm 0.000}$	$0.860_{\pm 0.000}$	$\mathbf{0.864}_{\pm 0.000}$
		$\mathcal{F}_{SW}$	15	$0.779_{\pm 0.007}$	0.783	—	$0.774_{\pm 0.007}$	$0.790_{\pm 0.007}$	$0.812_{\pm 0.004}$	$\mathbf{0.829}_{\pm 0.001}$	$0.828_{\pm 0.002}$
			25	$0.775_{\pm 0.006}$	0.781	—	$0.782_{\pm 0.006}$	$0.794_{\pm 0.006}$	$0.812_{\pm 0.002}$	$0.823_{\pm 0.000}$	$\mathbf{0.826}_{\pm 0.000}$
			50	$0.777_{\pm 0.003}$	0.782	—	$0.778_{\pm 0.005}$	$0.791_{\pm 0.005}$	$0.813_{\pm 0.001}$	$0.824_{\pm 0.000}$	$\mathbf{0.829}_{\pm 0.001}$
			75	$0.777_{\pm 0.001}$	0.780	—	$0.780_{\pm 0.003}$	$0.792_{\pm 0.003}$	$0.812_{\pm 0.001}$	$0.822_{\pm 0.000}$	$\mathbf{0.828}_{\pm 0.001}$
			100	$0.778_{\pm 0.002}$	0.781	—	$0.779_{\pm 0.003}$	$0.791_{\pm 0.003}$	$0.812_{\pm 0.000}$	$0.821_{\pm 0.000}$	$\mathbf{0.828}_{\pm 0.000}$

Table C.2: Win Rates (%) for the different methods.

c	$\mathcal{G}$	$\mathcal{F}$	N	Rand	TH	TLC	BR	PT	SA	UCT	GIL (ours)
HC	BA	$\mathcal{F}_{GF}$	15	1.800	1.000	0.400	0.900	17.100	9.300	**43.100**	26.400
			25	0.400	0.300	0.400	0.100	20.200	5.000	**47.900**	**25.700**
			50	0.000	0.000	0.000	0.000	12.900	2.300	**62.800**	22.000
			75	0.000	0.000	0.100	0.000	13.400	0.800	**66.400**	19.300
			100	0.000	0.000	0.000	0.000	11.200	0.700	**73.800**	14.300
		$\mathcal{F}_{SW}$	15	1.800	0.700	0.400	0.900	18.400	12.000	**39.500**	26.300
			25	0.300	0.100	0.400	0.300	20.600	8.900	**41.500**	27.900
			50	0.100	0.000	0.000	0.000	14.900	3.600	**60.800**	20.600
			75	0.100	0.100	0.100	0.000	15.400	2.200	**62.600**	19.500
			100	0.000	0.000	0.000	0.000	13.000	2.000	**68.400**	16.600
	ER	$\mathcal{F}_{GF}$	15	0.900	0.100	1.100	0.900	7.700	5.200	**54.600**	29.500
			25	0.300	0.000	0.200	0.100	5.000	2.400	**64.800**	**27.200**
			50	0.000	0.000	0.600	0.000	3.100	0.900	**65.500**	29.900
			75	0.000	0.000	0.600	0.000	1.700	0.200	**64.400**	33.100
			100	0.000	0.000	0.300	0.000	1.200	0.400	**67.800**	30.300
		$\mathcal{F}_{SW}$	15	0.800	0.200	1.100	0.600	7.300	5.700	**53.800**	30.500
			25	0.200	0.100	0.500	0.100	4.900	2.400	**60.600**	31.200
			50	0.000	0.000	0.800	0.000	2.700	1.500	**74.800**	20.200
			75	0.000	0.000	0.500	0.000	1.700	0.100	**69.000**	**28.700**
			100	0.000	0.000	0.200	0.000	1.400	0.000	**73.500**	24.900
	WS	$\mathcal{F}_{GF}$	15	0.300	0.000	0.800	0.200	0.700	3.100	**58.200**	36.700
			25	0.000	0.000	0.600	0.000	0.500	0.500	**57.200**	41.200
			50	0.000	0.000	0.100	0.000	0.000	0.000	**55.700**	44.200
			75	0.000	0.000	0.000	0.000	0.000	0.000	**50.700**	49.300
			100	0.000	0.000	0.000	0.000	0.000	0.000	43.700	**56.300**
		$\mathcal{F}_{SW}$	15	0.600	0.000	0.700	0.300	0.900	3.100	**55.800**	38.600
			25	0.000	0.000	0.500	0.000	0.400	1.200	**61.600**	36.300
			50	0.000	0.000	0.000	0.000	0.000	0.000	**60.600**	39.400
			75	0.000	0.000	0.000	0.000	0.000	0.000	49.800	**50.200**
			100	0.000	0.000	0.000	0.000	0.000	0.000	47.200	**52.800**
IC	BA	$\mathcal{F}_{GF}$	15	4.500	1.400	—	3.100	18.200	24.700	**25.100**	23.000
			25	2.100	0.400	—	2.200	22.800	22.900	**31.700**	17.900
			50	0.400	1.200	—	0.600	21.400	**34.000**	25.400	17.000
			75	0.200	0.800	—	0.300	20.300	**40.200**	20.500	17.700
			100	0.000	0.100	—	0.100	21.700	**48.200**	14.900	15.000
		$\mathcal{F}_{SW}$	15	1.900	1.800	—	1.600	19.200	24.200	25.600	**25.700**
			25	0.600	0.900	—	0.400	22.800	24.600	**27.200**	23.500
			50	0.100	0.600	—	0.000	21.200	24.300	**27.600**	26.200
			75	0.000	0.300	—	0.000	20.200	25.800	**25.100**	**28.600**
			100	0.000	0.000	—	0.000	21.500	24.500	25.300	**28.700**
	ER	$\mathcal{F}_{GF}$	15	2.300	1.000	—	2.400	9.600	20.300	**35.300**	29.100
			25	0.600	0.300	—	0.000	6.800	19.200	**39.600**	33.500
			50	0.000	0.100	—	0.000	1.900	17.000	**43.600**	37.400
			75	0.000	0.100	—	0.000	1.500	15.000	**57.200**	26.200
			100	0.000	0.100	—	0.000	0.500	14.000	**55.700**	29.700
		$\mathcal{F}_{SW}$	15	2.300	0.800	—	1.700	10.000	19.600	**35.100**	30.500
			25	0.800	0.500	—	0.300	7.500	18.700	**42.100**	30.100
			50	0.200	0.400	—	0.100	3.800	18.500	**49.000**	28.000
			75	0.000	0.200	—	0.000	1.200	12.600	**49.600**	36.400
			100	0.000	0.100	—	0.000	0.700	13.100	**54.400**	**31.700**
	WS	$\mathcal{F}_{GF}$	15	1.200	0.000	—	0.200	1.500	12.800	37.500	**46.800**
			25	0.200	0.000	—	1.200	2.500	14.800	36.200	**45.100**
			50	0.000	0.000	—	0.000	0.100	3.700	**27.200**	**69.000**
			75	0.000	0.000	—	0.000	0.000	1.300	18.400	**80.300**
			100	0.000	0.000	—	0.000	0.000	0.400	9.700	**89.900**
		$\mathcal{F}_{SW}$	15	0.700	0.100	—	1.000	2.300	15.400	**41.400**	39.100
			25	0.000	0.100	—	1.100	2.000	13.200	34.900	**48.700**
			50	0.000	0.000	—	0.000	0.100	2.800	**27.200**	**69.900**
			75	0.000	0.000	—	0.000	0.000	1.600	19.000	**79.400**
			100	0.000	0.000	—	0.000	0.000	0.600	12.000	**87.400**

Appendix D

Appendix for Chapter 6: *Graph Neural Modelling of Network Flows*

D.1 Implementation

In the future, the implementation will be made publicly available as Docker containers together with instructions that enable reproducing (up to hardware differences) all the results reported in the chapter, including tables and figures. We implement all approaches and baselines in Python using a variety of numerical and scientific computing packages [181, 158, 246, 270, 351].

D.2 Data Availability

The network topology data used in this chapter is part of the Repetita suite [138] and it is publicly available at `https://github.com/svissicchio/Repetita`. We also use the synthetic traffic generator from [154], available at `https://github.com/ngvozdiev/tm-gen`.

D.3 Parameters

All methods are given an equal grid search budget of 12 hyperparameter configurations consisting of the two choices of demand input representations mentioned above, three choices of learning rate $\alpha \in \{10^{-2}, 5 \times 10^{-3}, 10^{-3}\}$ and two choices of model complexity as follows: the MLP is initialised with a first hidden layer size in $\{64, 256\}$ for the *sum* representation and $\{64, 128\}$ for the *raw* representation, with subsequent layers having half the units. For RGAT, the size of the hidden features is $\{8, 32\}$, while for RGAT+PEW it is $\{4, 16\}$ to account for the larger parameter count. For the GNN-based methods, sum pooling is used to compute a graph-level embed-

ding from the node-level features. Despite potential over-smoothing issues of GNNs in graph classification (e.g., as described in [65]), for the flow routing problem, we set the number of layers equal to the diameter so that all traffic entering the network can also exit, including traffic between pairs of points that are the furthest away in the graph.

D.4 Runtime Details

Experiments were carried out on a cluster of 8 machines, each equipped with 2 Intel Xeon E5-2630 v3 processors and 128GB RAM. On this infrastructure, all the experiments reported in this chapter took approximately 28 days to complete. The training and evaluation of models were performed exclusively on CPUs.

D.5 Learning Curves

We next show learning curves for all graphs except Uninett2011, for which the curves were presented in Figure 6.6. They are generated as discussed in Section 6.4.5.

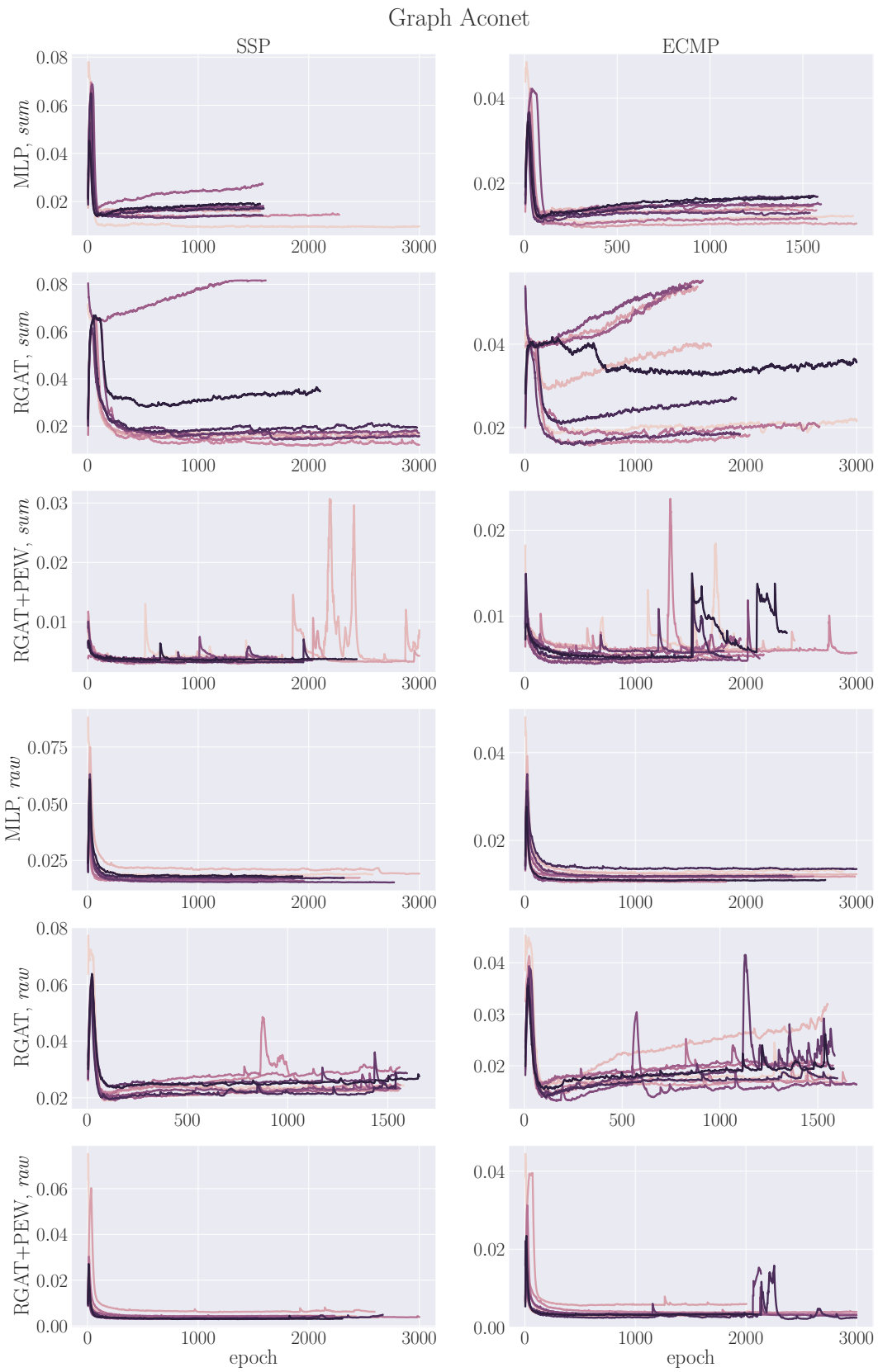


Figure D.1: Learning curves for Aconet.

