POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Application of Deep Reinforcement Learning to routing and scheduling

SEYED PEYMAN KAFAEI KASHEFI
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie industriel

Octobre 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Application of Deep Reinforcement Learning to routing and scheduling

présentée par **Seyed Peyman KAFAEI KASHEFI**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Michel GENDREAU, président
Louis-Martin ROUSSEAU, membre et directeur de recherche
Quentin CAPPART, membre et codirecteur de recherche
Nicolas CHAPADOS, membre et codirecteur de recherche
Antoine LEGRAIN, membre
Kevin TIERNEY, membre externe

DEDICATION

To Mahdis,
whose endless love, belief in me, and sacrifice made this journey possible
To my parents,
for their immense encouragement and support

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Prof. Louis-Martin Rousseau, and my co-advisors Prof. Quentin Cappart and Dr. Nicolas Chapados for their unwavering support, invaluable advice, and motivation throughout my Ph.D. journey. Your insightful feedback and guidance pushed my thinking and brought my work to a higher level.

In addition, I would like to offer my special thanks to Dr. Marc-Andre Renaud, Dr. Stephane Bedwani, and doctors and researchers at the Department of radiation oncology of Centre Hospitalier de l'Université de Montréal (CHUM) for their patience and constant assistance towards my first work.

Furthermore, I acknowledge Mitacs Inc. for providing financial aid for the fourth year of my Ph.D. study.

And last but not least, I would like to extend my gratitude to my friends at HANALOG, CORAIL, CIRRELT, and GERAD for their support and who made my Ph.D. journey more exciting.

RÉSUMÉ

Récemment, on a assisté à une montée en puissance des techniques d'apprentissage automatique dans le domaine de l'optimisation. Les méthodes d'apprentissage supervisé peuvent exploiter des structures sous-jacentes dans des multitudes de problèmes et utiliser cette expertise pour aider à trouver plus efficacement la solution de nouvelles instances de ces problèmes. Dans l'apprentissage par renforcement (Reinforcement Learning, RL), cependant, un agent interagit avec l'environnement contrôlé par les contraintes d'un problème d'optimisation. Une fonction de récompense qui s'aligne sur la fonction objectif du problème d'optimisation conduit le processus de recherche itératif à prendre de meilleures décisions pour atteindre la politique optimale. Comme les problèmes d'optimisation combinatoire peuvent généralement être formulés comme des problèmes de prise de décision séquentielle, ils peuvent être modélisés comme des processus de décision de Markov (Markov Decision Process, MDP). L'utilisation de méthodes de recherche exhaustive peut devenir prohibitive pour ces problèmes, c'est pourquoi des méthodes d'apprentissage par renforcement peuvent être utilisées pour les résoudre. En raison de la nature combinatoire de ces problèmes, dans de nombreux cas, les algorithmes de programmation en nombres entiers standard des logiciels de programmation mathématique commerciaux ne peuvent pas résoudre efficacement les modèles correspondants. D'un autre point de vue, de nombreux problèmes issus de cas pratiques réels sont de nature online, ce qui signifie que certaines informations ne sont pas disponibles au départ et qu'elles seront révélées au fil du temps. Dans ce cas, l'information des paramètres stochastiques n'est pas connue à l'avance. L'utilisation de la programmation linéaire mixte en nombres entiers pour ce type de problèmes serait coûteuse en termes de calculs. Néanmoins, les méthodes d'apprentissage par renforcement fonctionnent également très bien pour ces problèmes dynamiques. En outre, la majorité des problèmes d'optimisation combinatoire peuvent être définis sur des graphes. Les récentes avancées dans le domaine de l'apprentissage profond, en particulier les réseaux de neurones graphiques, ont permis de remarquer l'application des méthodes d'apprentissage par renforcement à ces problèmes. Les réseaux neuronaux graphiques capturent l'information relationnelle cachée entre tous les éléments (nœuds et arêtes) des graphes pour faire de meilleures prédictions. Dans cette thèse, nous tirons parti de l'apprentissage par renforcement et des réseaux neuronaux graphiques pour développer des algorithmes efficaces pour les problèmes d'optimisation combinatoire dans diverses applications.

Dans le premier projet, nous développons une méthode d'apprentissage profond par renforcement pour trouver les meilleures orientations de faisceaux pour irradier les patients sous radiothérapie pour le traitement du cancer en utilisant le système Cyberknife. Nous trouvons simultanément la trajectoire pour les traverser afin de réduire le temps de traitement tout en maintenant la qualité du traitement par rapport à l'utilisation de toutes les orientations possibles du faisceau. En utilisant l'agent entraîné, nous pouvons générer des plans de traitement spécifiques au patient au lieu d'utiliser un ensemble fixe de faisceaux pour chaque patient et chaque site cancéreux. À cette fin, nous représentons le problème par un graphe et appliquons des réseaux neuronaux de graphe pour apprendre une nouvelle représentation du graphe et les caractéristiques de ses éléments correspondants. Grâce à cette méthode, nous pouvons utiliser de nombreuses caractéristiques différentes concernant l'anatomie des patients et les informations dosimétriques. Nous proposons un algorithme de Q-Learning profond pour entraîner un agent à sélectionner un sous-ensemble de faisceaux et l'ordre dans lequel les visiter. Les résultats sont ensuite transmis à un problème d'optimisation directe de l'ouverture où d'autres paramètres de traitement sont calculés. Les résultats montrent que l'utilisation de la méthode proposée réduit le temps de traitement tout en maintenant la qualité du traitement par rapport aux traitements cliniques de base.

Dans le deuxième projet, nous proposons un algorithme d'apprentissage par renforcement profond basé sur les réseaux de neurones graphiques pour résoudre le problème de routage dynamique et d'affectation de longueurs d'onde dans un contexte de problème d'optimisation des réseaux de télécommunications. Les demandes de trafic entrant arrivent dynamiquement en suivant une distribution inconnue. Nous modélisons ce problème comme un MDP. En raison de la nature dynamique du problème, nous développons un algorithme RL profond avec l'objectif de maximiser les demandes de trafic admises. Les résultats montrent que l'algorithme proposé est plus performant que les heuristiques largement utilisées (tant dans la littérature de recherche que dans la pratique). En adoptant cet algorithme, nous fournissons un modèle qui résout le routage et l'affectation des ressources (longueurs d'onde) en même temps. Nous considérons un cas d'information parfaite, où toutes les informations sont connues de l'optimiseur, et avons formulé un modèle de programmation mathématique résolu avec des solveurs commerciaux comme base de référence (limite supérieure) pour comparer les résultats de l'algorithme.

Dans le troisième projet, nous abordons le problème d'ordonnancement dynamique sous contraintes multi-ressources qui découle d'un problème réel dans l'industrie. Nous considérons un cas où chaque tâche a une date limite spécifique et une date d'expiration. Nous appliquons des Q-Networks avec approximation de fonction pour générer des solutions qui minimisent les coûts totaux de retard et les coûts d'expiration. Nous définissons d'abord le problème comme un graphe et attribuons plusieurs caractéristiques à chaque nœud. Pour capturer toutes les relations latentes entre les éléments, nous appliquons des Graph Attention Networks comme décodeurs pour apprendre une nouvelle représentation du graphe d'entrée à chaque instant de décision. L'agent RL sélectionne une tâche (parmi l'ensemble des tâches arrivées) et lui affecte des ressources afin de minimiser les pénalités cumulées attendues. Nous comparons les résultats avec différentes méthodes de la littérature. Nous formulons également le problème avec une information parfaite à l'aide de la programmation par contraintes, afin d'obtenir une borne inférieure pour les solutions.

ABSTRACT

Recently there has been a surge to benefit from machine learning techniques in the field of optimization. Supervised learning methods can infer the innate patterns in multitudes of problems and use that expertise to help find the solution for new instances of the problems more efficiently. In Reinforcement Learning (RL), however, an agent interacts with the environment that is controlled by the constraints of an optimization problem. A reward function that aligns with the objective function of the optimization problem drives the iterative search process toward making better decisions to reach the optimal policy. Since Combinatorial Optimization (CO) problems can generally be formulated as sequential decision-making problems, they can be modeled as Markov Decision Process (MDP). Employing exhaustive search methods can become prohibitive for these problems, therefore RL methods can be employed to solve them. Because of the combinatorial nature of these problems, in many cases, standard integer programming algorithms in commercial mathematical programming software packages cannot solve the corresponding models effectively. From another point of view, many problems that arise from real practical cases are online in nature, meaning that some of the information is not available at the beginning, and they will be revealed as time goes on. In this case, the information of the stochastic parameters is not known in advance. Using Mixed Integer Linear Programming (MILP) for these kinds of problems would be expensive in terms of computations. Nevertheless, Reinforcement Learning methods perform very well for these dynamic problems as well. In addition, the majority of Combinatorial Optimization problems can be defined over graphs. Recent advances in deep learning in particular Graph Neural Networks (GNN) have led to a noticeable improvement in the application of Reinforcement Learning methods in these problems. GNNs capture the hidden relational information among all the elements (nodes and edges) of the graphs to make better predictions. In this dissertation, we take advantage of RL and GNN to develop efficient algorithms for CO problems in diverse applications.

In the first essay, we develop a deep RL method to find the best beam orientations to irradiate the patients undergoing radiation therapy for cancer treatment using the Cyberknife system. We simultaneously find the trajectory to traverse them in order to reduce the treatment time while maintaining the quality of the treatment compared to using all the possible beam orientations. Using the trained agent, we can generate patient-specific treatment plans instead of using a fixed set of beams for every patient and every cancer site. To this end, we represent the problem over a graph and apply GNNs to learn a new representation of the graph and the features of its corresponding elements. Using this method, we can use many different features concerning the anatomy of the patients and the dosimetric information. We propose a deep Q-Learning algorithm to train an agent to select a subset of the beams and the order to visit them. The results are then forwarded to a Direct Aperture Optimization (DAO) problem where other treatment parameters are calculated. The results show that using the proposed method reduces the treatment time while maintaining the quality of the treatment compared to the clinical treatment baselines.

In the second essay, a GNN-based deep RL algorithm is proposed to solve the dynamic Routing and Wavelength Assignment (RWA) problem in the context of telecommunications network optimization problems. The incoming traffic requests arrive dynamically following an unknown distribution. We model this problem as a MDP. Due to the dynamic nature of the problem, we develop a deep RL algorithm with the objective of maximizing the admitted traffic requests. The results show that the proposed algorithm outperforms the widely used heuristics (both in the research literature and in practice). By adopting this algorithm, we provide a model that solves the routing and resource (wavelength) assignment at the same time. We consider a case with the perfect information, where all the information is known to the optimizer, formulated a mathematical programming model of this problem, and solved it with commercial solvers as a baseline (upper bound) to compare the results of the algorithm.

In the third essay, we address a dynamic multi-resource constrained scheduling problem that arises from a real problem in the industry. We consider a case where each job has a specific deadline and an expiration date. We apply Q-Networks with function approximation to generate solutions that minimize the total tardiness costs and expiration costs. We first define the problem as a graph and assign several features to each node. To capture all the latent relations among elements, we apply Graph Attention Networks as a decoder to learn a new representation of the input graph at each decision time. The RL agent selects a job (from the set of arrived jobs) and assigns resources to it in order to minimize the cumulative expected penalties. We compare the results with different methods in the literature. We also formulate the problem with perfect information with Constraint Programming (CP), to achieve a lower bound for the solutions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

3D-CRT	Three dimensional conformal radiation therapy
BOO	Beam Orientation Optimization
CK	Cyberknife
CNN	Convolutional Neural Networks
CO	Combinatorial Optimization
COVERT	Cost Over Time
CP	Constraint Programming
CR	Critical Ratio
CT	Computed Tomography
CVRP	Capacitated Vehicle Routing Problem
DAO	Direct Aperture Optimization
DQN	Deep Q-Networks
DRL	Deep Reinforcement Learning
DVH	Dose-Volume Histogram
FMO	Fluence Map Optimization
GAT	Graph Attention Networks
GNN	Graph Neural Networks
IMRT	Intensity Modulated Radiation Therapy
LSTM	Long Short-Term Memory
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MILP	Mixed Integer Linear Programming
MLC	Multileaf Collimator
MRI	Magnetic Resonance Imaging
PCTSP	Prize Collecting TSP
PPO	Proximal Policy Optimization
OARs	Organs-At-Risk
OBS	Optimal Burst Switching
OP	Orienteering Problem
RL	Reinforcement Learning
RNN	Recurrent Neural Networks
RMSA	Routing Modulation and Spectrum Assignment
ReLU	Rectified Linear Units

RWA	Routing and Wavelength Assignment
SBRT	Stereotactic Body Radiation Therapy
SDN	Software-Defined Networking
SPT	Shortest Processing Time
SRS	Stereotactic Radiosurgery
TSP	Traveling Salesman Problem
VMAT	Volumetric Modulated Arc Therapy
WCOVERT	Weighted Cost Over Time
WMDD	Weighted Modified Due Date
WDM	Wavelength-Division Multiplexing
WSPT	Weighted Shortest Processing Time

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Combinatorial Optimization problems arise in different domains such as operations research, computer science, and discrete mathematics. Many problems in the fields of routing, telecommunication, scheduling, transportation, planning, and decision-making processes are classified as CO problems. Generally, in CO problems we seek to select a subset from a finite set that optimizes an objective function. Due to their discrete and non-convex nature, these problems are known to be difficult to solve from the complexity theory point of view [1]. The majority of current algorithms developed to solve these problems suffer from certain limitations when dealing with large-scale problems: expensive execution, either in terms of required infrastructure or the actual execution time, and the lack of generalization due to devising meticulous, problem-specific configurations.

Recent advances in Machine Learning and in particular in Deep Reinforcement Learning (DRL) have demonstrated noticeable improvement in the decision-making problems. These data-dependent (or experience-dependent) methods, which exploit the intrinsic patterns of the problems, have gained attention in solving CO problems. The promise is that by using these methods and inferring these patterns, we can develop more efficient algorithms that achieve high-quality solutions in a shorter amount of time for practical cases. In addition, once a model is trained, it can be generalized to many other unseen instances and solved in a short amount of time.

In addition, due to the discrete nature of CO problems, many of them can be introduced over a graph structure. The majority of these problems can be modeled as a graph, whereas for some, the variable-constraint relationships can be formulated as bipartite graphs. These graphs hold critical information on the patterns in their patterns and structures, which machine learning approaches, in particular, Graph Neural Networks can exploit [2].

In this dissertation, we apply methods from machine learning to different CO problems arising in diverse contexts which are introduced in the remaining of this chapter.

1.1 Radiation Therapy

More than 50% of the patients suffering from cancer need to be treated with radiation therapy, either as a standalone treatment mode or in combination with surgery or chemotherapy. The workflow of a patient going through radiation therapy can be classified into six main steps. In *patient consultation*, the oncologists evaluate the severity of the case and make decisions on how to proceed. The treatment scheme is determined—Intensity Modulated Radiation Therapy (IMRT), Volumetric Modulated Arc Therapy (VMAT), Cyberknife (CK), etc. Computed Tomography (CT) scans, Magnetic Resonance Imaging (MRI), and further possible required information which assist in the precise diagnosis are gathered in the *planning image acquisition* step . Using this detailed information, the oncologists begin *target and structure segmentation.* Tumors, Organs-At-Risk (OARs) and other structures are carefully detected and segmented. By meticulously detecting anatomical landmarks, oncologists and dosimetrists enter the most crucial step, the *treatment planning.* In this step, all the parameters of the radiation therapy are determined. The result of this whole process is limited by the quality and accuracy of the treatment plan devised in this step. The frequency and the length of treatment sessions are then determined. The patient attends the oncology and radiation therapy centers for *treatment delivery.* After finishing up the treatment, post-treatment patient *follow-up* steps are carried out to improve the quality for future patients and further assist the patient in this process.

Machine learning techniques may facilitate decision-making in each of the aforementioned steps. In this work, we are interested in the *treatment planning* phase, where objectives are sought after within some logical constraints. The problems in this step are categorized as large-scale and complex CO problems. The treatment planning is generally divided into two major components: Beam Orientation Optimization (BOO) and Fluence Map Optimization (FMO). In the former, several beam orientations are selected from the set of all possible beams (4π plan). The patient will be irradiated through the selected beams. If the candidate beams are selected in a hypothetical circle around the patient, a *coplanar* treatment is chosen. Otherwise, if the beams are scattered in 3D space around the patient, a *non-coplanar* treatment is followed. The intensity (fluence) of each beam is optimized in the FMO problem. This can be achieved by either modulating the beam itself emanating from the machine (IMRT or VMAT) or by spending some time with a fixed beam dose rate (CK).

Several different radiation therapy modalities exist which are presented as follows. Based on the type, location, and severity of the tumor, the oncologist recommends the most proper method. Of course, the selection is limited by the modalities available at the clinic.

Three dimensional conformal radiation therapy (3D-CRT) Using sophisticated image techniques, three-dimensional representations of tumors and other surrounding structures are generated in this technique. By using blocks, the radiation beams are shaped to embody the shape of the tumor. Nonetheless, this method cannot closely conform to the shape of the tumor [3].

IMRT A special case of 3D-CRT with more capability for conforming to the shape of tumors. The beams are discretized into smaller beamlets where each beamlet can adjust its intensity independently from others. Only a few fixed beam orientations (generally coplanar) are employed in this method. Beam is only on at these particular beam orientations and during the movements to reposition, the beam is off.

VMAT In this method, the beam rotates around the patient's body while the beam is on. Dose rate, beam shape and the speed of movement are all adjustable during the rotation as well. This results in more conforming treatment and reduces the delivery time.

Stereotactic Radiosurgery (SRS) Using smaller beam grids and allowing for more beam orientations, this method yields high quality treatment but by spending much more time during delivery. The Cyberknife system is one of the machines using this modality of treatment.

In this work, we focus on the last modality and in particular the CK system. We are interested in the case where the beams are discretized into beamlets and an extra equipment called Multileaf Collimator (MLC) is mounted on the device, which can adjust the shapes of the beam to the tumor with great extent of accuracy.

1.1.1 The Cyberknife system

The Cyberknife system (`Accuray Inc., Sunnyvale, CA`) is one the most prominent devices that delivers Stereotactic Body Radiation Therapy (SBRT). It allows maximal flexibility in beam orientations and field shaping and fewer monitor units [4]. The standard beam set of the CK system consists of nearly 200 beams which are typically non-coplanar [5]. However, using all of those nodes will not be beneficial, and may lead to long treatment times without any meaningful contribution to the quality of the treatment [6].

In the first work presented in Chapter 4, we propose a GNN-based DRL algorithm to simultaneously find the best beams to irradiate the patient, and the trajectory to follow them for Cyberknife. To this end, we construct a reward function based on different dosimetric and anatomical characteristics.

1.2 Telecommunication Networks Optimization

In recent years, the data traffic in communication networks has surged dramatically with the rapid growth of technologies (such as 5G and cloud computing) and new network service. In response to this rapid growth, it is critical to increase the automation and intelligence within telecommunication networks. Optical networks have been the major carrier in communication networks. In optical networks, the higher layers provide services to the lower level physical resources of the network [7]. Networking Problems can be categorized in three main steps. Initially, based on the requirements and estimated demand a network is designed from scratch in the *Network Planning* step. By deploying the designed network, traffic is moved across it. This traffic needs to be routed over the network and the corresponding network resources are assigned to it based on the individual requests in the *Traffic Engineering* step. Networks may be subjected to increased traffic or other problems, and an overall maintenance could be helpful in the *Network Engineering* step. The latter is an expensive intervention and a good design and more importantly a well-planned traffic engineering would be critical. In this work, we consider the second step.

All-optical networks focus on resource allocation in several dimensions of the problem, i.e., links, wavelengths, and time slots for traffic provisioning. These resource management decisions are time consuming and sophisticated for practical problems. Route planning is one of the fundamental parts of in the resource management of the all-optical networks. Aside from this, assignment of other types of resources, such as wavelengths, spectrums, and modulation format, are essential in all optical networks. In the scope of this thesis, we only consider routes and wavelengths as the resources of the optical networks.

In Wavelength division Multiplexing (WDM) networks, each fiber link has several wavelengths to transmit the data. RWA problem refers to selecting a route and assigning a wavelength (a “lightpath”) for each incoming traffic request [8]. An optical lightpath traverses multiple fiber links and optical nodes, all with the same wavelength. This is called the “wavelength continuity” constraint. RWA problem is shown to be NP-hard [9] and can be formulated as a Combinatorial Optimization problem. Many of the approaches to solve the RWA problem, either using machine learning or not, find the route and the wavelength in separate steps. In other words, for an incoming traffic connection request, a route is first selected, without consideration of the wavelengths already assigned throughout the network, and then a wavelength will be assigned to generate a lightpath. These selections are, however, substantially intertwined and the status of one greatly affects the other one. A comprehensive survey on the use of machine learning techniques in different variations of optical networks is presented in [7].

In terms of the incoming traffic connection requests, the RWA problem can be categorized into static and dynamic. In the static setting, all the information are available before the arrival of the requests. The routing and wavelength assignment decisions are determined in advance. On the other hand, lightpaths are provisioned at the arrival of the new requests as they arrive in a dynamic RWA setting. In this condition, the lightpaths usually leave the network while their corresponding service time is finished.

In the second work of this dissertation presented in Chapter 5, we first formulate the dynamic Routing and Wavelength Assignment problem as a Markov Decision Process. We then develop a Deep Q-Networks method that trains an agent with the objective of minimizing the number of blocked incoming traffic connection requests. We utilize the graph-structured input, the underlying communication network, and adopt a GNN framework to embed critical latent information to find a better policy.

1.3 Dynamic Scheduling

Scheduling is the process of planning, arranging and optimizing workflows in a production process. The main purpose of scheduling is to keep due dates of customers and minimize the cost and time of the production, meaning enhancing the productivity of the production process. Moreover, companies must be able to operate in highly uncertain environments, and satisfy the ever changing market conditions, keep production quality and maintain sustainable production. As the production systems develop from basic workshops to flexible and service-oriented productions, their complexity increases and the uncertainties around the process become critical.

Task scheduling solution approaches can be categorized into exact and approximate algorithms. Exact algorithms explore the full scope of the solution space and find the global optimal solution. This exhaustive approach is not efficient in practical, large-scale problems due to the computational complexity. In dynamic conditions, where the future orders are not known, the use of exact methods is substantially restricted due to the lack of knowledge to make decisions. In contrast, approximate scheduling methods do not explore the whole solution space. They devise various search strategies to find better solutions iteratively. These methods find feasible solutions more quickly with much lower computational complexity. However, there is no guarantee to find the global optimal solution of the scheduling problems.

Scheduling problems are characterized by three sets, namely, the jobs, the machines and the optional additional resources. Machines can be parallel, meaning they can perform the same function, or they can be dedicated, i.e., specialized in certain functions. For the parallel setting, the processing power of the machines can be equal (identical). Their processing power can also vary among them but constant for each machine (uniform) or depending on a particular task (unrelated). For dedicated tasks, there are three models of processing jobs:

flow shop, open shop, and job shop. Each job can be comprised of a subset of operations. In the open shop problem, the number of operations is the same for each job. Also each operations needs to be processed on a particular machine in a sequential order. A similar condition exists in flow shop; however, an additional precedence constraint is introduced for fulfilling operations. The difference in job shop is having an arbitrary number of operations for each job.

A scheduling problem can be made more complex by introducing resources. In this case, any task may require the use of several resources besides a machine to be processed on. Resources are divided into three categories. A resource may be renewable, meaning that its temporary availability is constrained at a given time. A non-renewable resource means that its total consumption is constrained. A resource having both previous constraints is called doubly constrained.

The third work of this dissertation presented in Chapter 6 revolves around a dynamic multi-resource constrained scheduling problem arising from a real-world problem from industry. This production system has parallel identical machines to process the input. In order to forward the production, a number of further resources are required. There exists different types of resources, and each of those are renewable. The jobs arrive in a dynamic fashion into the production system, creating an uncertain environment. We discuss a system in which each job requires some raw material. These raw materials are transferred by either a robotic arm, a liquid dispenser or a human operator. Once the job is finished, a human operator removes it from the machine. The type of these additional resources are directly correlated to the job, and it is known once the job arrives.

CHAPTER 2 LITERATURE REVIEW

This chapter provides an overview of the preliminaries and basic concepts used throughout the thesis. Section 2.1 presents a basic summary of Reinforcement Learning. Section 2.2 introduces Graph Neural Networks, an essential part of the methods developed in this dissertation. We then describe the application of Machine Learning methods in CO problems in Section 2.3. Sections 2.4, 2.5, and 2.6 provide a literature review on Radiation Therapy Treatment Planning, dynamic Resource and Wavelength Assignment Problem, and dynamic multi-resource constrained scheduling, respectively, considering the role of Machine Learning in their corresponding solution approaches.