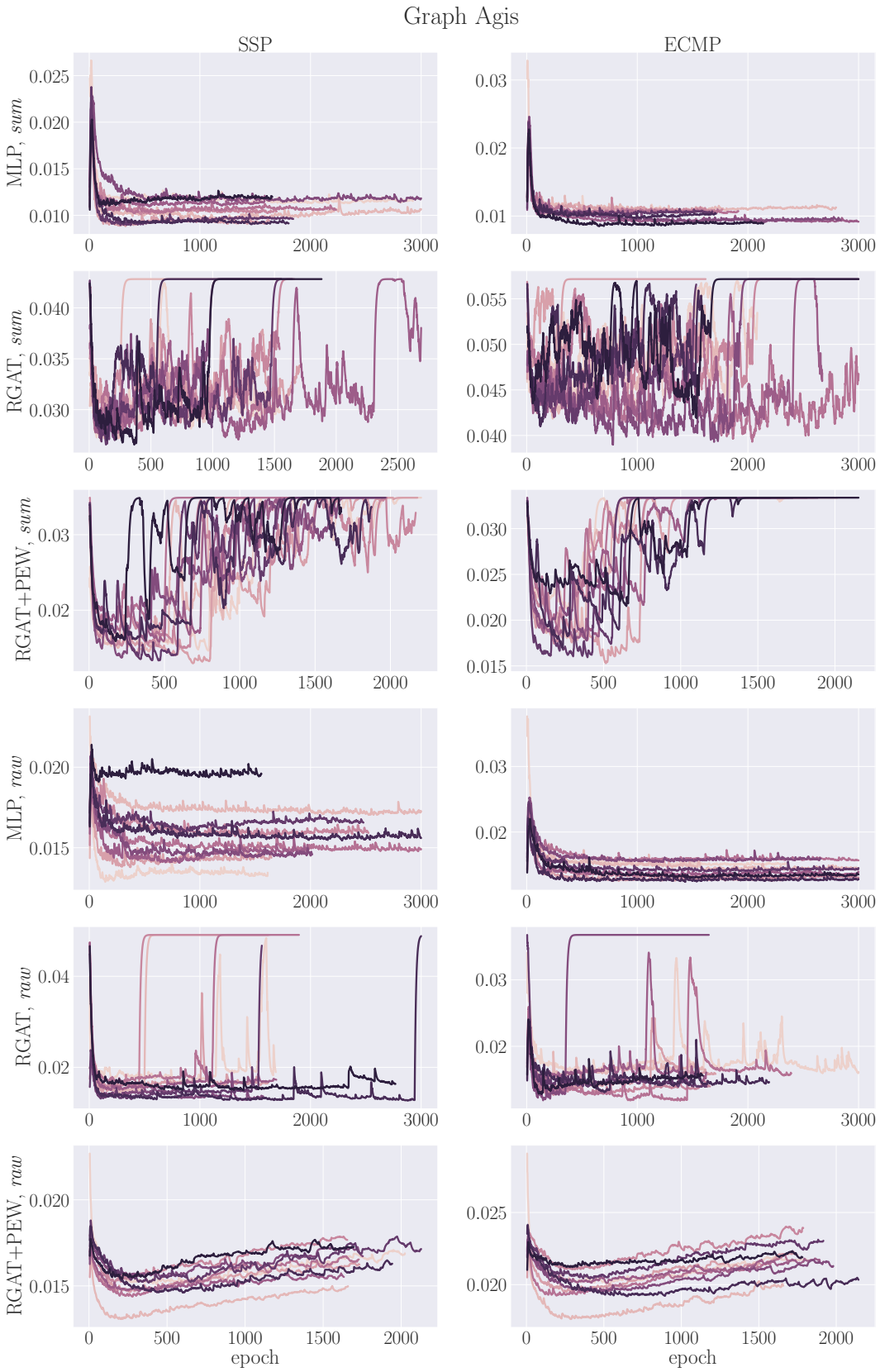


Figure D.2: Learning curves for Agis.

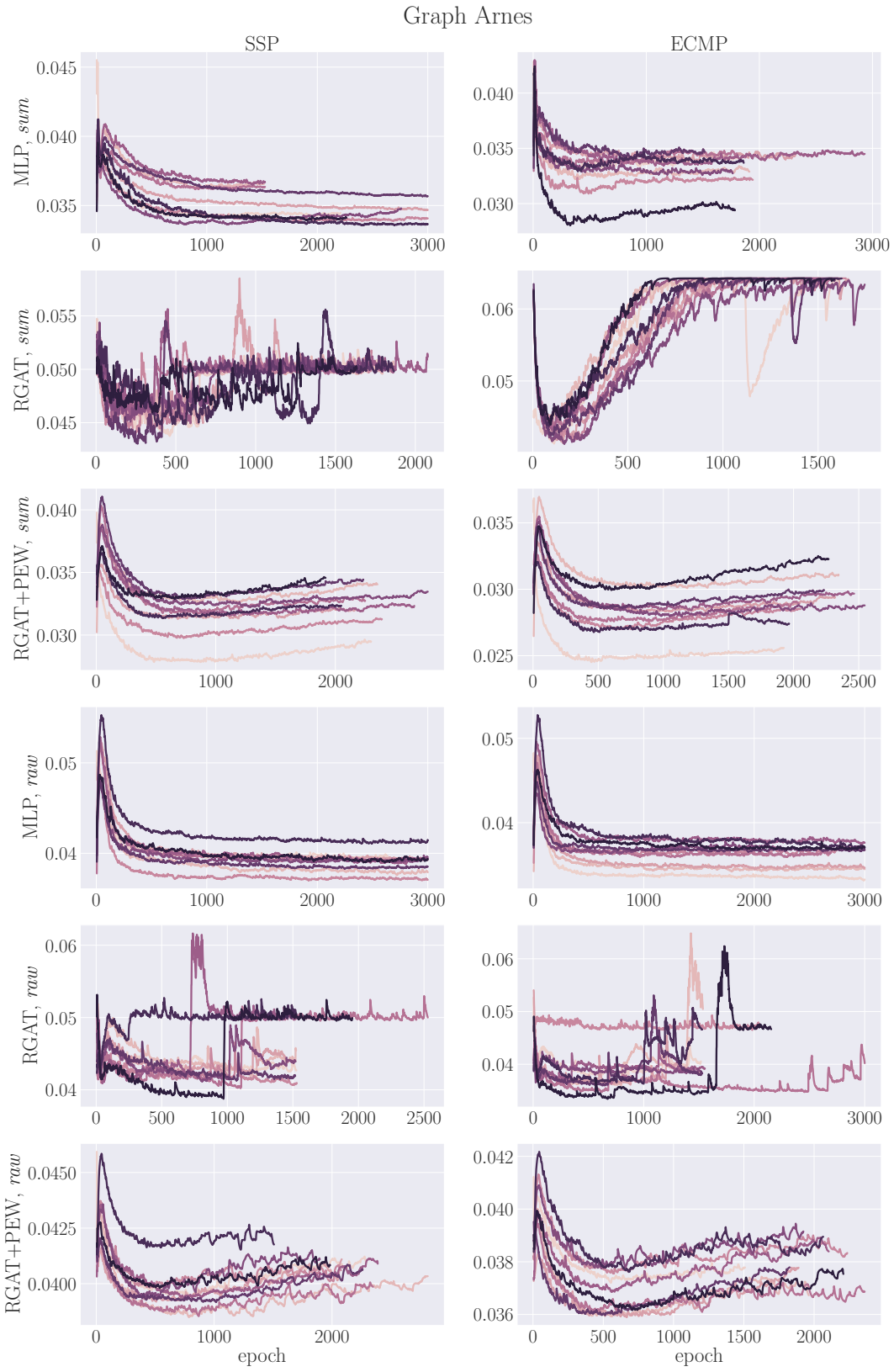


Figure D.3: Learning curves for Arnes.

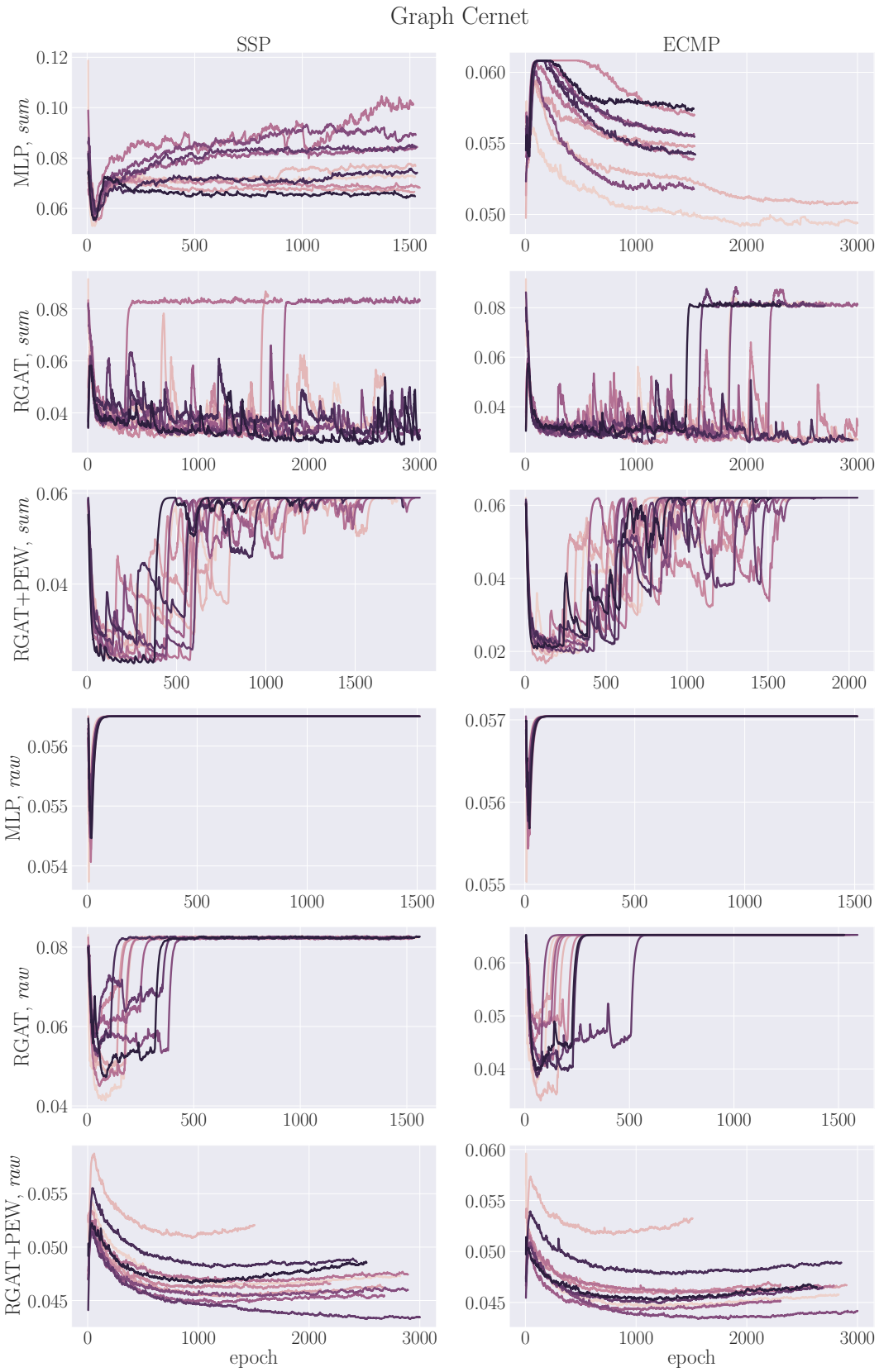


Figure D.4: Learning curves for Cernet.