2.1 Reinforcement Learning

Recent years have seen a surge in applying machine learning techniques in the field of CO problems [10]. Based on the nature of the CO problems, which can generally be formulated as sequential decision-making problems, RL has proven to be suitable for these computation tasks. RL is goal-oriented learning and is conceptually different from other machine learning techniques−supervised and unsupervised learning. Moreover, RL uses exploitation and exploration mechanisms, which are not available to the other approaches [11]. Several researchers studied and categorized the previous literature focusing on successfully applying RL methods for CO problems [12, 13].

A Reinforcement Learning agent interacts with the environment, takes various possible actions and observes the reward (or cost) of them, and remembers these effects within the current state of the environment. The agent has no prior expert information about the environment where it is operating in. Only gathering experience from these interactions develops an optimal policy that results in the maximum expected collected rewards at the end of its life span, i.e., the episode. The end of the episode is defined by the optimization problem to be addressed. This optimization problem can be modeled as an MDP. Finding the optimal solution of an MDP generally requires an exhaustive search, computing for all possible state-action pair combinations. An alternative to this solution approach is to use RL.

Two major approaches for RL methods are tabular and approximate solution methods. The main building blocks of any RL method are comprised of four essential elements. A policy (the action-selection strategy followed by the agent), a reward (the feedback of the environment based on the selected action), a value function (long-term collected rewards), and an optional model of the environment known to the agent. We focus on “model-free” RL in this dissertation.

The RL agent aims to find an optimal policy and by following that obtains the optimal value function. The recursive relation for the optimal value function by selecting actions at each decision point is called the *Bellman optimality equation*. A solution to this can be found through an iterative process. Watkins and Dayan [14] developed an algorithm based on this premise, called Q-learning. In the Q-learning algorithm, we create a table consisting of all state-action combinations and a value for each of those, which are updated iteratively. The Q-values reflect the cumulative expected reward at a particular state-action pair, assuming that the current policy is followed until the end of the episode. In practical problems, however, the size of this Q-table is extremely large, prohibiting the algorithm to be efficient. To address this issue, function approximation methods were introduced. Mnih et al. [15] proposed a method by integrating deep neural networks and a Q-learning algorithm, namely Deep Q-Networks (DQN). DQN uses different neural network structures to approximate the Q-values without actually exploring the same pair before during the training phase.

2.2 Graph Neural Networks

Successfully applying machine learning techniques within the CO is challenging, especially when the underlying problem involves graphs. Those problems that discuss real-world practical problems are generally large-scale and sparse in nature. The potential machine learning method then must be scalable and have the capability to handle sparsity. The majority of the advanced machine learning methods are within the *supervised learning* category. They require a large number of labeled data (for training) to devise a model that can predict the new instances with high accuracy and precision. Employing these methods within CO problems means solving multitudes of large-scale, possibly difficult CO problems for the purpose of training which contradicts the objective. In addition, supervised learning methods can only be generalized to new instances arising from the same distribution of the training data. In the context of CO problems, we need machine learning methods that have the capability to generalize beyond the training instances [2].

GNNs [16] is a family of neural networks that can handle the above issues. They can operate over graphs. The promise is that they can achieve relational reasoning and combinatorial generalization [17]. To take advantage of the machine learning models for CO problems, effectively representing graphs is critical. The key idea is to learn a new representation of the underlying graph that can be handled by “classical” machine learning techniques. There are two major techniques for extracting features to represent graphs including: feature engineering and representation learning. The former requires meticulous hand-engineered features specifically tailored for a problem. This approach is time-consuming and expensive. The latter, on the hand, learns the features automatically and requires minimal human effort [18]. Lamb et al. [19] present a high-level study of applications of GNNs in different reasoning tasks. More recent developments such as algorithmic reasoning are studied in [2]. GNNs have shown high-performance results in different domains where the data can be illustrated as an underlying graph [17, 20, 21]. Arising from the first ideas introduced by Kireev [22], many researchers studied machine learning for graph-structured data. Veličković et al. [23] and Hamilton et al. [24] examine representation learning on large scale graphs. Veličković et al. [23] proposed a novel GNN architecture, namely Graph Attention Networks (GAT), leveraging attention mechanism layers to overcome the shortcomings of the previous approaches, i.e., only considering the structural information, and not taking the importance of these information into account. Spectral approaches are discussed in the works of Monti et al. [25] and Defferrard et al. [26]. Recent advances in deep learning techniques have resulted in developments in representation learning over graphs algorithms. They greatly facilitated computations over graphs. GNNs show state-of-the-art performance on different frontiers, making this research field truly interdisciplinary [18].

The information about the nodes, edges, and connections of a graph are represented in an adjacency matrix. Nodes and edges may have further properties called node features and edge features. In the field of machine learning, one can use the graph-structured data for several purposes. Firstly, one can perform node level predictions. For example imagine we have a graph with unlabeled nodes and one simply wants to predict attributes about these nodes or classify them. GNNs will use the information on the other elements of this graph to infer on these unlabeled nodes. Another possibility is link (edge-level) prediction. As an example, one can predict whether there will be a connection between two nodes in a particular graph. To generalize, one can can use the whole graph as an input to classify it or predict a specific attribute of interest.

In the past, there has been other approaches to handle graph-structured data such as hand crafted features as the input for machine learning models. However, graph data has some interesting properties that make it difficult to work with. Feed forward neural networks typically expect a fixed size input. This brings about the first difficultly of the graph-structured data. The size and shape of a graph might change within a dataset. This particular property applies to other datasets such as images, but for these applications, one can simply resize, pad, or crop the images to the same size across the dataset. Such operations, however, are not defined over graph data. For instance, we cannot simply remove additional nodes and edges. Therefore, it is critical to use a method that can handle arbitrary input shapes.

Another feature of the graphs is called isomorphism in graph theory. It means that two graphs that look different can be structurally identical. Therefore, the algorithm that deals with graph data input needs to be permutation invariant. This is the main reason that we cannot directly use the adjacency matrix as the input of the feed forward neural networks, as it is sensitive to changes in the node order. Finally, the structure of the graphs is non-euclidean. For images for instance, there exist a clear grid that can be expressed by coordinates. Graphs, on the other hand, are dynamic structures that may lay differently in the space and the distance metrics are not clearly defined.

The fundamental idea of the GNNs is to learn a suitable representation of the graph data that can be used by neural networks, that is called representation learning. Using all the information about the graph, including the node features, the edge features, and the connections stored in the adjacency matrix, the GNNs output new representations which are also called embeddings for each of the nodes. These embeddings contain the structural as well as the feature information of the other nodes in the graph. In this manner, each node knows the properties of some other node, its connection and its context in the graph. Finally, the embeddings can be used to perform predictions for the machine learning problem that is aimed to be solved. For instance, if the objective is node level predictions, we would use the node embeddings of a specific unlabeled node to obtain a prediction. On the other hand, if we want to obtain graph level predictions, all of the node embeddings are combined in a certain way to output a new representation of the whole graph. Alternatively, pooling operations can be utilized to compress the graph into a fixed size vector. This representation can consequently be used to run a prediction. Similar nodes, meaning nodes with similar features and in similar contexts will lead to similar embeddings. In the same way, similar graphs will result in similar embeddings using GNNs. The size of the node embeddings is a hyper parameter that can differ from the initial feature size. Edge features can also be processed in the GNNs and will be combined into node embeddings.

Within the GNNs, there are several “message passing” layers that are the core building blocks of the Graph Neural Networks. These layers are responsible for combining the node and edge information as well as the structural information into the node embeddings. The message passing is carried out for each node. This is done by gathering the current information of neighbouring nodes, combining them in certain ways to get a new embedding, and updating the node features to new embeddings. This approach is also called graph convolution [27]. In graphs, we use the information in the node’s neighborhood, and combine it in a new embedding vector. In other words, the neighboring nodes of a particular node, share their embeddings with it. This step is done simultaneously and in parallel for all the nodes in the graph. This sharing mechanism is called message passing. During the learning process, many messages pass back and forth between the nodes.

The features of node u are represented by h_u. The feature update for node u is mainly performed using the two following operations: aggregate and update. Aggregate uses the features of the all direct neighbors v of a node u, that is shown $v \in \mathcal{N}(u)$, and combines them in a specific way. Then, the update operation uses the current features in layer (k) of the graph neural network, and combines it with the aggregated neighbor features. The general message passing formulation can be presented as follows:

$$h_u^{(k+1)} = UPDATE^{(k)}\left(h_u^{(k)}, AGGREGATE^{(k)}\left(h_v^{(k)}, \forall v \in \mathcal{N}(u)\right)\right) \tag{2.1}$$

This basic formula remains the same for all the different variants of the message passing graph neural networks. The difference would be in how the update and aggregate functions are performed. There are many operations, such as mean, max, neural networks, and recurrent neural networks have been used as these functions. Gated Graph Neural Networks uses a recurrent neural network to update the node features iteratively over time [28]. Kipf and Welling [29] aggregate the neighboring information as a normalized sum of the states. Additionally, they incorporate the update function into this aggregation by adding a self loop for a particular node and including it into the summations. In their approach, therefore, the update and aggregate functions are combined into one computation. Another work by Zaheer et al. [30] uses multi-layer perceptrons, also known as feed forward neural networks, to perform the aggregate operation. In this approach, there are learnable weights that can be optimized for the best aggregation of the neighboring states. Attention mechanism [31] is also applied to GNNs by Veličković et al. [32]. This means that the importance of the features of the neighbor nodes is considered for the the aggregation. As a result, the updated embedding contains more information about important neighbor features. The corresponding message passing formulation of the mentioned variants of GNN are presented in Table 2.1 Besides these, many variations of the update and aggregate functions has been proposed in the literature which are studied by Zhou et al. [33].

Table 2.1 GNN variants.

Variant	**Message passing formulations**
Gated Graph Neural Networks	$h_u^{(k)} = \mathrm{GRU}\left(h_u^{(k-1)}, m_{\mathcal{N}(u)}^{(k)}\right)$
Graph Convolutional Networks	$h_v^{(k)} = \sigma\left(\mathrm{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{h_v}{\sqrt{\lvert\mathcal{N}(u)\rvert\lvert\mathcal{N}(v)\rvert}}\right)$
Multi Layer Perceptron	$m_{\mathcal{N}(u)} = \mathrm{MLP}_\theta\left(\sum_{v \in \mathcal{N}(u)} \mathrm{MLP}_\Phi(h_v)\right)$
Graph Attention Networks	$h_u^{(k+1)} = \sigma\left(\sum_{v \in \mathcal{N}(u)} \alpha_{u,v}^{(k)} \mathrm{W} h_v^{(k)}\right)$

2.2.1 Graph Attention Networks

Instead of using using a weighted average of the features of the neighboring nodes in which the weights only depend on the number of the connections of the nodes as in the work presented in [29]. They use a particular function that calculates the weights. This function considers the embeddings of the source and the target nodes. This allows the weights to depend more than just the number of the neighbors, but instead it could include anything that the embeddings capture such as the features and the local structure. This function is called the attention function [31], shown by “a”, as it allows the node to attend to some of the neighbors more than the others:

$$h_u = \sum_{v \in \mathcal{N}(u)} a\left(h_u, h_v\right) h_v \tag{2.2}$$

A softmax function is applied to the attention function to normalize the scores, across the neighborhood.

$$\begin{align} \alpha_{u,v} &= \mathrm{softmax}_v\Big(a\left(h_u, h_v\right)\Big) \tag{2.3} \\ &= \frac{\exp\Big(a\left(h_u, h_v\right)\Big)}{\sum_{k \in \mathcal{N}(v)} \exp\Big(a\left(h_u, h_k\right)\Big)} \tag{2.4} \end{align}$$

There are various methods to compute the attention function using the embeddings of the nodes. One methods is to simply calculate the dot product of the embeddings, $a(h_u, h_v) = h_u.h_v$ for the nodes u and v. Using this, if the embeddings are well aligned, the attention function attains a larger number and if they are orthogonal it will yield the value of -1. The disadvantage of this this approach is that there are no learnable parameters in this attention mechanism which consequently limits the modeling capacity. We can thereby use neural networks that contain parameters that can be adjusted to suit our problem: $a(h_u, h_v) = \sigma\left(\mathrm{a}^T.[\mathrm{W}h_u || \mathrm{W}h_v]\right)$. W is a learnable linear projection matrix, and “a” is a learnable vector of parameters. This equation applies a linear projection on each embedding separately, concatenates the results, and takes the dot product with the direction that should be attended to, the vector “a”. A nonlinear activation function, σ is then applied to the results. Veličković et al. [32] used a LeakyReLU activation to obtain the final functional form for calculating the raw attention scores. The results are then forwarded to the softmax function to get the normalized attention scores:

$$\alpha_{u,v} = \frac{\exp\Big(\text{LeakyReLU}\big(\mathrm{a}^T.[\mathrm{W}h_u||\mathrm{W}h_v]\big)\Big)}{\sum_{k\in\mathcal{N}(v)}\exp\Big(\text{LeakyReLU}\big(\mathrm{a}^T.[\mathrm{W}h_u||\mathrm{W}h_k]\big)\Big)} \tag{2.5}$$

The final form of the Graph Attention Networks message passing equation for node u is:

$$h_u = \sigma\left(\sum_{v\in\mathcal{N}(u)} \alpha_{u,v}\mathrm{W}h_v\right) \tag{2.6}$$

The raw features of the neighbors of node u are multiplied by the projection matrix W, which is the same projection matrix used in the attention matrix. This message is scaled by the normalized attention mechanism, $\alpha_{u,v}$. The summation of the scaled messages of all the neighbors of node u is then passed through a nonlinear activation function to get the final updated embedding for node u.