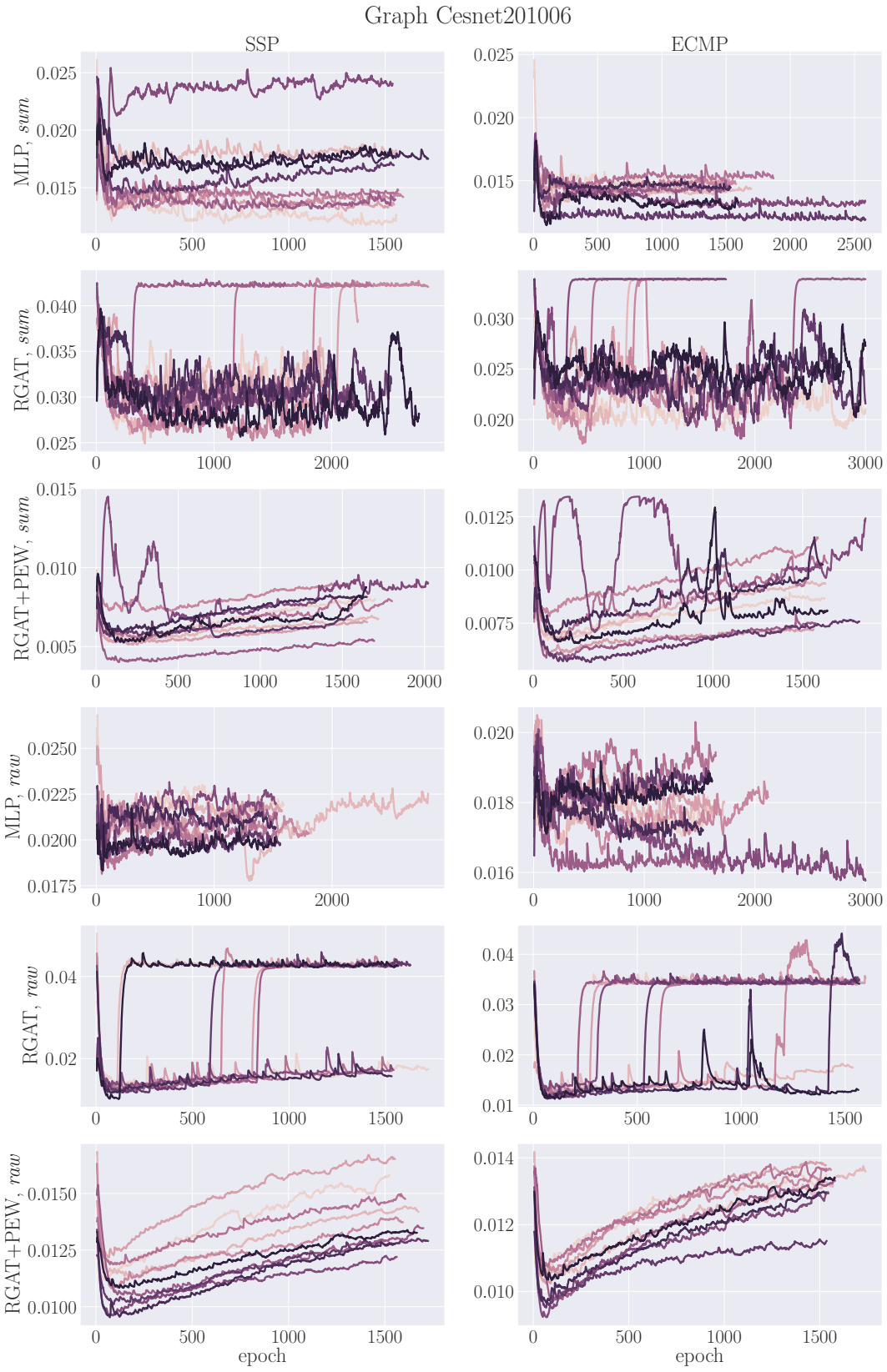


Figure D.5: Learning curves for Cesnet201006.

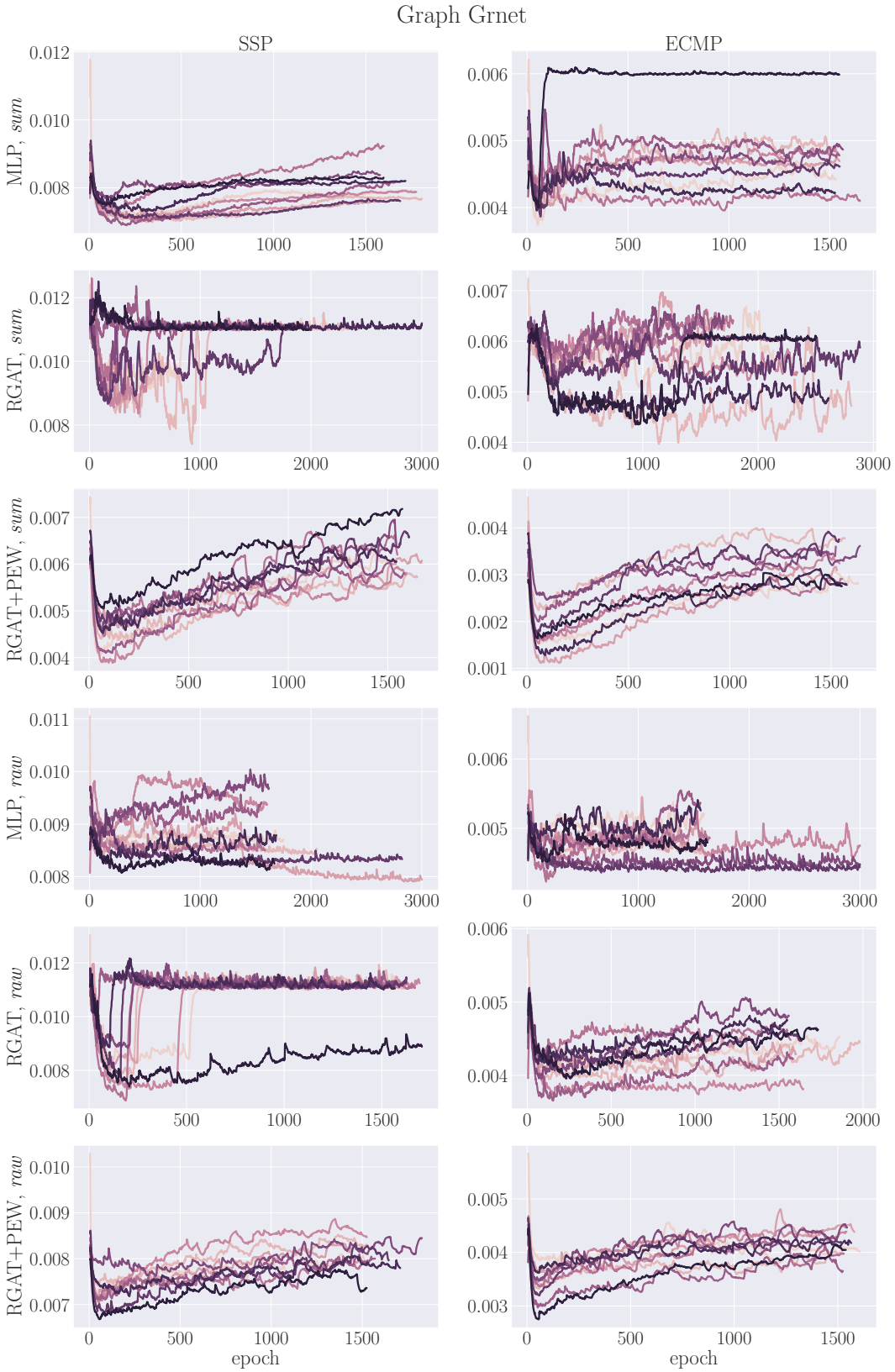


Figure D.6: Learning curves for Grnet.

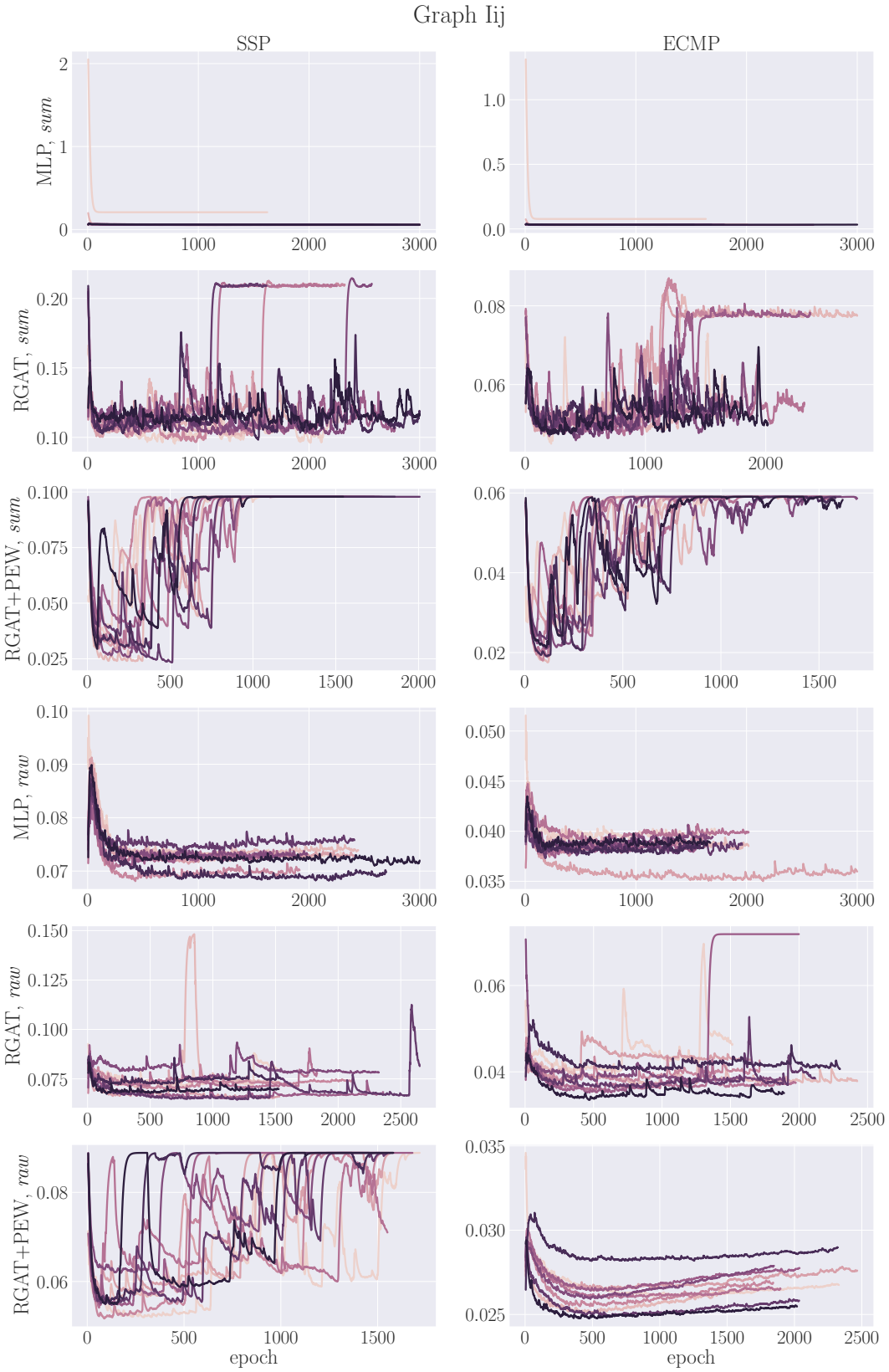


Figure D.7: Learning curves for Iij.

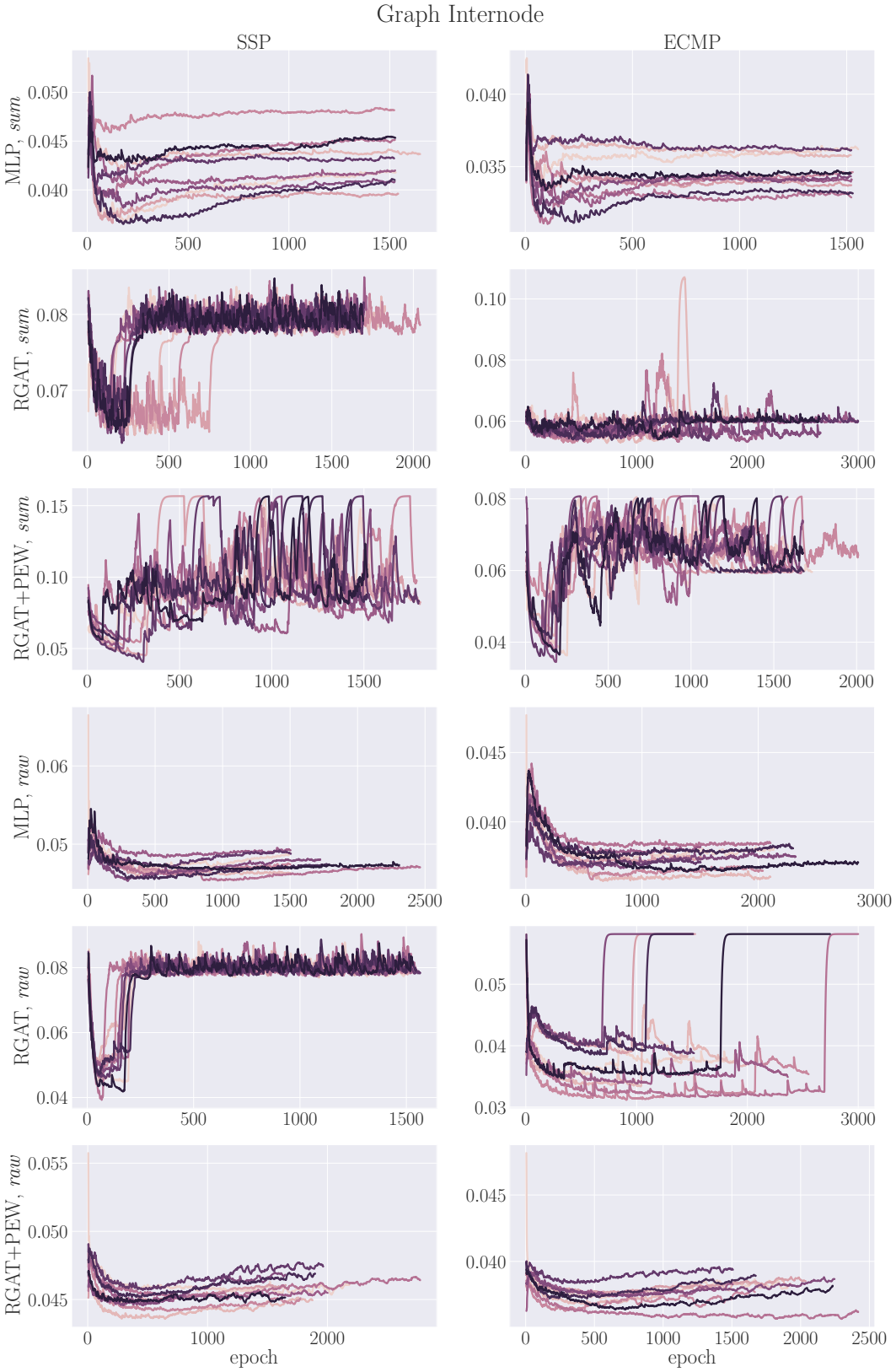


Figure D.8: Learning curves for Internode.