Figure 2.1 provides an example of the message passing layers. Consider a graph with five nodes, numbered from 1 to 5. The message passing is carried out for all five nodes simultaneously. For illustration purpose, we focus on node 1 in the following, which is shown in yellow. To update the features in node 1 from layer (k) to $(k+1)$, we collect information of its direct neighbors, nodes 2, 3, and 4, which means we perform the message passing. What we have at this stage is the information of the current node features, and the information about the neighbor node features. We then perform an aggregation over the neighbor nodes to combine their corresponding information. Finally, the current node features and the aggregated information are put together to generate the updated features of node 1 in layer $(k+1)$. After doing this for all the nodes, we obtain a new embedding for every node of the graph. As it can be observed, after doing message passing once, the node 5 only has the information of node 4 (blue features) and the information of its own (green features). At this time step, this node has no information about the yellow node. After several message passing layers in the GNN structure, we obtain the final updated features for each node. At the end, every single node of the graph, knows something about all other nodes, and the structure of the graph. This is stored in each of the node embeddings. Eventually, we can use these final embeddings to perform predictions as they contain all the information about the graph that is required. In other words, we learn these embeddings by iteratively combining the node information in the local neighborhoods. By iteratively, we mean we first learn about the direct neighbors, then the neighbors of neighbors and so forth.

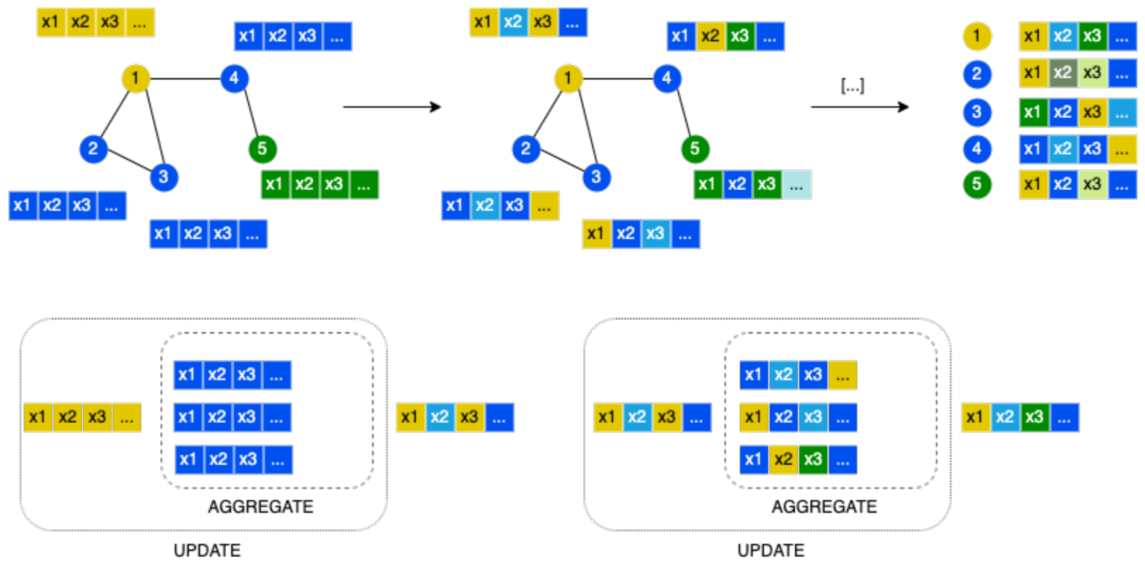


Figure 2.1 Message Passing layers example.

2.3 Machine learning applications in Combinatorial Optimization

Many researchers generated standalone heuristics that do not use a linear, integer or constraint programming solver to help a machine learning method satisfy the constraints of a CO problem. Vinyals et al. [34] proposed a sequence-to-sequence learning method, "pointer network", to find a tour for the Traveling Salesman Problem (TSP). In their work, given a random sequence of cities, they produce a feasible solution using a neural network architecture. Each node (city) is defined as a vector and a Long Short-Term Memory (LSTM) encoder generates a new representation of each node. A decoder, based on another LSTM, using the pointer mechanism (attention-based mechanism) outputs a probability distribution over the nodes to be selected as the next city in the tour. The algorithm works in an iterative fashion to build complete tours and delivers a permutation of the input sequence as a final solution. They, however, trained this network with supervised learning. Therefore, many instances of TSPs were required to be solved by conventional methods to generate labels for the training data. The model's performance is also determined by the quality of the training data in supervised learning. To overcome these issues, Bello et al. [35] substituted the supervised learning with an RL agent within the same neural network architecture. This has been the first work integrating Deep Reinforcement Learning in the context of Combinatorial Optimization problems. Considering an instance of a TSP as a sequence fails to address the critical information on the underlying graph of the problem, the missing part of the aforementioned studies. Dai et al. [36] integrated GNNs for the first time in optimization problems to address this issue. Using GNNs allowed the algorithm to capture critical, mostly hidden information among elements of the underlying graph. Unlike the sequence-to-sequence learning proposed before, they used a Structure2Vec architecture [37]. They also explored applying the same structures to other classical graph optimization problems such as Maximum Cut and Minimum Vertex Cover. Their extended architecture was applied to the Set Covering Problem as well. Several variations of routing problems such as the Traveling Salesman Problem, Capacitated Vehicle Routing Problem (CVRP), Orienteering Problem (OP), Prize Collecting TSP (PCTSP) were tackled by Kool et al. [38]. They present an encoder-decoder architecture, based on Graph Attention Networks. They trained this proposed neural network using Policy Gradient methods, in the Actor-Critic RL approach. Deudon et al. [39] replaced the Recurrent Neural Networks (RNN) encoder with a transformer architecture [31] (using multi-head attention). The solution generated by the RL is further improved by a simple yet effective 2-opt heuristic. They show combining learning methods with heuristics can achieve a solution closer to the optimal solution. Similar to [38], the input to the decoder in their architecture is the new representation of the input graph following a GAT. Nazari et al. [40] also proposed similar frameworks to solve the static and dynamic versions of the TSP and CVRP. François et al. [41] demonstrate that the solutions developed by the mentioned approaches can be used as the first solution of a local search algorithm for the TSP, resulting in a more efficient algorithm. An extensive survey on the use of Graph Neural Networks in Combinatorial Optimization is presented in [2].

2.4 Radiation therapy treatment planning

Bahr et al. [42] proposed the very first linear programming model for radiation therapy treatment planning. To follow that, many researchers applied different techniques from operations research to the field of medical physics. As described in Chapter 1, treatment planning problems can be decomposed into two main modules: Beam Orientation Optimization and Fluence Map Optimization. The purpose of BOO is to find a subset of all possible beams such that the treatment goals are satisfied [43, 44]. The main challenge of BOO arises from the fact that the effect of the selected subset of beams can only be truly viewed based on the quality of dose distribution, after solving the FMO problem. Considering this, two major approaches exist in the literature. First, to solve BOO and FMO as a single CO problem, with one objective function. Following this, the best beam orientations that contribute to the best dose distribution towards the patient in the final treatment are selected [45]. However, this approach results in a large-scale optimization problem prohibiting to reach of optimal solutions for practical instances in oncology centers [46]. Different methods based on column generation [47] and benders decomposition [44] are proposed for this approach. Pugachev et al. [48], Li [49] present a metaheuristic method (Simulated Annealing) to resolve this issue. In contrast to the first approach and to address the computation complexity, BOO and FMO can be solved sequentially. The proposed methods following this approach exploit some information about the beams prior to solving for the optimized beams [50, 51]. Individual beam scores are generated based on different criteria and the subset of the beams are generated by gathering the ones with the highest scores.

The planning and delivery of radiation therapy treatment is a complex task, but can be facilitated considering the recent advances in artificial intelligence and especially deep learning [52]. In recent years, machine learning and especially deep learning methods have attracted studies in all of the steps of the radiation therapy workflow. A low number of previous research projects exist that revolve around the “treatment planning” step, compared to the multitudes of articles applying deep learning techniques to image segmentation or computer-aided cancer detection (*target and structure segmentation* step).

Radiation treatment planning aims to determine the optimal parameters of treatment delivery. This planning is carried out by using dedicated software developed by manufacturers, which are mainly driven by users. Several methods using Machine Learning have been developed to aid treatment planning. The promise is to reduce planning time and increase the quality of the treatment. They mainly use a knowledge-based pool of previously delivered plans; and the (semi) automatic planning finds the most similar instances and uses their parameters [53, 52]. The majority of the previous use of machine learning in treatment planning is dedicated to dose prediction using a database of previously treated patients. Nguyen et al. [54] apply a U-net Architecture with additional Convolutional Neural Networks (CNN) layers to predict the dose delivered to each voxel from the structures of a patient.

Amit et al. [55] developed a random forest regression algorithm to map anatomical features of the patients to beam scores. An optimization model is then built to select the subset of beam orientations. They show that the dose distribution of the automatic procedure is comparable in terms of target volume coverage, and organ sparing is superior to plans produced with a fixed set of common beams. Obtaining data in the field of radiation therapy is difficult and the supervised learning they used requires multitudes of labeled training data, not accessible to all. Dong et al. [56] presents a novel trajectory selection using Monte Carlo Tree Search (MCTS). They opt for the Maximum Upper Confidence Bounds to intelligently select the trajectory with consideration of geometric and physical constraints. They demonstrated similar target volume coverage while sparing OARs better as compared to the coplanar plans.

Similar to VMAT, arc therapy has been studied for Cyberknife. Smyth et al. [57, 58] define a cost map for the robotic arm and find the trajectory achieving the minimum cost using individual beam scores. A similar algorithm is devised by Kearney et al. [59] to produce arc trajectories for Cyberknife. Bedford et al. [5] propose an evolutionary algorithm to select beam orientations. They show that a plan with 15 beams delivers a treatment quality comparable with the one with 110 beams in approximately half time. Despite the higher complexity of the trajectory optimization for Cyberknife compared to IMRT and VMAT, due to the extensive flexibility of the trajectories, very few researchers studied this problem.

2.5 Routing and Wavelengths Assignment

There is a trend to use machine learning-based routing algorithms in all-optical networks in recent years. Some studies use a pool of previously solved problems and obtain route generation rules by employing supervised learning methods [60]. On the other hand, routing can be viewed as a decision-making task for which RL methods have proven to be beneficial [61].

Troia et al. [60] develops a classification method for the routing problem in Software-Defined Networking (SDN) optical networks. Prior to training, they create several optimal RWA configurations using MILP and existing heuristics. Their algorithm captures the incoming traffic matrices and classifies each request to one of the pre-computed routing solutions. This results in an efficient algorithm that attains solutions quickly. However, the quality of the solution is bound by the number of existing classes in the classification task.

Kiran et al. [62] propose an Reinforcement Learning method that finds alternative routing for Optimal Burst Switching (OBS). They model this problem as a Multi-armed bandit problem and use a tabular Q-learning method that selects the next link at each node for the next hop of the burst. Kiran et al. [63] proposed a multi-agent approach for this problem where each agent considers the actions taken by the others as well. They discuss that, within the OBS context, single-agent RL methods may result in sub-optimal solutions. They provide separate algorithms for routing and wavelength assignment, each based on a multi-armed bandit problem. Some researchers consider other types of resources in the WDM networks. Chen et al. [64] applied a DRL algorithm for an Routing Modulation and Spectrum Assignment (RMSA) problem in elastic optical networks. They opted for policy gradient methods, Advantage actor-critic, to select the best resources for the incoming request (route, modality, and spectrum). Different techniques can improve the performance of a DRL solution approach. Xu et al. [65] make use of experience replay buffer to address the traffic engineering in communication networks. They show a reduction in the delay for incoming requests while increasing the throughput compared to widely-used baseline methods. Several authors study different variations of the RWA problem. Networks with physical impairment are studied in [66]. Similar to the majority of the published articles, the authors separate routing and wavelength assignment sub-problems in their algorithm, and opt for the first-fit heuristic for wavelength assignment. Recent advances in GNN have captured the attention of researchers in communication networks as well. The underlying network topology of the communication network problems can easily be translated into graph-structured input. Rusek et al. [67] and Swaminathan et al. [68] developed GNN-based routing algorithms with the objective of minimizing the request admittance delay.