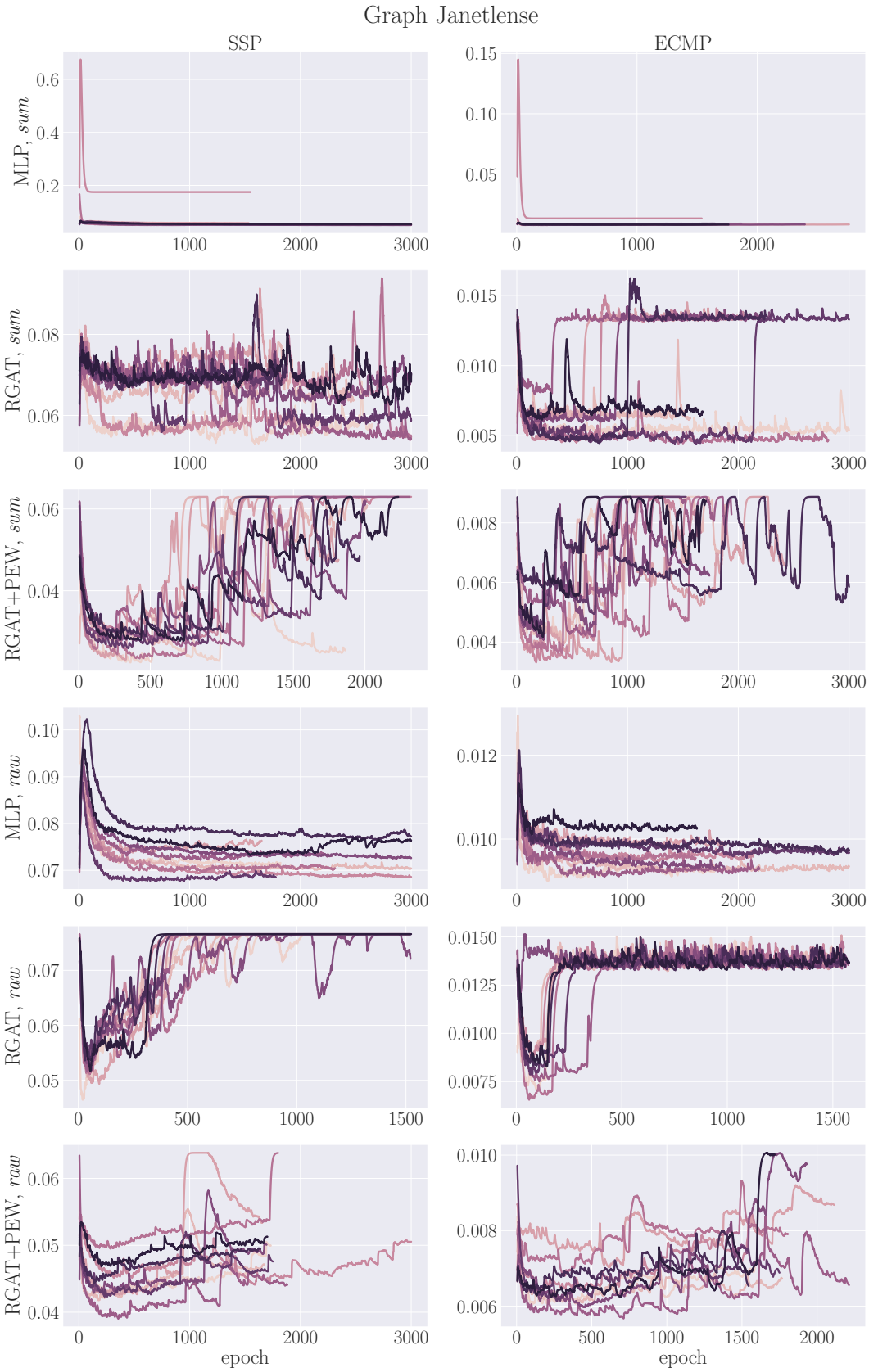


Figure D.9: Learning curves for Janetlense.

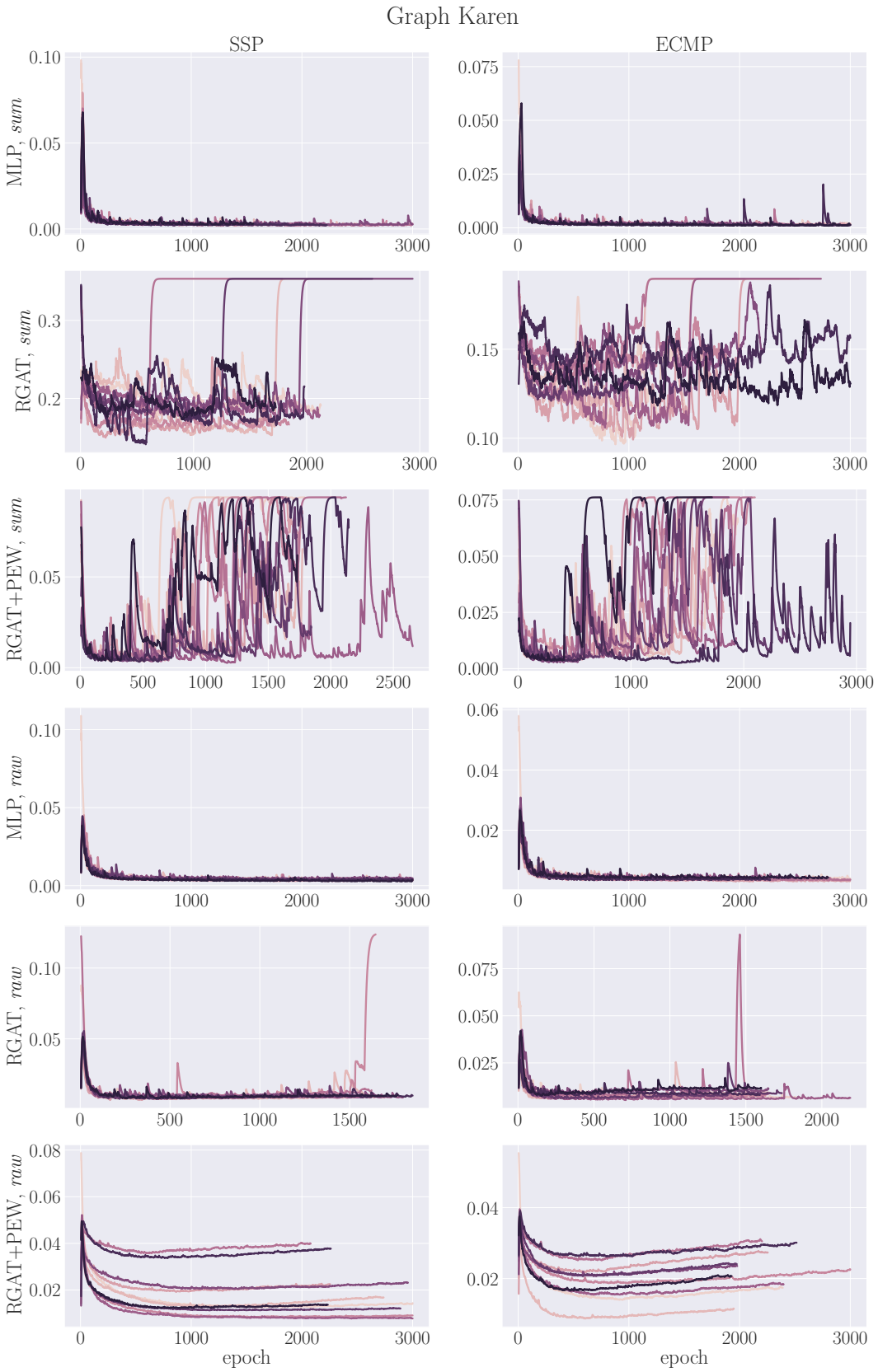


Figure D.10: Learning curves for Karen.

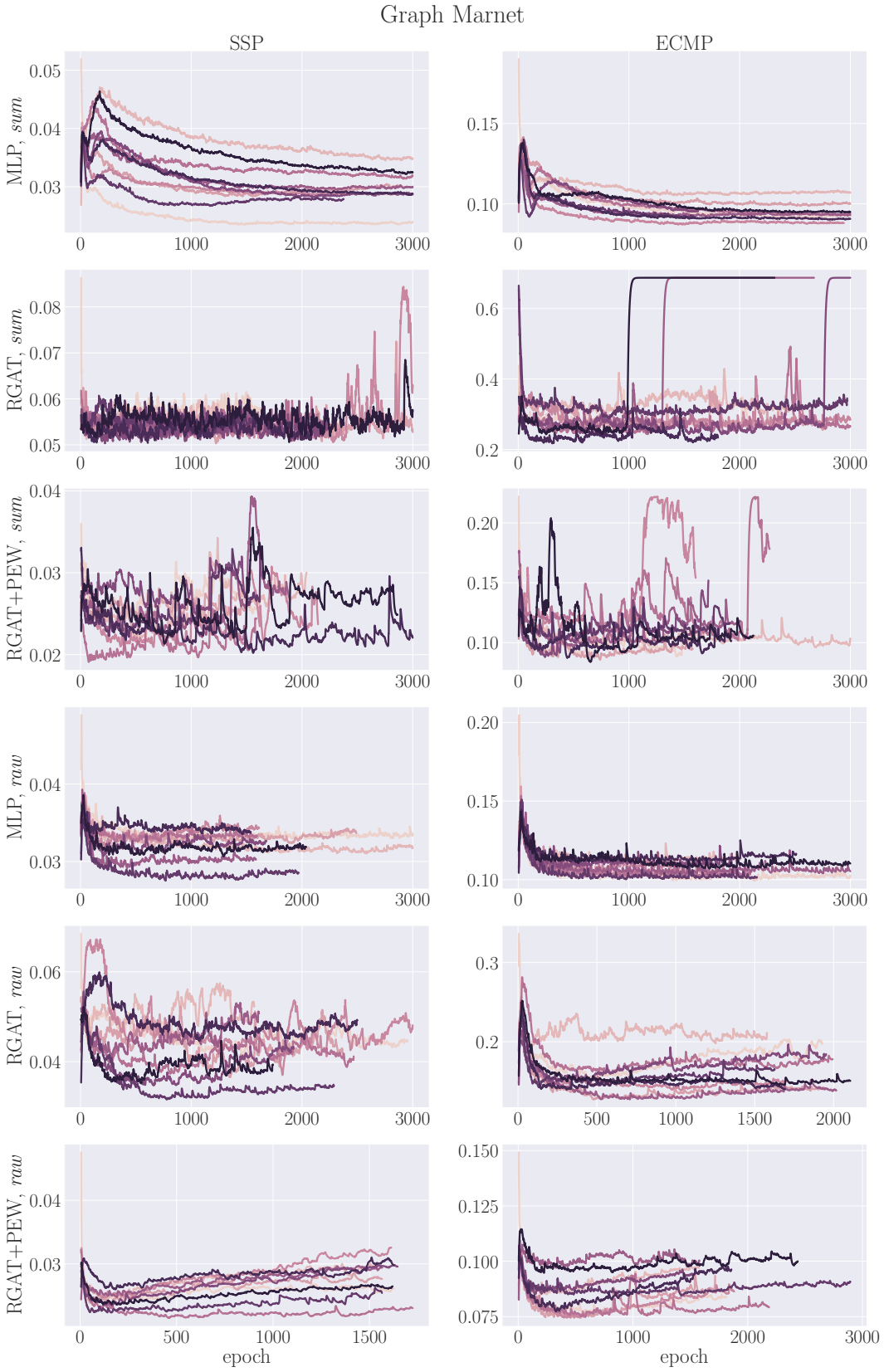


Figure D.11: Learning curves for Marnet.

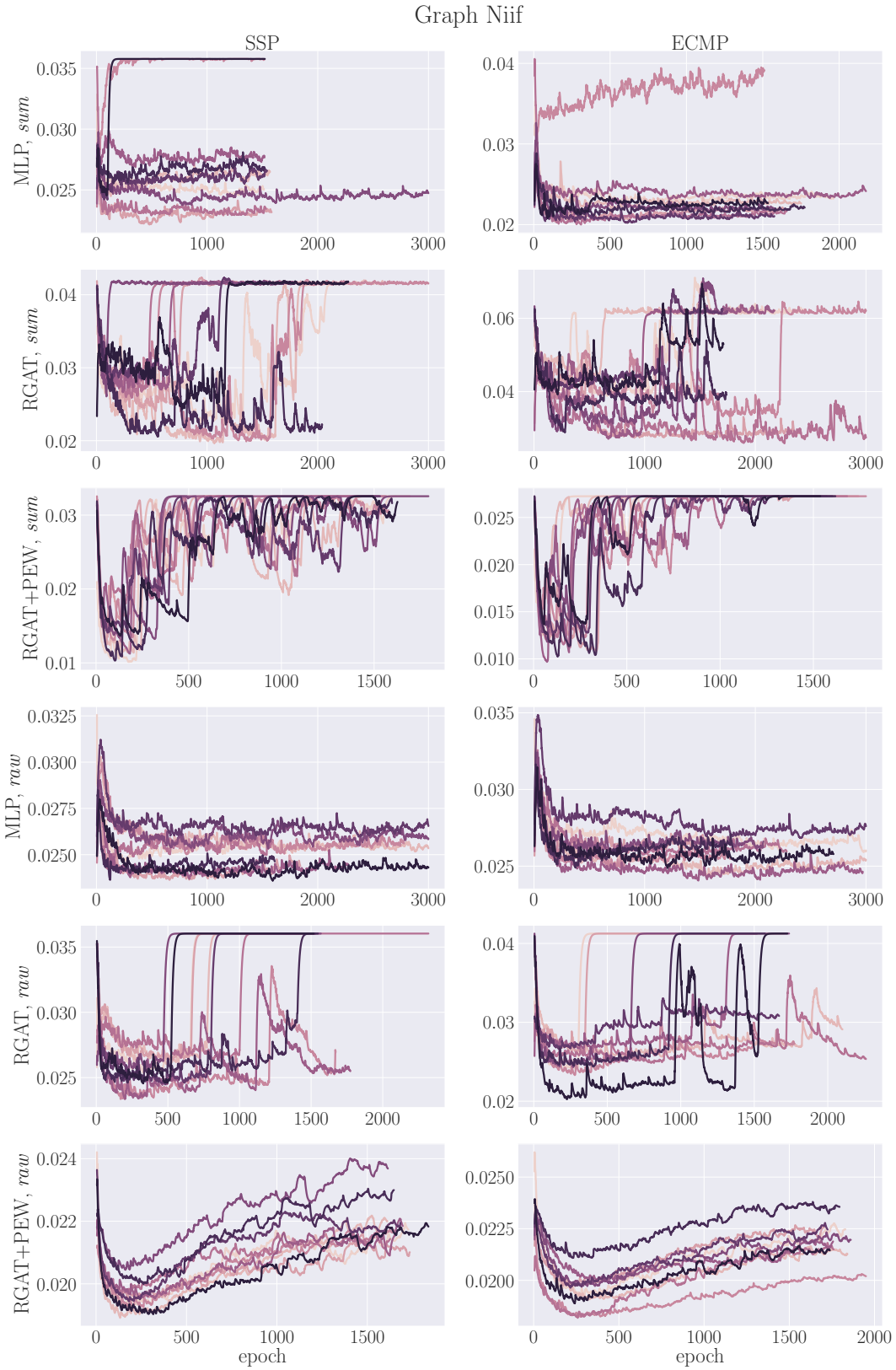


Figure D.12: Learning curves for Niif.

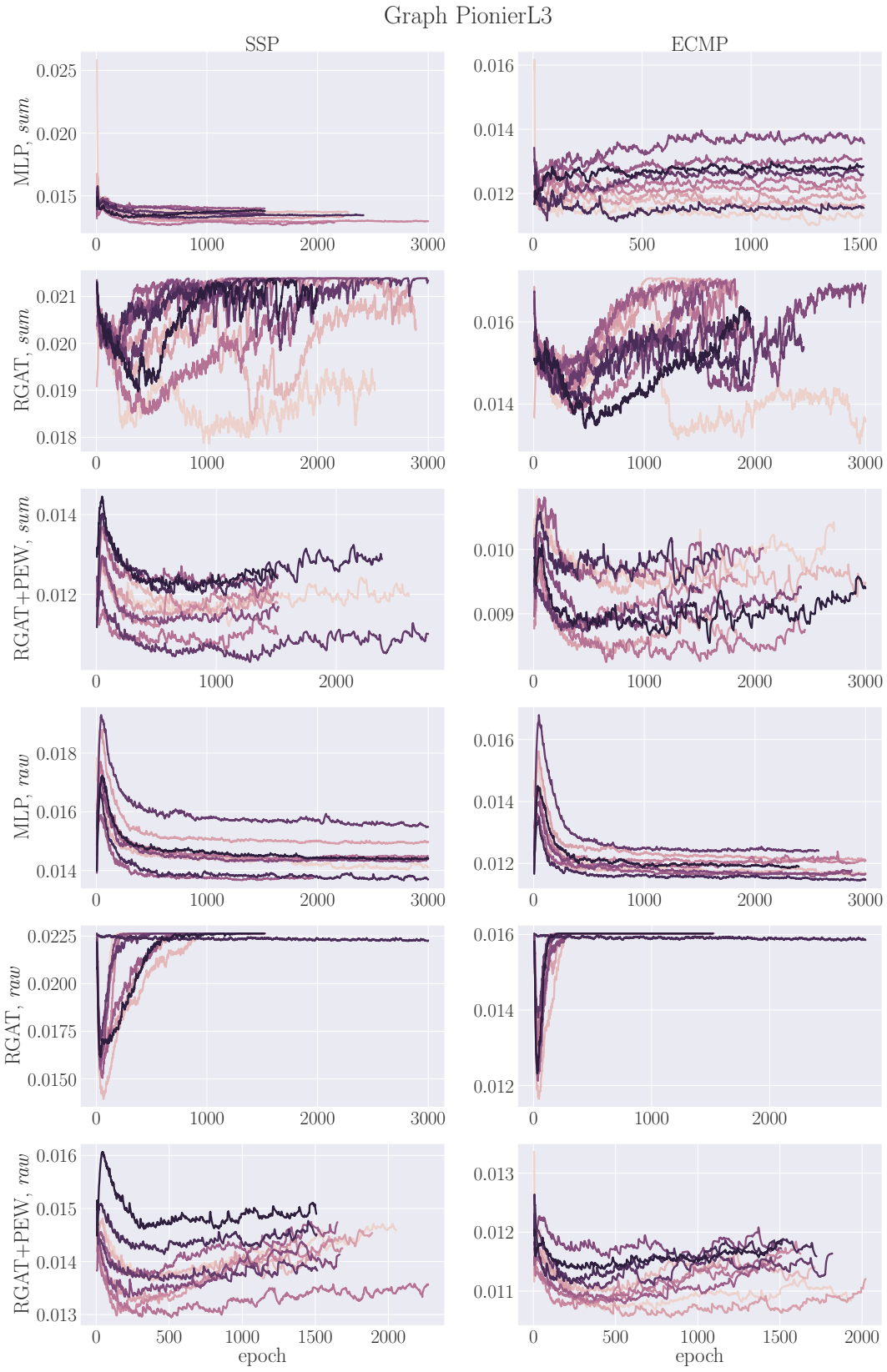


Figure D.13: Learning curves for PionierL3.

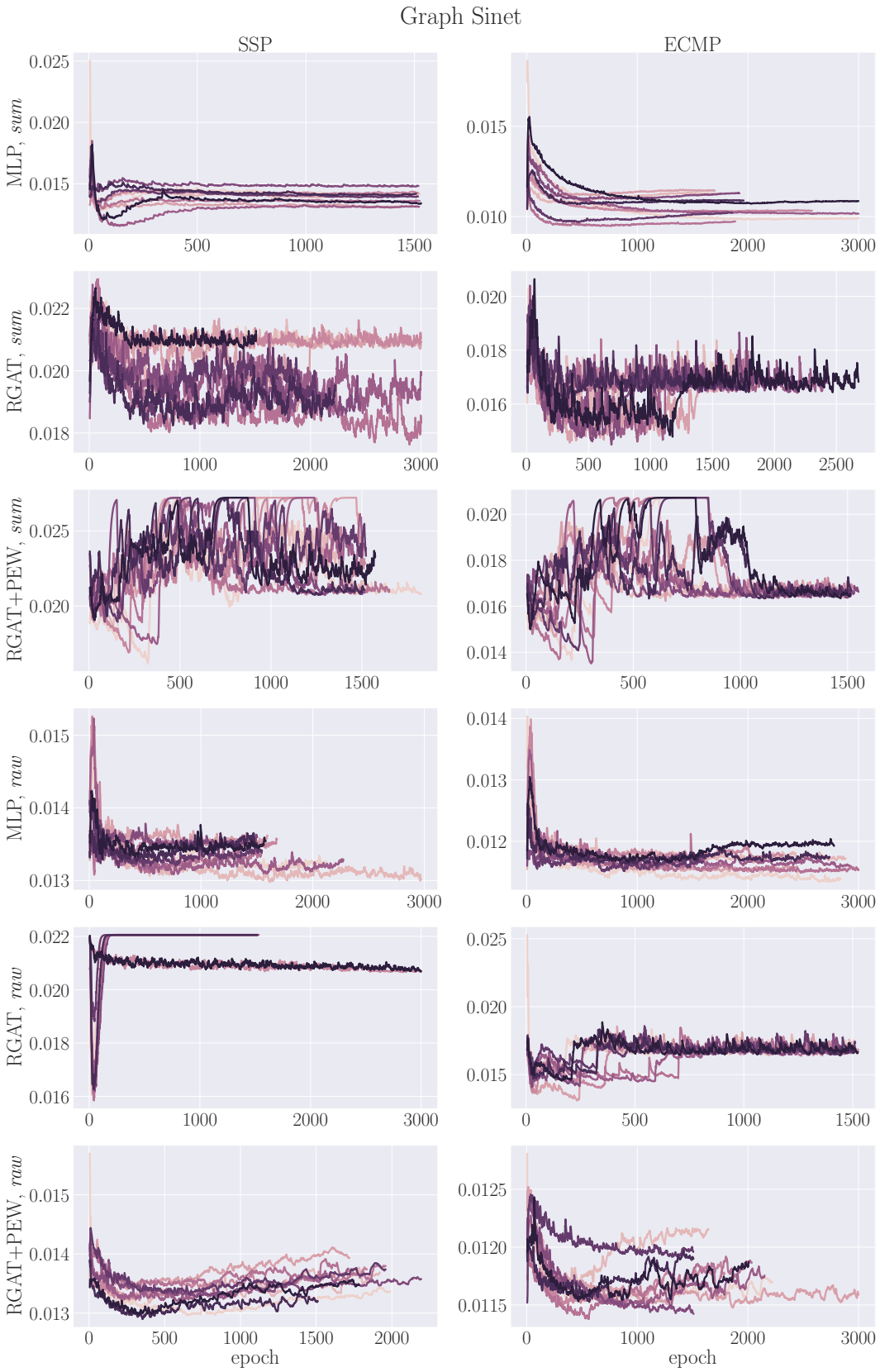


Figure D.14: Learning curves for Sinet.

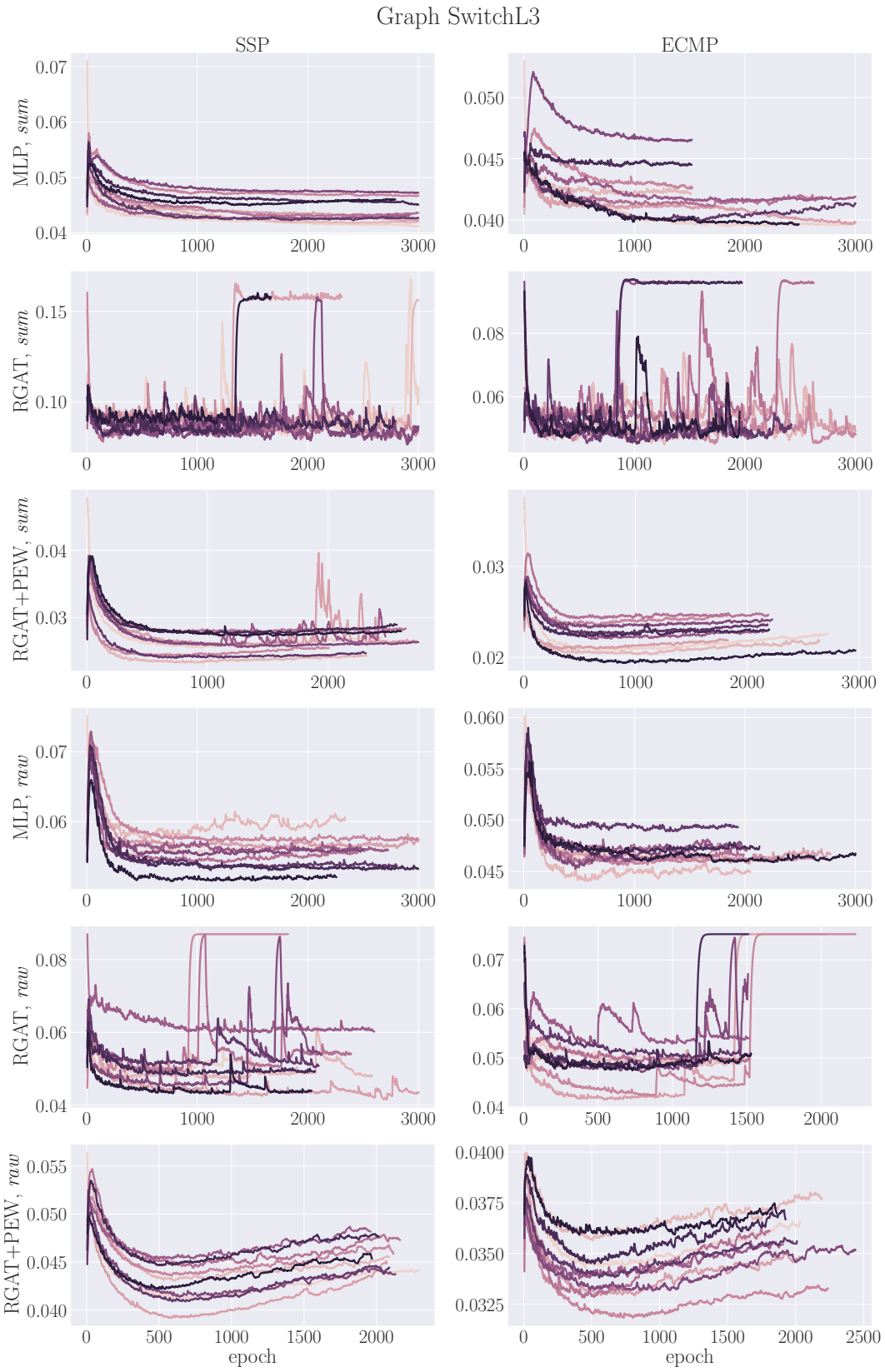


Figure D.15: Learning curves for SwitchL3.