2.6 Dynamic Scheduling

The dynamic (or online) scheduling problem discusses the case where jobs arrive in the system in an online manner, during a time horizon. This is in contrast with the case where all the information about all the incoming jobs is known in advance, at the time of making decisions. The dynamic scheduling problem can be considered as a dynamic decision making approach. The orders arrive at random times in the system, and, different resources requirement and availability change over time. The scheduling agent needs to determine the assignment of which resource, based on the requirement of the eligible tasks and the availability of the different types of resources.

One of the critical issues in dynamic scheduling is the existence of uncertainty at decision points. DRL methods are well-suited for addressing this issue. Mahadevan and Theocharous [69] demonstrated the superior capability of RL methods in production systems. In recent years and with remarkable success with DRL [70], these methods are transferred to a plethora of applications such as production systems.

Even static scheduling problems are usually NP-hard or NP-complete problems [71]. Therefore, it is difficult to find an efficient exact algorithm to solve practical problems. Most of the researchers in the field of scheduling develop metaheuristics and tailor them for their problems. The majority of these studies model static scheduling, and rarely consider resource constraints [72]. These methods fail to meet the requirements of the highly flexible and dynamic environment in complex practical scheduling problems. Therefore, for multi-resource constrained scheduling problems, heuristics are used. The performance of the schedules is thereby limited by the quality of the heuristic devised for the particular problem, and once the environments change slightly, those heuristics need to be readjusted [73].

Similar to other Combinatorial Optimization problems, machine learning and in particular

Deep Reinforcement Learning methods can help address these issues within the scheduling problem context. Luo et al. [72] discuss how scheduling problems can be formulated as an MDP and sequential decisions can be made at the arrival of new jobs in a dynamic fashion. The complexity of DRL is less than the complexity meta-heuristic algorithms. More importantly, DRL methods have generalization potential. Once the model is trained, it can be adapted to modified environments (different number of machines and resources, different kinds of production processes). Aydin and Öztemel [74] propose a method based on Reinforcement Learning for the dynamic job shop scheduling problem. They develop a modified Q-learning agent to select the best “priority rule” among the available ones. They still make use of priority lists for scheduling; however, selecting the next one is carried out using a variant of the tabular Q-learning method. Therefore, the scheduling is mainly based on previous knowledge rather than the exploration capability of the RL agent. Wang and Usher [75] present a Q-learning algorithm for a production dispatching problem with a single machine. A solution approach based on RL for unrelated parallel machine problems with machine constraints is proposed by Zhang et al. [76]. They demonstrat the effectiveness of their algorithm against the widely-used baselines such as weighted shortest processing time, weighted modified due date, and weighted cost over time.

A few researchers adopted Graph Neural Networks during the training phase of DRL agents for job shop scheduling problems. They employ a Proximal Policy Optimization (PPO) [77]. Their results achieve a lower makespan compared to different heuristic rules. Nevertheless, resource constraints are not discussed in their problem. This issue is addressed by Luo et al. [72]. They develop a PPO for the multi-constrained dynamic workshop scheduling problem. They use numerical and graphical features to encode the workshop’s state. A fully connected and a multi-layer CNN are used to extract latent features from the numerical and the graphical features, respectively. Each job has only one operation in their work, and each machine has a buffer for the waiting jobs.

CHAPTER 3 GENERAL ORGANIZATION OF THE DOCUMENT

As presented in Section 2.3, recently many researchers have adopted methods from Machine Learning to solve Combinatorial Optimization problems. This trend has been more prevalent with the advances in deep learning. Since CO problems can be generally represented as a graph, applying Graph Neural Networks has greatly facilitated the learning mechanism over graph-structured data. The promise is to enable algorithms that utilize the latent relationship between graph elements that can be critical to finding better solutions.

The literature on the Cyberknife system treatment planning in Section 2.4 as well as in-person discussion with physicians and medical physicists show that the lengthy treatment is a major drawback of this radiation treatment modality. Besides, for all the patients, only a few predetermined gantry trajectories are currently used, which may not be ideal. This long treatment plan can adversely affect the treatment quality due to the inadvertent movements of the patients during the treatment and also entails discomfort for many patients. From another point of view, because of this treatment time, fewer patients can be admitted at a given time window, and the wait times (which can be dangerous for patients' health) can be large. Considering this drawback, in Chapter 4, we propose an efficient algorithm that simultaneously finds the beam orientations and the trajectory to traverse them such that the treatment quality would be similar to the clinical treatment. This method incorporates advantages from Deep Reinforcement Learning.

The literature on Routing and Wavelength Assignment in Section 2.5 reveals that most of the previous studies focus on the routing part of the problem and resort to heuristics for the wavelength (resource) assignment part. In addition, in practice, the incoming traffic arrivals (and releases) happen online, in a dynamic manner rather than having all the information at the beginning. In Chapter 5, we utilized the potential of reinforcement learning to solve this problem. As networks can be represented as graphs, we employ Graph Neural Networks for the representation learning to generate inputs for the neural networks to make decisions on routes and wavelengths at the same time.

In Section 2.6 and based on the requirements of the producer, it can be observed that most of the solution methods devised for the Scheduling problem are suitable for the static case. This means situations where all the information regarding the arrival times and processing times of the jobs are known in advance in a deterministic manner. In the problem we consider, the distribution of arrival times and processing times of the jobs are not known to the scheduling agent. The jobs arrive in a dynamic fashion. We propose an efficient algorithm in Chapter 6 to output a solution for a variant of the dynamic multi-resource constrained scheduling problem. We employ GNNs and Deep Q-Networks to train a scheduling agent which generalizes well to problems that do not participate in the training phase.

In Chapter 7, we provide a general discussion on the proposed methods as a whole. Finally, in Chapter 8, we summarize the contributions of the dissertation, explain the limitations of the proposed approaches, and offer future research directions.

CHAPTER 4 ARTICLE 1: DEEP Q-LEARNING FOR SIMULTANEOUS BEAM ORIENTATION AND TRAJECTORY OPTIMIZATION FOR CYBERKNIFE

Peyman Kafaei

Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, Canada

Quentin Cappart

Department of Computer Engineering and Software Engineering, Polytechnique Montréal, Montréal, Canada

Marc-André Renaud

Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, Canada

Nicolas Chapados

Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, Canada

Louis-Martin Rousseau

Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, Canada

Abstract Objective. Despite the high-quality treatment, the long treatment time of the Cyberknife system is believed to be a drawback. The high flexibility of its robotic arm requires meticulous path-finding algorithms to deliver the prescribed dose in the shortest time. Approach. We proposed a Deep Q-learning based on Graph Neural Networks to find the subset of the beams and the order to traverse them. A complex reward function is defined to minimize the distance covered by the robotic arm while avoiding the selection of close beams. Individual beam scores are also generated based on their effect on the beam intensity and are incorporated in the reward function. Main results. The performance of the presented method is evaluated on three clinical cases suffering from lung cancer. Applying this approach leads to an average of 35% reduction in the treatment time, while delivering the prescribed dose provided by the physicians. Significance. Shorter treatment times results in a better treatment experience for individual patients, reduces discomfort and the side effects of inadvertent movements for them. Additionally, it creates the opportunity to treat a higher number of patients in a given time period at the radiation therapy centers.

Keywords deep reinforcement learning, Cyberknife, beam orientation optimization, radiation therapy, trajectory.

History This article is published in *Physics in Medicine and Biology*, Vol. 66, No. 21, 215002.

4.1 Introduction

A high-quality radiation therapy treatment requires delivering a prescribed dose to the target volume while sparing normal tissues and OARs. The deposited dose needs to closely follow the clinical prescriptions without having a large dose gradient within the tumor. To this end, selection of the best possible set of beams through which the patient is treated with radiation is considerably important.

The problem targeted in this work is to find the best beam orientations for the Cyberknife radiosurgery system (`Accuray Inc., Sunnyvale, CA`) with a MLC. This system delivers noncoplanar beams towards the patient and is used for SRS or SBRT. While the Cyberknife system can deliver high-quality treatment plans in terms of dose conformity, the long delivery times of certain Cyberknife plans have been cited as a risk factor [59]. It should be pointed out that more than half of the overall treatment time corresponds to the robotic arm movement between beams and not the beam-on time (actual treatment) during which inadvertent patient movements diminish the treatment quality. The total treatment time might take about one hour [78]. Finding the optimal beam trajectory is challenging as it necessitates considering the total treatment time as well as the final dose distributions.

Non-coplanar plans delivered on standard linear accelerators such asVMAT or IMRT tend to have reasonable treatment times because the treatment field is larger, and there are not as many degrees of freedom as on the Cyberknife. However, the high flexibility of the Cyberknife system requires the consideration of treatment time as a critical factor through meticulously selecting beams and optimizing the trajectory.

In this paper, we propose to use DRL to optimize beam orientations for the treatment plans delivered with the Cyberknife system. This method has the benefit to exploit different geometric and dosimetric features to pick the best beams and to simultaneously minimize the delivery time. This directly enables us to consider numerous patient-specific features for beam selection, unlike the common practice that uses a fixed trajectory [5].

The contributions of this paper are as follows:

1. We propose a Deep Reinforcement Learning algorithm, namely deep Q-learning, to optimize the beam orientations. Experimental results obtained show that this approach can obtain a high-quality solution in a shorter amount of time compared to a standard mixed-integer programming formulation. The final treatment quality is also on par with the treatment using all the possible beams.

2. Our proposed solving process leverages both dosimetric and geometric features at the

same time. This directly leads to a realistic objective compared with individual beam score methods. In addition, we consider another measure to distribute the selected beams around the patient and avoid having clusters of beams in specific positions around the patient.

4.2 Related Works

Treatment Planning problems can be represented by two main subproblems: BOO problem and FMO problem. The former finds the optimal subset of the beams to treat the patient through them. The latter computes the amount of the dose delivered at each beam in different organs.

Two major approaches exist to select the most favorable subset of beams. The first approach optimizes the beam orientations and beam fluence maps simultaneously. The beams having positive intensities (positive fluence) in the solution are thus selected in the trajectory. This approach leads to an NP-hard CO problem, meaning there are no known algorithm able to solve it in polynomial time [79]. Although this results in an optimal solution concerning the prescribed dose objectives [45], it entails a major drawback: The computation time may be prohibitive [46]. A partial solution to these issues is to find an approximate solution, instead of the optimal one. This can be attained using local-search methods, such as simulated annealing [48], or by controlling the execution time of mixed integer linear programming models [80]. In addition, some researchers propose methods such as Benders Decomposition [44] or Column Generation [47] to solve the mixed integer mathematical programming formulation of this problem.

A second option is to decouple optimizing beam orientation and beam fluence maps and solve them sequentially. Following this strategy, researchers can exploit some information about the beams before the optimization of the beam intensities. To solve the beam orientation problem, various quality measures of an individual or subsets of beams can be generated [50, 51]. This reduces the computation time despite some approximation errors. The selected beam orientations are then forwarded to another optimization problem to find their respective fluence [81]. Numerous scores based on geometric features of organs and dosimetric characteristics of the candidate beams are defined. Beams are then ranked and selected based on their score. The focus of the methods considering geometric measures is based on the premise that avoiding OARs is essential for an acceptable plan [48, 50]. The overlap between the volumes of the target and the OARs from every beam's-eye-view gives a such metric [82] as well as the position of the OARs with respect to the target value (background or foreground) due to the physical characteristics of the photon beams [82]. The latter assumption neglects the cases where the target volume is surrounded by the OARs as in lung tumors.

Ranking beam orientations based on dosimetric effects have also been considered. Bangert and Oelfke [81] generate locally ideal beam orientations for each target voxel using radio-biological features and conclude that beam orientations typically cluster around distinct positions. They use k-means clustering to select k beams from the potential locally ideal set of beams. In addition to this metric, Yuan et al. [51] define a similarity score for pairwise beam orientations to select more scattered beams iteratively, starting from an empty set of selected beams.