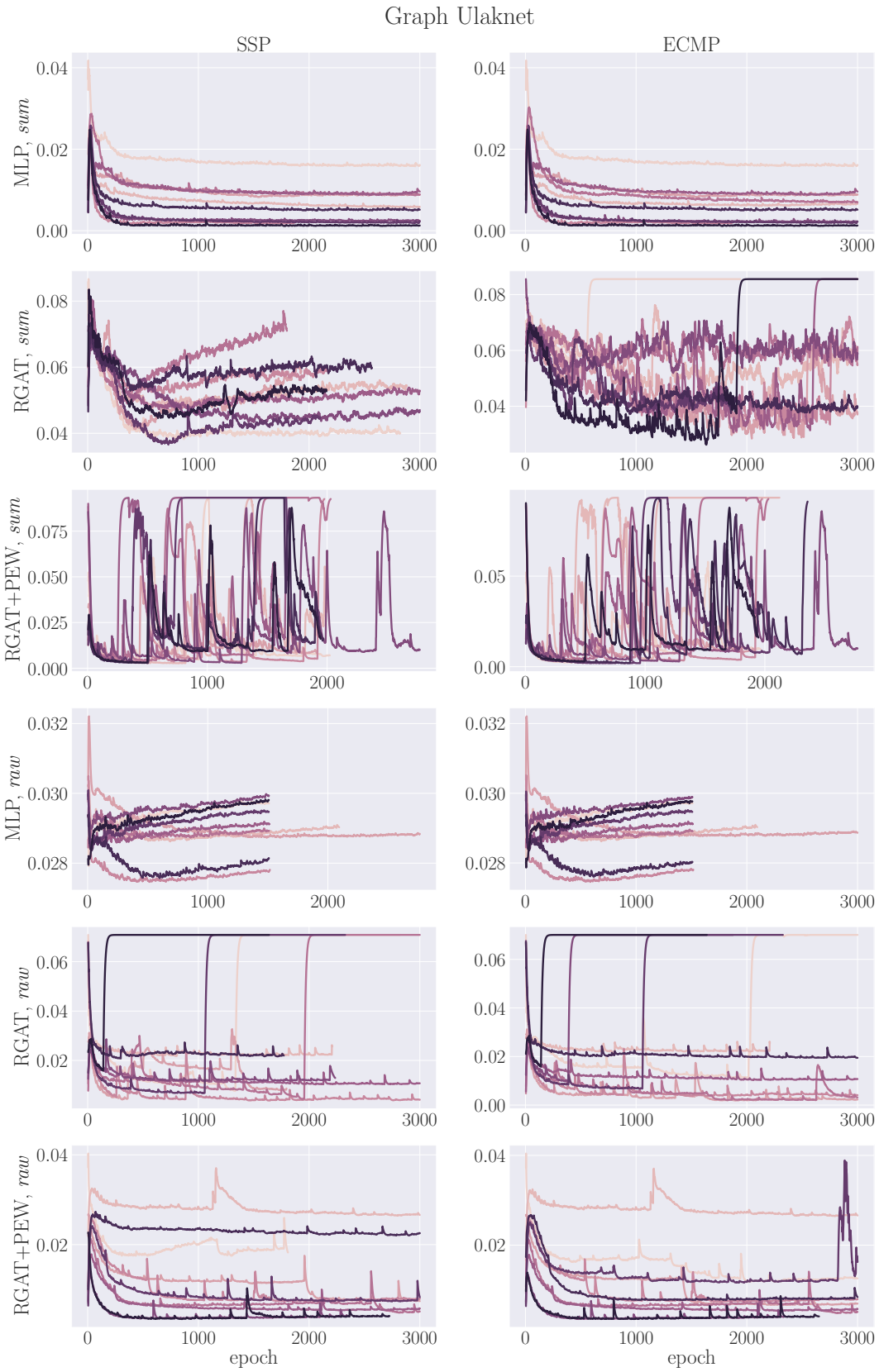


Figure D.16: Learning curves for Ulaknet.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[2] Kenshin Abe, Zijian Xu, Issei Sato, and Masashi Sugiyama. Solving NP-hard Problems on Graphs with Extended AlphaGo Zero. *arXiv preprint arXiv:1905.11623*, 2019.

[3] B. Abramson. Expected-outcome: a general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):182–193, 1990.

[4] Richa Agarwala, David L. Applegate, Donna Maglott, Gregory D. Schuler, and Alejandro A. Schäffer. A fast and scalable radiation hybrid map construction and integration strategy. *Genome Research*, 10(3):350–364, 2000.

[5] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *ICML*, 2020.

[6] Ravindra K. Ahuja. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[7] Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, and M. R. Reddy. Chapter 1 Applications of network optimization. In *Handbooks in Operations Research and Management Science*, volume 7 of *Network Models*, pages 1–83. Elsevier, 1995.

[8] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.

[9] Stefano Allesina, Antonio Bodini, and Mercedes Pascual. Functional links

and robustness in food webs. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1524):1701–1709, 2009.

[10] Paul Almasan, José Suárez-Varela, Bo Wu, Shihan Xiao, Pere Barlet-Ros, and Albert Cabello. Towards real-time routing optimization with deep reinforcement learning: Open challenges. In *HPSR*, 2021.

[11] Thomas Anthony, Zheng Tian, and David Barber. Thinking Fast and Slow with Deep Learning and Tree Search. In *NeurIPS*, 2017.

[12] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256, 2002.

[13] Daniel Awduche, Angela Chiu, Anwar Elwalid, Indra Widjaja, and XiPeng Xiao. Overview and principles of internet traffic engineering. RFC 3272, RFC Editor, May 2002.

[14] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*, 2016.

[16] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1999.

[17] Per Bak. *How Nature Works: the Science of Self-organized Criticality*. Copernicus, New York, NY, USA, 1996.

[18] Albert-László Barabási. *Network Science*. Cambridge University Press, 2016.

[19] Albert-László Barabási and Réka Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.

[20] Alain Barrat, Marc Barthélemy, and Alessandro Vespignani. *Dynamical Processes on Complex Networks*. Cambridge University Press, 2008.

[21] Thomas D. Barrett, William R. Clements, Jakob N. Foerster, and A. I. Lvovsky. Exploratory Combinatorial Optimization with Reinforcement Learning. In *AAAI*, 2020.

[22] Marc Barthélemy and Alessandro Flammini. Modeling urban street patterns. *Physical Review Letters*, 100(13):138702, 2008.

[23] Marc Barthélemy. Spatial networks. *Physics Reports*, 499(1-3), 2011.

[24] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Álvaro Sánchez-González, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[25] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *NeurIPS*, 2002.

[26] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *NeurIPS*. 2016.

[27] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *ICML*, 2017.

[28] Richard A. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[29] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. In *ICLR Workshops*, 2017.

[30] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009.

[31] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon. *European Journal of Operational Research*, 290:405–421, 2021.

[32] Alan A. Berryman. The origins and evolution of predator-prey theory. *Ecology*, 73(5):1530–1535, 1992.

[33] Giulia Bertagnolli, Riccardo Gallotti, and Manlio De Domenico. Quantifying efficient information exchange in real network flows. *Communications Physics*, 4(1):1–10, 2021.

[34] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 1995.

[35] Alina Beygelzimer, Geoffrey Grinstein, Ralph Linsker, and Irina Rish. Improving Network Robustness by Edge Modification. *Physica A*, 357:593–612, 2005.

[36] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[37] Monica Bianchini, Marco Gori, and Franco Scarselli. Inside PageRank. *ACM Transactions on Internet Technology*, 5(1):92–128, February 2005.

[38] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *ECML-PKDD*, 2013.

[39] Norman Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[40] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35 (3):268–308, 2003.

[41] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating Graphs via Random Walks. In *ICML*, 2018.

[42] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.

[43] Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *ICLR*, 2022.

[44] Justin A. Boyan and Michael L. Littman. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In *NeurIPS*, 1994.

[45] John Bradshaw, Brooks Paige, Matt J. Kusner, Marwin H. S. Segler, and José Miguel Hernández-Lobato. A Model to Search for Synthesizable Molecules. In *NeurIPS*, 2019.

[46] Yann Bramoullé and Rachel Kranton. Public goods in networks. *Journal of Economic Theory*, 135(1):478–494, 2007.

[47] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[48] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Netwowrks and ISDN Systems*, 30:107–117, 1998.

[49] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[50] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[51] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1): 1–43, 2012.

[52] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*, 2014.

[53] Ed Bullmore and Olaf Sporns. The economy of brain network organization. *Nature Reviews Neuroscience*, 13(5):336–349, 2012.

[54] Luciana S. Buriol, Mauricio G. C. Resende, Celso C. Ribeiro, and Mikkel Thorup. A memetic algorithm for OSPF routing. In *INFORMS Telecom*, 2002.

[55] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Hammerla. Relational graph attention networks. *arXiv preprint arXiv:1904.05811*, 2019.

[56] Duncan S. Callaway, M. E. J. Newman, Steven H. Strogatz, and Duncan J. Watts. Network robustness and fragility: Percolation on random graphs. *Physical Review Letters*, 85:5468–5471, 2000.

[57] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

[58] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. In *IJCAI*, 2021.

[59] Tristan Cazenave. Reflexive Monte-Carlo search. In *Computer Games Workshop*, 2007.

[60] Tristan Cazenave. Nested Monte-Carlo search. In *IJCAI*, 2009.

[61] Hale Cetinay, Karel Devriendt, and Piet Van Mieghem. Nodal vulnerability to targeted attacks in power grids. *Applied Network Science*, 3(1):34, 2018.

[62] Donald Chan and Daniel Mercier. Ic insertion: an application of the travelling salesman problem. *The International Journal of Production Research*, 27(10): 1837–1841, 1989.

[63] Olivier Chapelle and Lihong Li. An Empirical Evaluation of Thompson Sampling. In *NeurIPS*, 2011.

[64] Guillaume M. J-B. Chaslot, Mark H. M. Winands, H. Jaap Van Den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, 04(03):343–357, 2008.

[65] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, 2020.

[66] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *FOCS*, 2022.

[67] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *NeurIPS*, 2019.

[68] Marco Chiesa, Guy Kindler, and Michael Schapira. Traffic engineering with equal-cost-multipath: An algorithmic perspective. *IEEE/ACM Transactions on Networking*, 25(2):779–792, 2017.

[69] Marco Chiesa, Gábor Rétvári, and Michael Schapira. Oblivious routing in ip networks. *IEEE/ACM Transactions on Networking*, 26(3):1292–1305, 2018.

[70] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP*, 2014.

[71] Nicholas A. Christakis and James H. Fowler. The spread of obesity in a large social network over 32 years. *New England Journal of Medicine*, 357(4):370–379, 2007.

[72] Nicholas A. Christakis and James H. Fowler. The collective dynamics of smoking in a large social network. *New England Journal of Medicine*, 358(21): 2249–2258, 2008.

[73] Gian Paolo Cimellaro, Andrei M. Reinhorn, and Michel Bruneau. Framework for analytical quantification of disaster resilience. *Engineering Structures*, 32: 3639–3649, 2010.

[74] Geoff Clarke and John W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[75] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA*, 2018.

[76] Reuven Cohen, Keren Erez, Daniel ben Avraham, and Shlomo Havlin. Resilience of the Internet to Random Breakdowns. *Physical Review Letters*, 85 (21):4626–4628, 2000.

[77] Reuven Cohen, Keren Erez, Daniel ben Avraham, and Shlomo Havlin. Breakdown of the Internet under Intentional Attack. *Physical Review Letters*, 86(16): 3682–3685, 2001.

[78] Vincent Conitzer and Tuomas Sandholm. Complexity Results about Nash Equilibria. In *IJCAI*, 2003.

[79] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, 1971.

[80] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Fourth edition, 2022.

[81] Georges A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[82] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[83] Paulo da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *ACML*, 2020.

[84] Allan Dafoe, Edward Hughes, Yoram Bachrach, Tantum Collins, Kevin R. McKee, Joel Z Leibo, Kate Larson, and Thore Graepel. Open Problems in Cooperative AI. In *NeurIPS Cooperative AI Workshop*, 2020.

[85] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016.

[86] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *ICML*, 2018.

[87] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-Directed Variational Autoencoder for Structured Data. In *ICLR*, 2018.

[88] Jesper Dall and Michael Christensen. Random geometric graphs. *Physical Review E*, 66(1):016121, 2002.

[89] Luca Dall'Asta, Paolo Pin, and Abolfazl Ramezanpour. Statistical Mechanics of maximal independent sets. *Physical Review E*, 80(6):061136, 2009.

[90] Luca Dall'Asta, Paolo Pin, and Abolfazl Ramezanpour. Optimal Equilibria of the Best Shot Game. *Journal of Public Economic Theory*, 13(6):885–901, 2011.

[91] George B. Dantzig and Mukund N. Thapa. *Linear Programming, 1: Introduction*. Springer, 1997.

[92] George B. Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[93] Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Goal-directed graph construction using reinforcement learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2254): 20210168, 2021.

[94] Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Solving Graph-based Public Goods Games with Tree Search and Imitation Learning. In *Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[95] Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Graph Neural Modeling of Network Flows. *arXiv preprint arXiv:2209.05208*, 2022.

[96] Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Planning spatial networks with Monte Carlo tree search. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 479(2269):20220383, 2023.

[97] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR*, 2017.

[98] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The Complexity of Computing a Nash Equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[99] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. In *ICML Deep Generative Models Workshop*, 2018.

[100] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NeurIPS*, 2016.

[101] Vic Degraeve, Gilles Vandewiele, Femke Ongenae, and Sofie Van Hoecke. R-GCN: The R could stand for random. *arXiv preprint arXiv:2203.02424*, 2022.

[102] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *RSS*, 2011.

[103] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.

[104] Sybil Derrible and Christopher Kennedy. The complexity and robustness of metro networks. *Physica A: Statistical Mechanics and its Applications*, 389(17): 3678–3691, 2010.

[105] Bistra Dilkina, Katherine J. Lai, and Carla P. Gomes. Upgrading shortest paths in networks. In *CPAIOR*, 2011.

[106] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning Structural Node Embeddings Via Diffusion Wavelets. In *KDD*, 2018.

[107] Christoffel Doorman, Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Dynamic Network Reconfiguration for Entropy Maximization using Deep Reinforcement Learning. In *Proceedings of the First Learning on Graphs (LoG) Conference*, 2022.

[108] James E. Doran and Donald Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 294(1437):235–259, 1966.

[109] Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[110] Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation quotienting for generalizable neural combinatorial optimization. *arXiv preprint arXiv:2301.03313*, 2023.

[111] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep Reinforcement Learning in Large Discrete Action Spaces. In *ICML*, 2015.

[112] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NeurIPS*, 2015.

[113] Wendy Ellens, Floske M. Spieksma, Piet Van Mieghem, Almerima Jamakovic, and Robert E. Kooij. Effective graph resistance. *Linear Algebra and its Applications*, 435(10):2491–2506, 2011.

[114] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[115] M. Ericsson, Mauricio G. C. Resende, and Panos M. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization*, 6(3):299–333, 2002.

[116] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *ICLR*, 2020.

[117] Ernesto Estrada and Naomichi Hatano. Communicability in complex networks. *Physical Review E*, 77(3):036111, 2008.

[118] Shimon Even. *Graph Algorithms*. Cambridge University Press, 2011.

[119] Jonas K. Falkner and Lars Schmidt-Thieme. Learning to solve vehicle routing problems with time windows through joint attention. *arXiv preprint arXiv:2006.09100*, 2020.

[120] Nick Feamster and Jennifer Rexford. Why (and how) networks should run themselves. *arXiv preprint arXiv:1710.11583*, 2017.

[121] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Transactions On Networking*, 9(3): 265–279, 2001.

[122] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

[123] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016.

[124] Bernard Fortz and Mikkel Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *IEEE INFOCOM*, 2000.

[125] Bernard Fortz and Mikkel Thorup. Optimizing OSPF/IS-IS Weights in a Changing World. *IEEE Journal on Selected Areas in Communications*, 20(4): 756–767, 2002.

[126] Bernard Fortz and Mikkel Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29(1):13–48, 2004.

[127] Feng Fu, Daniel I. Rosenbloom, Long Wang, and Martin A. Nowak. Imitation dynamics of vaccination behaviour on social networks. *Proceedings of the Royal Society B: Biological Sciences*, 278(1702):42–49, 2011.

[128] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *ICLR*, 2018.

[129] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, 2018.

[130] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[131] Andrea Galeotti, Sanjeev Goyal, Matthew O. Jackson, Fernando Vega-Redondo, and Leeat Yariv. Network Games. *Review of Economic Studies*, 77(1): 218–244, 2009.

[132] Alexander A. Ganin, Emanuele Massaro, Alexander Gutfraind, Nicolas Steen, Jeffrey M. Keisler, Alexander Kott, Rami Mangoubi, and Igor Linkov. Operational resilience: Concepts, design and analysis. *Scientific Reports*, 6(1):1–12, 2016.

[133] Javier García and Fernando Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 16(42):44, 2015.

[134] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co, 1979.

[135] Vikas Garg and Tommi Jaakkola. Learning tree structured potential games. In *NeurIPS*, 2016.

[136] Michael T. Gastner and M. E. J. Newman. Shape and efficiency in spatial distribution networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2006(01):P01015, 2006.

[137] Michael T. Gastner and M. E. J. Newman. The spatial structure of networks. *The European Physical Journal B*, 49(2):247–252, 2006.

[138] Steven Gay, Pierre Schaus, and Stefano Vissicchio. Repetita: Repeatable experiments for performance evaluation of traffic-engineering algorithms. *arXiv preprint arXiv:1710.08665*, 2017.

[139] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *ICML*, 2007.

[140] Dobrik Georgiev and Pietro Liò. Neural bipartite matching. In *ICML Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020.

[141] Dobrik Georgiev, Pietro Barbiero, Dmitry Kazhdan, Petar Veličković, and Pietro Liò. Algorithmic concept-based explainable reasoning. In *AAAI*, 2022.

[142] Fabien Geyer and Georg Carle. Learning and Generating Distributed Routing Protocols Using Graph-Based Deep Learning. In *Big-DAMA*, 2018.

[143] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

[144] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *ICML*, 2017.

[145] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *NeurIPS*, 2014.

[146] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[147] Sanjeev Goyal. *Connections: An Introduction to the Economics of Networks*. Princeton University Press, 2012.

[148] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

[149] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *KDD*, 2016.

[150] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016.

[151] Jim Guichard, François Le Faucheur, and Jean-Philippe Vasseur. *Definitive MPLS Network Designs*. Cisco Press, 2005.

[152] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models. *arXiv preprint arXiv:1705.10843*, 2018.

[153] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L. Lewis, and Xiaoshi Wang. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In *NeurIPS*, 2014.

[154] Nikola Gvozdiev, Stefano Vissicchio, Brad Karp, and Mark Handley. On low-latency-capable topologies, and their impact on the design of intra-domain routing. In *SIGCOMM*, 2018.

[155] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Central Science*, 4(2):268–276, 2018.

[156] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[157] Hamed Haddadi and Olivier Bonaventure (Editors). Recent Advances in Networking, 2013.

[158] Aric Hagberg, Pieter Swart, and Daniel S. Chult. Exploring network structure, dynamics, and function using networkx. In *SciPy*, 2008.

[159] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40(3): 52–74, 2017.

[160] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NeurIPS*, 2017.

[161] Frank Harary and Edgar M. Palmer. *Graphical Enumeration*. Academic Press, New York, 1973.

[162] Renaud Hartert, Stefano Vissicchio, Pierre Schaus, Olivier Bonaventure, Clarence Filsfils, Thomas Telkamp, and Pierre Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. *ACM SIGCOMM Computer Communication Review*, 45(4):15–28, 2015.

[163] Yehuda Hassin and David Peleg. Sparse communication networks and efficient routing in the plane (extended abstract). In *PODC*, 2000.

[164] Ji He, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng. Deep Reinforcement Learning with a Combinatorial Action Space for Predicting Popular Reddit Threads. In *EMNLP*, 2016.

[165] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep Convolutional Networks on Graph-Structured Data. *arXiv preprint arXiv:1506.05163*, 2015.

[166] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI*. 2018.

[167] Jack Hirshleifer. From weakest-link to best-shot: The voluntary provision of public goods. *Public Choice*, 41(3):371–386, 1983.

[168] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NeurIPS*, 2016.

[169] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[170] Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.

[171] Richard A. Holley and Thomas M. Liggett. Ergodic theorems for weakly interacting infinite systems and the voter model. *The Annals of Probability*, pages 643–663, 1975.

[172] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical Review E*, 65(5), 2002.

[173] Jean Honorio and Luis E. Ortiz. Learning the structure and parameters of large-population graphical games from behavioral data. *Journal of Machine Learning Research*, 16(1):1157–1210, 2015.

[174] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(04):439–562, 2006.

[175] Oliver Hope and Eiko Yoneki. GDDR: GNN-based Data-Driven Routing. In *ICDCS*, 2021.

[176] John J. Hopfield and David W. Tank. "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152, 1985.

[177] C. Hopps. Analysis of an equal-cost multi-path algorithm. RFC 2992, RFC Editor, November 2000.

[178] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[179] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *ECAI*, 2020.

[180] T. Chiang Hu. Multi-commodity network flows. *Operations Research*, 11(3): 344–360, 1963.

[181] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[182] Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, Andreea Deac, Beatrice Bevilacqua, Yaroslav Ganin, Charles Blundell, and Petar Veličković. A generalist neural algorithmic learner. In *LoG*, 2022.

[183] Mohammad Irfan and Luis Ortiz. A game-theoretic approach to influence in networks. In *AAAI*, 2011.

[184] Swami Iyer, Timothy Killingback, Bala Sundaram, and Zhen Wang. Attack Robustness and Centrality of Complex Networks. *PLoS ONE*, 8(4):e59613, April 2013.

[185] Matthew O. Jackson and Yves Zenou. Chapter 3 - Games on Networks. In *Handbook of Game Theory with Economic Applications*, volume 4, pages 95–163. Elsevier, 2015.

[186] Adam B. Jaffe, Manuel Trajtenberg, and Rebecca Henderson. Geographic localization of knowledge spillovers as evidenced by patent citations. *The Quarterly Journal of Economics*, 108(3):577–598, 1993.

[187] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, 2013.

[188] Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. Wiley, New York, 1995.

[189] Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. Unleashing the potential of data-driven networking. In *COMSNETS*, 2017.

[190] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction Tree Variational Autoencoder for Molecular Graph Generation. In *ICML*, 2018.

[191] Syu-Ning Johnn, Victor-Alexandru Darvariu, Julia Handl, and Joerg Kalcsics. Graph Reinforcement Learning for Operator Selection in the ALNS Metaheuristic. In *Proceedings of the International Conference in Optimization and Learning (OLA2023)*, 2023.

[192] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[193] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, pages 1–29, 2022.

[194] Marcus Kaiser and Claus C. Hilgetag. Spatial growth of real-world networks. *Physical Review E*, 69(3):036103, 2004.

[195] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *SIGCOMM*, 2005.

[196] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[197] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. In *CVPR*, 2019.

[198] Michael Kearns, Michael L. Littman, and Satinder Singh. Graphical Models for Game Theory. In *UAI*, 2001.

[199] Ralph L. Keeney, Howard Raiffa, and Richard F. Meyer. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge University Press, 1993.

[200] David Kempe, Sixie Yu, and Yevgeniy Vorobeychik. Inducing Equilibria in Networked Public Goods Games through Network Structure Modification. In *AAMAS*, 2020.

[201] Donald Kennedy. Sustainability and the Commons. *Science*, 302(5652):1861–1861, 2003.

[202] William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London Series A*, 115(772):700–721, 1927.

[203] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*, 2017.

[204] Elias B. Khalil, Christopher Morris, and Andrea Lodi. MIP-GNN: A data-driven framework for guiding combinatorial solvers. In *AAAI*, 2022.

[205] Minsu Kim and Jinkyoo Park. Learning collaborative policies to solve np-hard routing problems. In *NeurIPS*, 2021.

[206] Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. In *NeurIPS*, 2022.

[207] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

[208] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *ICLR*, 2013.

[209] Thomas N. Kipf and Max Welling. Variational Graph Auto-Encoders. In *NeurIPS '16 Bayesian Deep Learning Workshop*, 2016.

[210] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017.

[211] Scott Kirkpatrick, C. Daniel Gelatt Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[212] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[213] Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In *ECML*, 2006.

[214] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT Press, 2009.

[215] Peter Kollock. Social Dilemmas: The Anatomy of Cooperation. *Annual Review of Sociology*, 24(1):183–214, 1998.

[216] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! In *ICLR*, 2019.

[217] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[218] Jérôme Kunegis. KONECT: the Koblenz network collection. In *WWW Companion*, 2013.

[219] Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar Variational Autoencoder. In *ICML*, 2017.

[220] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In *NeurIPS*, 2020.

[221] Alisa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[222] Alexandre Laterre, Yunguan Fu, Mohamed Khalil Jabri, Alain-Sam Cohen, David Kas, Karl Hajjar, Torbjorn S. Dahl, Amine Kerkeni, and Karim Beguir.

Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. *arXiv preprint arXiv:1807.01672*, 2018.

[223] Vito Latora and Massimo Marchiori. Efficient Behavior of Small-World Networks. *Physical Review Letters*, 87(19):198701, 2001.

[224] Vito Latora and Massimo Marchiori. Economic small-world behavior in weighted networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 32(2):249–263, 2003.

[225] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998.

[226] John O. Ledyard. *Public Goods: A Survey of Experimental Research*, pages 111–194. Princeton University Press, 1995.

[227] Yan Leng, Xiaowen Dong, Junfeng Wu, and Alex Pentland. Learning quadratic games on networks. In *ICML*, 2020.

[228] Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.

[229] Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML*, 2013.

[230] Vadim Levit, Zohar Komarovsky, Tal Grinshpoun, and Amnon Meisels. Incentive-based search for efficient equilibria of the public goods game. *Artificial Intelligence*, 262:142–162, 2018.

[231] Kevin Leyton-Brown and Yoav Shoham. Essentials of game theory: A concise multidisciplinary introduction. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2(1):1–88, 2008.

[232] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. In *ICLR*, 2017.

[233] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning Deep Generative Models of Graphs. In *ICML*, 2018.

[234] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. In *NeurIPS*, 2018.