Several researchers have incorporated beam FMO problems into the scalar beam scores to develop more advanced measures of quality. Bangert and Unkelbach [46] have shown that an early stopping mechanism before reaching the optimal solution of this problem is a surrogate for the optimal solution and can thus be used as individual beam metrics. In another study, after selecting beam orientations using solely geometric scores, Smyth et al. [58] run the beam fluence optimization for a few iterations. They then perturb the couch angle for each selected beam and evaluate the FMO to find the best objective function corresponding to the best beams. Starting from every possible beam orientation around the patient, referred to as the 4π plan [83], Langhans et al. [84] solve the FMO problem and eliminate the beams with the total dose of less than the average and repeat this process until $\mathcal{N} = 20$ beams are selected. By assigning different geometric scores to every selected beam orientation, they solve a path-finding algorithm with a variation of A^* algorithm and output a trajectory to be traversed during the treatment delivery. Nonetheless, one disadvantage is that the angular separation of the selected beams is such that the delivery would not be feasible due to machine restrictions. Lyu et al. [85] follow a similar approach and activate a subset of dosimetrically promising beams from an initial 4π plan. They assign FMO-based individual and pairwise beam costs to find the optimal trajectory.

In this paper, we apply DRL to solve the BOO problem. We consider a number of measures in order to optimize beam selection and thereby reduce the treatment time. Even though we decouple BOO and FMO problems, our proposed method uses measures from both geometric features of the beams and dosimetric features related to each patient. Therefore, dose information is taken into account without further complicating the problem. The subset of beams selected by our approach is then forwarded to the DAO module [86] to obtain the intensities of the beams and delivery sequence.

4.3 Materials and Methods

4.3.1 The Cyberknife system

The extra degrees of freedom in the Cyberknife system enable a non-coplanar delivery with limited restrictions. The robotic arm can deliver radiation at a discrete set of positions around the patient. Such positions are referred to as *nodes*. In this work, a node and a *beam* are used interchangeably as we assume that there is only one beam orientation per node. These nodes are uniformly distributed on a sphere centered at the imaging center of the Cyberknife system. Starting from the resting point, the robotic arm can move from the current node to any other node that does not entail a collision with the patient's body. The Cyberknife equipped with a MLC allows more flexibility in field shaping and delivers fewer monitor units in comparison with cone collimators [5]. The InCise MLC mounted on the Cyberknife system has 26 leaf pairs, each with a width of 3.85 mm and a maximum field size of 115 $mm \times 100$ mm at 800 mm source-to-axis distance. The radiation field from each beam is discretized into beamlets with the size of twice the MLC leaf width by 5 mm [5] and a dose distribution is generated for each beamlet.

This work assumes a step-and-shoot delivery [87]. Delivery of the radiation is only allowed at nodes. Unlike IMRT and VMAT, the dose rate of the Cyberknife system is constant during delivery and cannot be increased or decreased as a parameter of the device; therefore, the dose deposited in the patient is proportional to the delivery time.

4.3.2 Deep Reinforcement Learning

Reinforcement learning is a sub-field of machine learning that considers the decision-making dynamics of an agent that interacts with an environment. The goal is to find an optimal sequence of actions that the agent must follow in order to accomplish a given task. Reinforcement learning problems are commonly modeled as a MDP [11].

Let $\langle S, A, T, R \rangle$ be the tuple representing the agent-environment interactions in the proposed RL structure. S is the set of states and A denotes the set of all possible actions. By taking an action the state of the environment changes to a new one following the transition function T. The environment sends a signal to the agent by a deterministic reward function R as a result of selected action. At time step t, the agent receives a representation of the state of the environment $s_t \in S$. Using the current information at s_t, the agent takes an action $a_t \in A$ which gives the state-action pair (s_t, a_t). At the next time step, the state of the environment is updated to $s_{t+1} = T(s_t, a_t)$ and the agent receives a scalar reward of $r_t = R(s_t, a_t) \in \mathbb{R}$. In a deterministic environment, the selection of an arbitrary action in a specific state always results in the same next state. The goal of reinforcement learning is to learn the agent's behaviour policy $\pi : S \rightarrow A$, indicating the action to be taken at each state, such that it optimizes the sum of the rewards. The agent visits a sequence of states $s_t \in S$ at each time step $t \in [1, \Theta]$ creating an *episode.* State s_Θ is referred to as the *terminal state.* The expected return after time step t is $G_t = \sum_{k=t+1}^{\Theta} R(s_k, a_k)$. The problem is to find the optimal policy $\pi^* = \text{argmax}_\pi \, Q^\pi(a, s) \; \forall s \in S, \; \forall a \in A$ where $Q^\pi(a, s)$ denotes the quality of selecting action a at state s under policy π. It is referred to as the Q-values, and, for this environment, we have $Q^\pi(a, s) = G_t$.

For relatively small environments, the optimal Q-values can be computed by an exhaustive exploration with Dynamic Programming. However, when the problem size increases, Dynamic Programming suffers from the so-called *curse of dimensionality* [88], making an exhaustive exploration not tractable anymore. In order to deal with this issue, Q-learning [14] provides an estimation of the optimal Q-value function for every action selected at specific states. It is computed by successive updates. This value is updated by $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_t(s_t, a_t) + \max_{a \in A_t} Q(s_{t+1}, a) - Q(s_t, a_t)\right]$ where α is the learning rate. The aim is thus to find the optimal policy such that the expected total reward gained through successive states is maximized.

In practice, the environments are comprised of an exponential number of states and many of those may not be visited during the previous updates. Neural fitted Q-learning [89] deals with this issue and uses neural networks to approximate the Q-value function. By learning a weight vector $\mathbf{w}$, the model provides an estimator such that $\widehat{Q}(s, a, \mathbf{w}) \approx Q(s, a)$. To this end, an optimizer, such as Adam [90] is used to minimize the squared loss between the current Q-value and the approximated Q-value and update the weight vector: $\mathbf{w} \leftarrow \mathbf{w} - \frac{1}{2}\alpha \nabla L(\mathbf{w})$ where the squared loss is $L(\mathbf{w}) = \left[R(s_t, a_t) + \max_{a \in A_t} \widehat{Q}(s_{t+1}, a, \mathbf{w}) - \widehat{Q}(s_t, a_t, \mathbf{w})\right]^2$. We use prioritized experience replay [91] to stabilize the training.

4.3.3 Graph Neural Networks

The Beam Orientation Optimization problem can be formulated as a CO problem. Recently, GNNs [23] emerged as a machine learning architecture to help solve CO problems efficiently [2]. For any problem that can be represented as a graph, the idea of the GNNs is to compute a vectorial representation of each node by aggregating features of the neighboring nodes [2]. The learned vector representation encodes crucial and latent structures that help to solve challenging CO problems. We formulate the BOO problem as a graph to exploit the benefits of GNNs. As shown in Figure 4.1, a state graph represents the partial trajectory with the selected beams depicted by full circles. Features are assigned to each node (represented by boxes with colors with different shades). The state graph passes through several layers in the GNNs and at each layer, a new representation of the node features is generated by exploiting the information of the neighbour edges and nodes. At the end, the final node features of the last layer are combined to create a vectorized representation of the state graph.

4.3.4 Problem Representation

An instance of the BOO problem can be represented by a simple, undirected, and fully-connected graph. Let $G = (V, E)$ be a graph representing a problem instance, where the set of vertices V corresponds to the set of all possible nodes (beams) around the patient for the Cyberknife system. The edges denote the direct path the robotic arm needs to traverse from one node to another. To fully represent the problem, we add distance-driven and dose-driven features to edges and nodes of the graph, respectively. At each time step, using Graph Neural Networks [36], we obtain an vectorial representation of the current state. This vector is then forwarded to the next steps to learn the Q-values.

4.3.5 Reinforcement Learning environment for BOO

The RL environment formalizes the problem we want to solve using the aforementioned learning algorithm. Let us consider an initial set of nodes around the patient. Due to the flexibility of the Cyberknife system, the robotic arm can move to an arbitrary node in the following step to irradiate the patient. The movements prone to collision of the robotic arm with the patient can be manually excluded by setting the cost of traversing their path to an arbitrary large value. As such, without loss of generality, collisions are not considered in our implementation. Therefore, at each time step, the set of available nodes includes the ones that have not been selected yet. An episode starts when the robotic arm is at the resting point, and at each time step, it moves towards the next node. The episode finishes when a predefined number of beams ($\mathcal{N}$) are selected [5, 84]. Each movement incurs a reward. Following the structure described by Cappart et al. [92], our RL environment is formally defined as follows:

State At each time t, the state $s_t \in S$ contains the ordered sequence of selected beams denoted by δ_t. A state can be represented as an undirected acyclic graph. The nodes of this graph include following features: (1) the nodes coordinates (x, y, z), (2) the average dose deposited in the OARs, d_{oar} and, (3) the average dose deposited in the target volume d_{tar} at unit intensity. For every structure (OARs, target volumes) that each beam passes through, we have computed the dose deposited in each voxel at unit

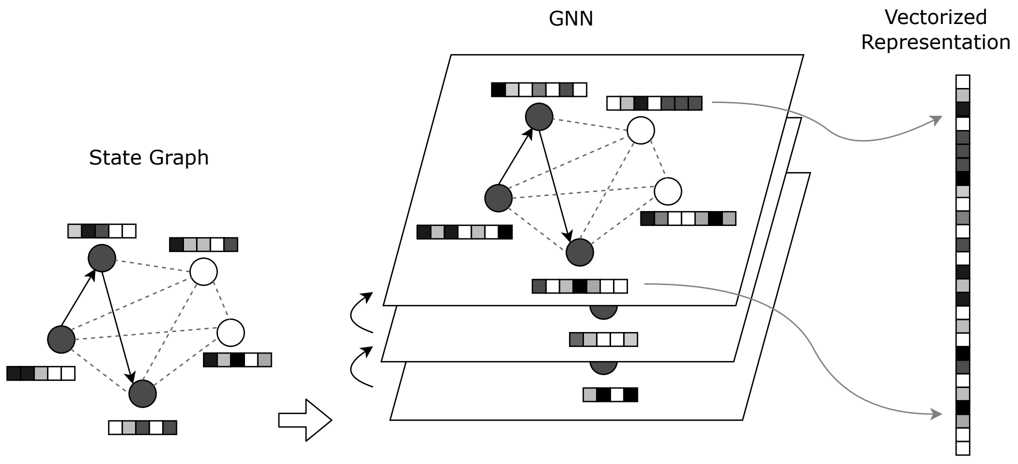


Figure 4.1 Vectorized representation of the state graph by applying GNNs.

intensity. We then average this value over the voxels for each individual structure to obtain the average dose deposited in each OAR (d_i, $i \in OARs$) and the target volumes (d_{tar}) separately corresponding to each beam. Each state is transformed into a d-dimensional vector of features ($d = 128$) using GNNs, which serves as the input of a fully connected neural network.

Action At each state s_t, the action $a_t \in A_t$ is defined as the next node to be visited. An action is available only if it is not already included in the trajectory (i.e., $a_t \notin \delta_t$).

Transition The state is updated according to the action performed. By selecting a new action, a new beam is appended to the sequence of the already selected beams. Therefore, $\delta_{t+1} = \delta_t \cup \{a_t\}$.

Reward At each time step t, the agent receives the reward $r_t = R(s_t, a_t)$. Let m be the last beam added to the trajectory in state s_t and the next beam selected is $a_t = n$. The reward is defined as follows:

$$R(s_t, a_t) = -(r_{dist} + r_{dose}), \quad t \neq \Theta \tag{4.1}$$

$$R(s_\Theta, a_t) = -(r_{dist} + r_{dose} + r_{spread})$$

r_{dist} is the euclidean distance between m and n. r_{dose} for beam n is defined as the ratio $\Big(\sum_{i \in OARs} \omega_i \; d_i\Big)\Big(\omega_{tar} \; d_{tar}\Big)^{-1}$ where ω_j is the importance of each structure j. For simplicity we have considered a similar weight for all the structures. r_{spread} is the beam-spread score. The latter accounts for the beam-spread measure among n and all previously selected beams in the trajectory denoted by $\sum_{i,j} K\Big(1 - \cos\alpha_{ij}\Big)^{-1}$ and

$i, j \in \delta_\Theta$. This ensures a maximum separation between selected nodes to produce a plan with higher quality [51, 43].

The importance of each part of the reward can be determined by weighting them; however, in our experiments, we set an equal weight on all terms except for the beam separation which is set by the choice of the parameter K.

4.3.6 Learning Algorithm

The learning algorithm relies on DQN presented in Algorithm 1. An agent is used to learn the weight vector ($\mathbf{w}$) of a fully connected neural network to output Q-values. At each iteration of the training phase, a random instance of the problem represented as a graph G is created. Each instance resembles a hypothetical patient. It must be noted that we generate the features to create the state graph. Therefore, for each instance, we generate random coordinates for the nodes from which the beams are delivered towards the patients. These values are normalized to be in the range of $(0, 1)$. The other important measure to shape random instances during training is the ratio of the dose delivered to OARs to the dose delivered in the target volume for each beam. To compute this, for each beam, we need the information on the dose deposited in every OAR ($d_i,\ i \in OARs$) and the target volume (d_{tar}) that the beam passes through. Prior to the training, for all the real patient data we calculated these values as follows. For every structure (OARs, target volumes) that each beam passes through, we compute the dose deposited in each voxel at unit intensity. We then average these values over the voxels of each OAR and the target volume to obtain the values of $d_i,\ i \in \text{OARs}$ and d_{tar}. We then add white Gaussian noise with zero mean and square coefficient of variation equal to 0.25 to augment the dataset of dose to OARs and dose to target. Following this, new instances has been added to the dataset for the training purpose [93]. Then during the training phase, the values of $d_i,\ i \in \text{OARs}$ and d_{tar} are randomly selected from this augmented dataset. We thereby compute the ratio of $\Big(\sum_{i \in \text{OARs}} d_i\Big)\Big(d_{tar}\Big)^{-1}$ for each possible beam. An episode is then constructed by selecting one beam at each time step t using the model's deep neural network architecture with the current weight vector. To make the training more robust, we exploit Prioritized Experience Replay [91]. At each time step, an experience tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ and its corresponding significance are added to the memory for further use in the training.

Always greedily selecting the action with the maximum immediate reward generally leads to inferior solutions due to the lack of exploration. Even following the action with the best approximated Q-value could lead to local minima if no exploration is used. To ensure a balanced exploitation-exploration trade-off, actions are chosen following a softmax action selection strategy. This is also superior to the ϵ-greedy strategy where it chooses equally among all available actions while exploring [92]. In the softmax action selection strategy, the greedy action still has the highest probability while others are weighted according to their value estimates. By adjusting a temperature hyper-parameter, the learning begins with higher exploration followed by increasingly favoring exploitation. At the beginning of the training, the temperature is set to 0, and it increases to a predefined maximum value as the training goes on.

Gradient-based optimizers tend to have difficulties if the rewards are large, sparse, or too small. To avoid such cases, we use scaling to map the reward space into an interval close to zero [92]. Let $\gamma \in \mathbb{R}$ be the scaling factor which is dependant on the value of the coordinates of the nodes. The rescaled reward at time step t is $\gamma\, r_t$.

We sample a batch from the stored experiences to learn the model for estimating optimal Q-values for every state-action pair. For each experience comprising the batch, we obtain an embedding (vectorized representation) of the state as illustrated in Figure 4.1 and then pass the embedding into a neural network called the *policy network* as an input, shown in Figure 4.1. The aim of the policy network is to approximate the optimal policy by finding the optimal Q-values for state-action pairs. We have such a network associated with each possible action from the input given state. The output of these networks is the estimated Q-value for each available action from the state s_t. It must be pointed out the output layer of this neural network presents the action space of the proposed RL algorithm. Each output unit (action) corresponds to a node of the state graph (a beam). The beams that are currently included in the partial trajectory are masked, illustrated by full circles in Figure 4.2, and cannot be chosen. Once the next beam is selected, the trajectory will be updated the state transitions to a new one as in Figure 4.2 (b).

At this point, the loss needs to be calculated by comparing the Q-value estimated from the policy network for the action a_t of the experience tuple in the batch and the corresponding (target) Q-value for the same action denoted by $q^*(s_t, a_t) = E\Big[R_{t+1} + \max_{a'} q^*(s_{t+1}, a')\Big]$ and $a' \in A_{t+1}$. The $\max_{a'} q^*(s_{t+1}, a')$ needs to be approximated. The target Q-values are obtained from a completely separate network cloned from the policy network named the target network with its weights are frozen with the original policy network's weights. The weights of the target network are updated to the policy network's new weights every certain amount of the time steps. This target network enables us to estimate the maximum Q-value for the next state s_{t+1} in the experience tuple to get the target Q-value $q^*(s_t, a_t)$.

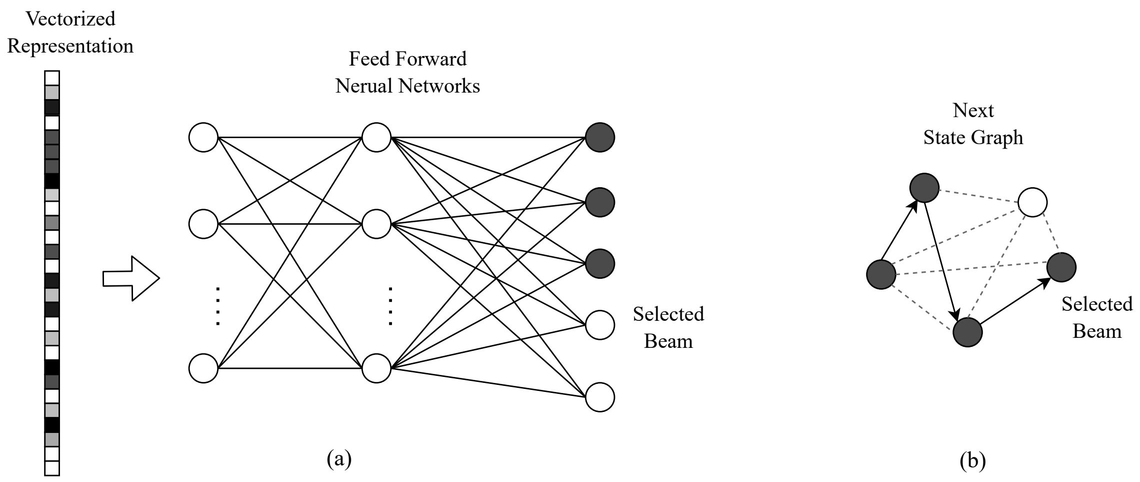


Figure 4.2 Action Selection. (a) Passing the embedding of the state through a Feed Forward Neural Network. (b) Updated trajectory after the action (selected beam) is performed. The state graph is transitioned into a new state.

4.3.7 Neural Network Architecture

Selecting the beams to incorporate in the final trajectory for the robotic arm movement of the Cyberknife system depends on the nodes of the underlying graph G. Therefore, to capture all the node and edge features, we employ a Graph Attention Network architecture to embed the graphs [32]. The policy network is a fully connected feed-forward network with three hidden layers, consisting of 128, 64, and 32 hidden units, respectively. The input of this network is the embedded current state. At each stage, we have one such network for every available action. The output of the policy network is the prediction of the Q-value for the current state-action pair. We use the *Rectified Linear Units* (ReLu) as the activation function of the hidden layers and no activation at the output layer.

Algorithm 1 Training

```
1: $\mathcal{I}$ is the number of iterations
2: $\Theta$ is the length of the episode
3: $N$ is the mini-batch size
4: $e_t$ is the experience tuple $(s_t, a_t, r_t, s_{t+1})$
5: $\alpha$ is the learning rate
6:
7: $\mathcal{M} \leftarrow$ initializeMemory($m$)        ▷ Creating the replay memory with size $m$
8: for $i$ from 1 to $\mathcal{I}$ do
9:     $G \leftarrow$ generateRandomInstance
10:    $(S, A, T, R) \leftarrow$ initializeEnvironment($G$)
11:    $s_1 \leftarrow$ initializeState($G$)
12:    for $t$ from 1 to $\Theta$ do
13:        $a_t \leftarrow$ softmaxSelection($s_t$)
14:        $r_t = \gamma\, R(s_t, a_t)$
15:        $s_{t+1} = T(s_t, a_t)$
16:        $\mathcal{M} \leftarrow$ updateMemory($e_t$)
17:        for $j$ from 1 to $N$ do
18:            $e \leftarrow$ getSampleFrom($\mathcal{M}$)
19:            $\mathcal{L}_j(\mathbf{w}) \leftarrow$ squared loss of $e$
20:        end for
21:        $\mathbf{w} \leftarrow \mathbf{w} + \frac{\alpha}{2N} \sum_{j=1}^{N} \mathcal{L}_j(\mathbf{w})$        ▷ Update the weight vector
22:    end for
23: end for
24: return $\mathbf{w}$
```

4.3.8 Solving Algorithm

Once trained, the model can be reused to obtain a trajectory for unseen instances. First, the instance is represented as a graph, and the relevant features are extracted. The features of each state s are inputted to the deep neural networks architecture of the model to estimate the value of the state-action function $\widehat{Q}$ for all feasible actions. The next node is selected following the greedy policy $\pi = \text{argmax}_{a \in A}\, \widehat{Q}(s, a)$. The node incurring the maximum Q-value is inserted into the list of selected nodes until a predetermined number of nodes are selected.

4.4 Experiments

4.4.1 Dataset

To evaluate the performance and efficiency of the proposed algorithm, we consider three challenging cases suffering from lung cancer. The data is collected from anonymous patient data who underwent radiation therapy using the Cyberknife system at *Centre Hospitalier de l'Université de Montréal* (CHUM). For the lung tumor, a choice of a few predetermined paths developed by the manufacturer is selected regardless of the innate differences among various patients. The first case consists of 28,800 beamlets and 6,241,184 voxels where the tumor is in the right lung. There are 26,208 beamlets and 7,736,670 voxels for the second case for which the tumor lies in the left lung. The third plan has 3,440,112 voxels and 29,952 beamlets, and the tumor lies in the right lung.

To this end, we evaluated the results of the proposed method with the current clinical treatment plan for the lung tumor including 100, 91, and 104 non-coplanar 800 source-to-axis distance (SAD) nodes configured by the manufacturer scattered around the centroid of the target volume for patient 1, 2, and 3 respectively.

4.4.2 Setup

Our model is implemented in Python 3.7. Training is carried out on one GPU (NVIDIA V100 Volta, 32GB memory) for 24 hours on randomly generated instances and Adam optimizer is used for training. To create an instance in the training phase, at any node, values for different features are selected from the augmented database. Therefore, we can compute the rewards namely distance-driven, dose-driven, and beam-spread measure. It should be pointed out that all of these randomly generated values are normalized to have similar ranges for all of the features during the learning.

Before initiating the training phase, we generate a set of 100 held-out validation instances to track the performance of the learned model and the baselines as the training goes on. The model resulting in the best average reward on the held-out validation set is then selected as the final one and tested on another set of randomly generated graphs with the same configuration as the training set to evaluate the generalization ability of the model.

Three baselines are developed to compare the performance of our proposed algorithm. They select a subset of beams and the best trajectory in a single pass. Firstly, for each instance in the validation set, we randomly create 50 trajectories and output the the best, the worst, and the average objective values. In addition, a greedy heuristic is developed as follows. Starting from the resting point of the robotic arm, at each time step, we select n nearest points as the candidates to be selected for the next node to visit. For each potential next node and the partial trajectory created so far, we compute the dose-driven and the beam-spread score and add the node yielding the maximum total reward to the tour. We also implemented and solved the problem (with the same instances used in the RL and the heuristic) in Gurobi 9.0.0 [94] with time limits. The mathematical programming model is represented in the appendices. Restricting the time to converge prevents finding the optimal solution using the exact solver and results in some negligible optimality gap. Nonetheless, for practical cases, the optimizer takes a long time to converge, which limits clinical applicability.

Once the subset of beams is selected, we follow the algorithm represented by Renaud et al. [86] for only photon particles. This algorithm is implemented in C++ and the quadratic mathematical programming is solved by IPOPT 3.13 [1] which is based on interior-point methods. This outputs the weight of the every selected beam and how to deliver the treatment.

4.4.3 Training

We initialize the parameters of the policy network (weight vector $\mathbf{w}$) according to *Xavier initialization* [95] sampled uniformly from $(-x, x)$ where $x = g\left[6\,/\,(dim_{in} + dim_{out})\right]^{1/2}$ where dim_{in} and dim_{out} refer to the input and output dimensions of the current layer and $g = \sqrt{2}$ is the scaling factor used in this work. The capacity of the replay memory is set to 50 and at each epoch, a batch of 32 experiences are sampled from it. We train for 24 hours on the training data generated randomly on-the-fly as described in Section 4.4.2. The learning rate of $\alpha = 4 \times 10^{-4}$ is set following different experiments.

Using the weight vector $\mathbf{w}$ learned in the training phase, at each time step the next node to be visited (next action) based on the current state s_t is selected by a greedy policy following $a_t = \text{argmax}_{a \in A_t} \hat{Q}(s_t, a, \mathbf{w})$. This process continues until the end of the episode where the predetermined number of beams are selected. With the way that the reward function is defined, the selected beams will not be clustered around some points in the space around the patient.