[235] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. Efficient Graph Generation with Graph Recurrent Attention Networks. In *NeurIPS*, 2019.

[236] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[237] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *ICLR*, 2020.

[238] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[239] Yishay Mansour and David Peleg. An approximation algorithm for minimum cost network design, Technical Report CS94-22. *Weizmann Institute of Science, Faculty of Mathematical Sciences*, 1994.

[240] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably Powerful Graph Networks. In *NeurIPS*, 2019.

[241] James L. McClelland and David E. Rumelhart. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MIT Press, 1987.

[242] James L. McClelland and David E. Rumelhart. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Psychological and Biological Models*. MIT Press, 1987.

[243] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[244] Richard D. McKelvey and Andrew McLennan. Computation of equilibria in finite games. *Handbook of Computational Economics*, 1:87–142, 1996.

[245] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[246] Wes McKinney. pandas: a foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14(9):1–9, 2011.

[247] Wided Medjroubi, Ulf Philipp Müller, Malte Scharf, Carsten Matke, and David Kleinhans. Open Data in Power Grid Modelling: New Approaches Towards Transparent Grid Models. *Energy Reports*, 3:14–21, 2017.

[248] Eli Meirom, Haggai Maron, Shie Mannor, and Gal Chechik. Controlling graph dynamics with reinforcement learning and graph neural networks. In *ICML*, 2021.

[249] Manfred Milinski, Dirk Semmann, Hans-Jürgen Krambeck, and Jochem Marotzke. Stabilizing the Earth's climate is not a losing game: Supporting evidence from public goods experiments. *PNAS*, 103(11):3994–3998, 2006.

[250] Marvin Minsky and Seymour Papert. *Perceptrons: an Introduction to Computational Geometry*. MIT Press, 1969.

[251] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

[252] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *ICML*, 2016.

[253] P. Read Montague, Peter Dayan, and Terrence J. Sejnowski. A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *The Journal of Neuroscience*, 16(5):1936–1947, 1996.

[254] Federico Monti, Michael M. Bronstein, and Xavier Bresson. Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. In *ICML*, 2017.

[255] Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning–based column selection for column generation. *Transportation Science*, 55(4):815–831, 2021.

[256] Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. In *NeurIPS*, 2019.

[257] John Moy. OSPF Version 2. RFC 2328, RFC Editor, April 1998.

[258] John Nash. Some games and machines for playing them. Technical Report D-1164, Rand Corporation, 1952.

[259] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takáč. Reinforcement Learning for Solving the Vehicle Routing Problem. In *NeurIPS*, 2018.

[260] M. E. J. Newman. Assortative mixing in networks. *Physical Review Letters*, 89 (20), 2002.

[261] M. E. J. Newman. Mixing patterns in networks. *Physical Review E*, 67(2), February 2003.

[262] M. E. J. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45(2), 2003.

[263] M. E. J. Newman. *Networks*. Oxford University Press, 2018.

[264] Andrew Y. Ng and Stuart J. Russell. Algorithms for Inverse Reinforcement Learning. In *ICML*, 2000.

[265] OEIS Foundation. The on-line encyclopedia of integer sequences, 2020. URL `https://oeis.org/A001187`.

[266] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *NeurIPS*, 2015.

[267] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *arXiv preprint arXiv:1609.03499*, 2016.

[268] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric Transitivity Preserving Graph Embedding. In *KDD*, 2016.

[269] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. Transfer graph neural networks for pandemic forecasting. In *AAAI*, 2021.

[270] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[271] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[272] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[273] Srinivas Peeta, F. Sibel Salman, Dilek Gunnec, and Kannan Viswanath. Pre-disaster investment decisions for strengthening a highway network. *Computers & Operations Research*, 37(10):1708–1719, 2010.

[274] Tiago P. Peixoto. Hierarchical block structures and high-resolution model selection in large networks. *Physical Review X*, 4(1):011047, 2014.

[275] Tiago P. Peixoto. Reconstructing networks with unknown and heterogeneous errors. *Physical Review X*, 8(4):041011, 2018.

[276] Tiago P. Peixoto. Bayesian stochastic blockmodeling. *Advances in Network Clustering and Blockmodeling*, pages 289–332, 2019.

[277] Mathew D. Penrose. *Random Geometric Graphs*. Oxford University Press, 2003.

[278] Mathew D. Penrose. Connectivity of soft random geometric graphs. *The Annals of Applied Probability*, 26(2):986–1028, 2016.

[279] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *KDD*, 2014.

[280] Leonid Peshkin and Virginia Savova. Reinforcement learning for adaptive routing. In *IJCNN*, 2002.

[281] Jervis Pinto and Alan Fern. Learning partial policies to speedup mdp tree search via reduction to iid learning. *The Journal of Machine Learning Research*, 18(1):2179–2213, 2017.

[282] Pavel G. Polishchuk, Timur I. Madzhidov, and Alexandre Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of Computer-aided Molecular Design*, 27(8):675–679, 2013.

[283] Dean A. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In *NeurIPS*, 1988.

[284] Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[285] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *STOC*, 2008.

[286] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *ICLR*, 2016.

[287] Joao Reis, Miguel Rocha, Truong Khoa Phan, David Griffin, Franck Le, and Miguel Rio. Deep neural networks for network routing. In *IJCNN*, 2019.

[288] Leonardo F. R. Ribeiro, Pedro H. P. Savarese, and Daniel R. Figueiredo. struc2vec: Learning Node Representations from Structural Identity. In *KDD*, 2017.

[289] Martin Riedmiller. Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *ECML*, 2005.

[290] Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[291] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[292] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[293] Christopher D. Rosin. Nested Rollout Policy Adaptation for Monte Carlo Tree Search. In *IJCAI*, 2011.

[294] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[295] Emanuele Rossi, Federico Monti, Yan Leng, Michael Bronstein, and Xiaowen Dong. Learning to infer structures of network games. In *ICML*, 2022.

[296] Camille Roth, Soong Moon Kang, Michael Batty, and Marc Barthélemy. A long-time limit for world subway networks. *Journal of The Royal Society Interface*, 9(75):2540–2550, 2012.

[297] Matthew Roughan. Simplifying the synthesis of internet traffic matrices. *ACM SIGCOMM Computer Communication Review*, 35(5):93–96, 2005.

[298] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[299] Adam Runions, Martin Fuhrer, Brendan Lane, Pavol Federl, Anne-Gaëlle Rolland-Lagan, and Przemyslaw Prusinkiewicz. Modeling and visualization of leaf venation patterns. In *SIGGRAPH*. 2005.

[300] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Unveiling the potential of graph neural networks for network modeling and optimization in SDN. In *SOSR*, 2019.

[301] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a Modern Approach*. Prentice Hall, Third edition, 2010.

[302] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[303] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[304] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR*, 2016.

[305] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, pages 593–607. Springer, 2018.

[306] Christian M. Schneider, André A. Moreira, Joao S. Andrade, Shlomo Havlin, and Hans J. Herrmann. Mitigation of malicious attacks on networks. *PNAS*, 108(10):3838–3841, 2011.

[307] Christian M. Schneider, Nuri Yazdani, Nuno A. M. Araújo, Shlomo Havlin, and Hans J. Herrmann. Towards designing robust coupled networks. *Nature Scientific Reports*, 3(1), December 2013.

[308] Alexander Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in Operations Research and Management Science*, 12:1–68, 2005.

[309] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[310] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[311] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-Walk: Learning to Walk over Graphs using Monte Carlo Tree Search. In *NeurIPS*, 2018.

[312] David Silver. Reinforcement Learning of Local Shape in the Game of Go. In *IJCAI*, 2007.

[313] David Silver, J. Andrew Bagnell, and Anthony Stentz. High performance outdoor navigation from overhead data using imitation learning. In *RSS*. 2008.

[314] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

[315] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[316] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[317] Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. In *ICANN*, 2018.

[318] Kate A. Smith. Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research. *INFORMS Journal on Computing*, 11(1): 15–34, 1999.

[319] Ricard V. Solé, Martí Rosas-Casals, Bernat Corominas-Murtra, and Sergi Valverde. Robustness of the European power grids under intentional attack. *Physical Review E*, 77(2):026102, 2008.

[320] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3): 714–735, 1997.

[321] Dietrich Stauffer and Ammon Aharony. *Introduction to Percolation Theory*. Taylor & Francis, 1992.

[322] Peter Stone. TPOT-RL Applied to Network Routing. In *ICML*, 2000.

[323] Ethan Stump, Ali Jadbabaie, and Vijay Kumar. Connectivity management in mobile robot teams. In *ICRA*, 2008.

[324] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[325] John E. Sulston, Einhard Schierenberg, John G. White, and J. Nichol Thomson. The embryonic cell lineage of the nematode caenorhabditis elegans. *Developmental Biology*, 100(1):64–119, 1983.

[326] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[327] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[328] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *NeurIPS*. 2017.

[329] Toshi Tanizawa, Gerald Paul, Reuven Cohen, Shlomo Havlin, and H. Eugene Stanley. Optimization of network robustness to waves of targeted and random attacks. *Physical Review E*, 71(4), 2005.

[330] Nigel Tao, Jonathan Baxter, and Lex Weaver. A Multi-Agent, Policy-Gradient approach to Network Routing. In *ICML*, pages 553–560, 2001.

[331] Éva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

[332] Gerald Tesauro and Gregory R. Galperin. On-line Policy Improvement using Monte-Carlo Search. In *NeurIPS*, 1997.

[333] Paul R. Thie and Gerard E. Keough. *An Introduction to Linear Programming and Game Theory*. John Wiley & Sons, 2011.

[334] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-Mixer: An all-MLP architecture for vision. In *NeurIPS*, 2021.

[335] Paolo Toth and Daniele Vigo, editors. *Vehicle routing: problems, methods, and applications*. Society for Industrial and Applied Mathematics, Second edition, 2015.

[336] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32(4):425–443, 1969.

[337] Rakshit Trivedi and Hongyuan Zha. Learning strategic network emergence games. In *NeurIPS*, 2020.

[338] Rakshit Trivedi, Jiachen Yang, and Hongyuan Zha. GraphOpt: Learning Optimization Models of Graph Formation. In *ICML*, 2020.

[339] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to route. In *ACM HotNets*, 2017.

[340] André X. C. N. Valente, Abhijit Sarkar, and Howard A. Stone. Two-Peak and Three-Peak Optimal Complex Networks. *Physical Review Letters*, 92(11), 2004.

[341] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2016.

[342] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

[343] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[344] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural Execution of Graph Algorithms. In *ICLR*, 2020.

[345] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *ICML*, 2018.

[346] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In *NeurIPS*, 2015.

[347] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305, 2008.

[348] Huijuan Wang and Piet Van Mieghem. Algebraic connectivity optimization via link addition. In *Bionetics*, 2008.

[349] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural Architecture Search using Deep Neural Networks and Monte Carlo Tree Search. In *AAAI*, 2020.

[350] Xiangrong Wang, Evangelos Pournaras, Robert E. Kooij, and Piet Van Mieghem. Improving robustness of complex networks via the effective graph resistance. *The European Physical Journal B*, 87(9):221, 2014.

[351] Michael L. Waskom. Seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[352] Christopher J. C. H. Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[353] Duncan J. Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, 2002.

[354] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440, 1998.

[355] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[356] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

[357] Walter E. Westman. Measuring the inertia and resilience of ecosystems. *BioScience*, 28(11):705–710, 1978.

[358] Norbert Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*. MIT Press, 1948.

[359] Matthew J. Williams and Mirco Musolesi. Spatio-temporal networks: Reachability, centrality and robustness. *Royal Society Open Science*, 3(6):160–196, 2016.

[360] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

[361] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[362] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[363] John Wreathall. Properties of resilient organizations: an initial view. In *Resilience Engineering*, pages 275–285. CRC Press, 2017.

[364] Xiaojian Wu, Daniel Sheldon, and Shlomo Zilberstein. Optimizing resilience in large scale networks. In *AAAI*, 2016.

[365] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[366] Zhi-Xi Wu and Petter Holme. Onion structure and network robustness. *Physical Review E*, 84(2), 2011.

[367] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *ICLR*, 2020.

[368] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM*, 2018.

[369] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*, 2018.

[370] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*, 2018.

[371] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. GNNExplainer: Generating explanations for graph neural networks. In *NeurIPS*, 2019.

[372] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *NeurIPS*, 2018.

[373] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *ICML*, 2018.

[374] Lantao Yu, Jiaming Song, and Stefano Ermon. Multi-agent adversarial inverse reinforcement learning. In *ICML*, 2019.

[375] Sixie Yu, Kai Zhou, P. Jeffrey Brantingham, and Yevgeniy Vorobeychik. Computing Equilibria in Binary Networked Public Goods Games. In *AAAI*, 2019.

[376] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J. Mankowitz, and Shie Mannor. Learn What Not to Learn: Action Elimination with Deep Reinforcement Learning. In *NeurIPS*, 2018.

[377] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

[378] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *NeurIPS*, 2020.

[379] Denghui Zhang, Zixuan Yuan, Hao Liu, Hui Xiong, et al. Learning to walk with dual agents for knowledge graph reasoning. In *AAAI*, 2022.

[380] Junjie Zhang, Minghao Ye, Zehua Guo, Chen-Yu Yen, and H. Jonathan Chao. CFR-RL: Traffic engineering with reinforcement learning in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10):2249–2259, 2020.

[381] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

[382] Lovro Šubelj and Marko Bajec. Robust network community detection using balanced propagation. *The European Physical Journal B*, 81(3):353–362, 2011.