4.4.4 Results

In this section, we evaluate the performance of the proposed algorithm based on the quality and efficiency of the treatment on the lung cases. We use the Dose-Volume Histogram (DVH) to evaluate the quality of the plans, a method widely used in practice. These histograms

[1]https://coin-or.github.io/Ipopt/

depict the percentage of the organs capturing a certain amount of dose. In order to attain an acceptable treatment plan regarding DVH constraints, we change the structures underdose and overdose weights during the optimization of the dose intensities. We increase the penalty for the structures whose DVH measures are worse than the critical dosimetry references or the proposed values by the physicians. A challenging issue is the voxels that could be identified in more than one structure. In this case, based on the important factor of individual structures, the aforementioned voxels are assigned to the structures with higher priority. All the structures are considered throughout the algorithm, to create individual beam scores and beamlets for the beam intensity optimization. It should be pointed out that fewer multi-structure voxels results in higher quality plans. The figures and tables representing the results for patient 2 and 3 are shown in the appendices.

Figures 4.3, A.1, and A.2 compare the DVH diagrams for the cases of $\mathcal{K} = 25$ beams (DQN-Plan) and the clinical plan which uses all the beams (Full-Plan). Healthy tissues are protected in both plans but the dose to the tumor is relatively lower in DQN-Plan. All treatment plans meet clinical needs. As for the healthy tissues for patient 1, the right and left lungs, and the ribs receive lower radiation in high percentage of their volume using the DQN-plan. The heart and trachea attain lower dose in high volumes using the proposed algorithm. In all these cases, the dose deposited in the structures are within the acceptable thresholds.

The heart, right lung, esophagus receive lower dose throughout the structure with the DQN-methods compared to the Full-Plan for patient 2. For the high percentage of the volumes of the ribs and the left lung (where the tumor is in), the DQN-Plan gets lower dose deposited. The dose deposited in tumor however is less in the majority of the volume when following the DQN-Plan.

In addition, the robotic arm traverses only 25 beams in the DQN-plan whereas the Full-Plan requires all the beams in the original treatment plan, 100, 93, and 104 for patients 1, 2, and 3. In general, the DQN-Plan maintains the quality of the Full-Plan treatment within the clinically accepted thresholds, while delivered in shorter time. Table 4.1 compares the total reward (objective), execution time, and the total distance traversed by the robotic arm, and time for different methods. It should be pointed out that the value of the objective function and the execution time are irrelevant for the clinical method which is currently used at CHUM. The arm travels the distance while the beam is off. The majority of the treatment time is spent on moving from one beam to the next one. The total reward (the objective) is comprised of three parts, total distance traversed, total dose-driven score collected, and the level of the sparsity of the selected beams around the patient. Therefore, merely having the lowest distance covered will not necessarily lead to a more advantageous treatment.

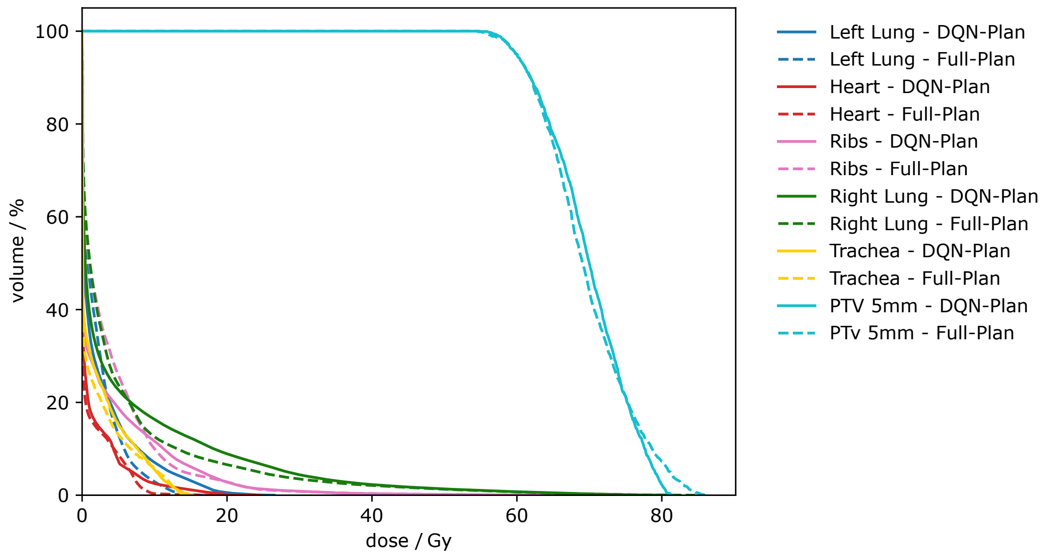


Figure 4.3 Comparison between DVH diagrams for patient 1.

Table 4.1 Solution comparisons between different methods of solving the BOO.

Patients	Method	Obj.	Execution time (s)	Total arm Distance	Time (min)
Patient 1	DQN	3.53	1.09	5,529	35
	Gurobi	3.27	3600.00	6,350	37
	Heuristic	4.50	2.47	5,494	35
	Random	4.65	0.15	2,696	31
	Clinical	NA[a]	NA[a]	17,726	54
Patient 2	DQN	4.18	1.08	4,656	35
	Gurobi	3.18	3600.00	4,836	36
	Heuristic	4.26	2.89	4,128	34
	Random	4.88	0.24	6,730	39
	Clinical	NA[a]	NA[a]	12,936	51
Patient 3	DQN	1.80	1.38	4,068	30
	Gurobi	1.53	3600.00	5,851	31
	Heuristic	4.26	1.26	3,696	29
	Random	2.41	0.09	8,166	35
	Clinical	NA[a]	NA[a]	22,553	50

As illustrated, although DQN-Plan does not yield the minimum distance traversed in comparison with the heuristic and random selection, it attains a better total reward and hence a higher treatment quality shown in Tables 4.2, A.1, and A.2. In these tables, $D_{X\%}$ represents the amount of dose deposited in at least $X\%$ of the structure, D_{max} is the maximum dose delivered to the structure, and D_{mean} is the mean dose to the structure. V_Y Gy percentage of the structure volume receiving Y Gy dose.

Table 4.1 also demonstrates the treatment time comparison between the clinical plan and the DQN-Plan. Using the DQN-Plan results in a 35%, 33%, and 40% reduction of the treatment time for patients 1, 2, and 3, respectively.

We represent the performance of the proposed Deep Q-learning algorithm during the training phase in Figure 4.4 for both patients. At certain points in time (every 100 episodes), we apply the learned model against the instances in the validation set and show it in the plot. The same results are also illustrated for each baseline (random selection, heuristic, and the results of the implementation in Gurobi). Figure 4.5 illustrates a trajectory generated by the DQN-method compared with one of the Clinical method.

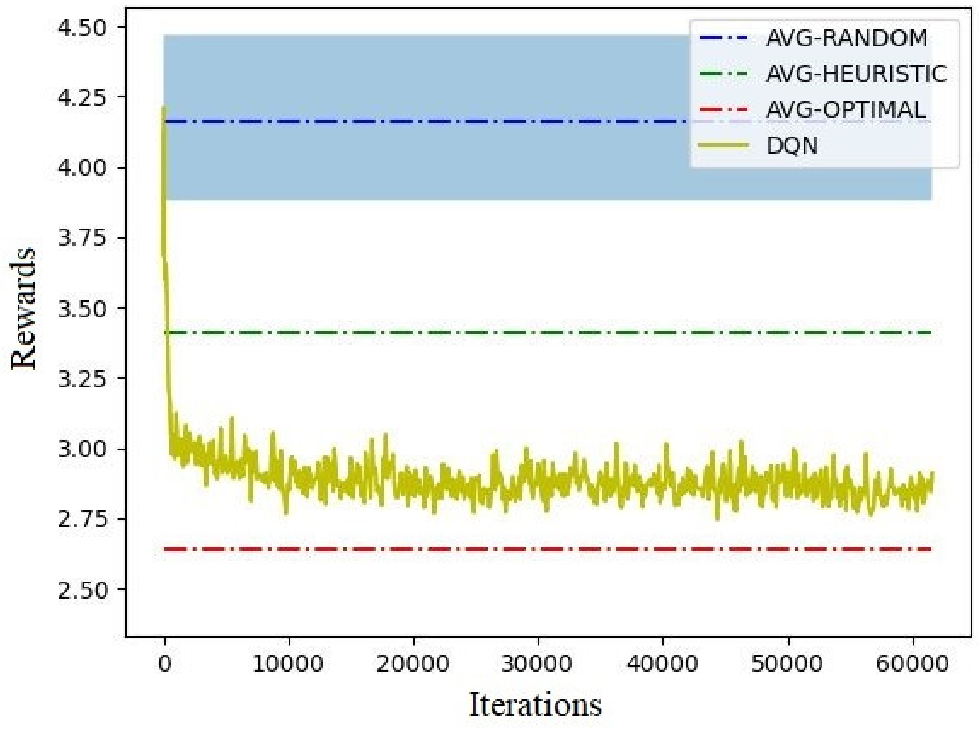


Figure 4.4 Total reward of the algorithm against the fixed validation set during training. The shaded area depicts the spectrum between the worst and the best random trajectories.

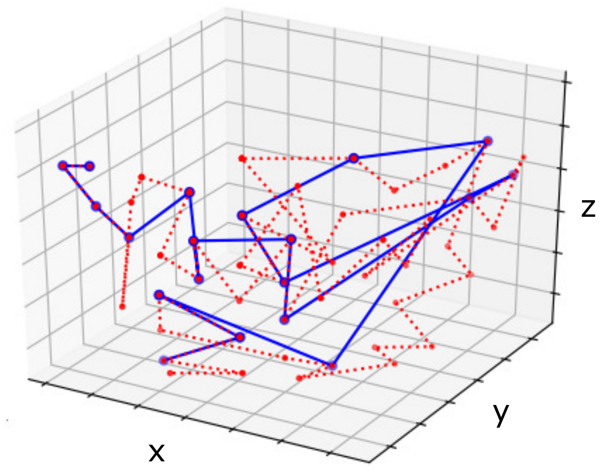


Figure 4.5 Comparison of the trajectories of the DQN-method (blue) and the Clinical-method (red).

4.4.5 Discussion

In this work, we proposed a method based on Deep Q-learning to solve the BOO problem for the Cyberknife system. It entails a reduction in treatment time while maintaining plan quality. The selected subset of the beams can be forwarded to any formulation of the Fluence Map Optimization or Direct Aperture Optimization problem (with different objective functions and constraints) to realize beam intensities and leaf sequencing.

Compared to previous methods, using a deep Q-learning approach enables us to include dose considerations related to each patient as well as the geometric characteristics of beams. The proposed algorithm exploits the benefits of both approaches of solving the BOO problem. Firstly, capturing the dose-related scores results in more realistic treatment planning. Methods assigning scores to beams by geometric characteristics generally suffer from the lack of dose-relevant rewards as they do not directly capture the information of the deposited dose of individual beams. However, we showed that including the dose-driven part in the rewards compensates this issue. Furthermore, similar to heuristics, using the learned models on new patients takes a short amount of time. However, the majority of the heuristics in the literature use only anatomical features of the patients.

As mentioned before, long treatment times are a fundamental drawback of the Cyberknife system treatment planning. We reduced the overall treatment time by considering travel times in the distance-driven part in reward computations of the proposed algorithm.

Finally, imposing spatial dispersion as the terminal reward of the episodes helps to choose the most spread subset of beams while minimizing the total rewards gained. As mentioned before, BOO can be modeled as a combinatorial optimization problem which is proven to be NP-hard. Adding these measures to this problem will increase the complexity of this problem. However, deep Q-learning method allows to consider multiple measures as different (immediate and terminal) rewards. Therefore, deep Q-learning is an efficient and effective method to tackle the BOO problem. This approach results in treatment plans with a substantially shorter treatment time compared to current clinical practice using all the beams. Although it takes a relatively long time for the model to be trained (as is the case for all learning tasks), once the model is trained, finding the solution to new instances in the inference part takes up to few seconds compared to the hours of computational time by solvers such as Gurobi. For training, we only generate random values for node coordinates and doses deposited to different organs through the beams corresponding at each node. Therefore, we do not restrain the learning for a particular case.

Another challenge with the current clinical methods is the use of a set of fixed trajectories devised by the manufacturer for all patients with the same tumors. Neglecting individual-specific variations is likely to diminish the quality of the plan, where even small changes can greatly influence the health of the patients.

The treatment plan resulting from the beams generated by the DQN-plan has acceptable dose quality. For the first patient, the maximum dose, average dose, and median dose deposited in the target volume is 4.5 Gy lower, 0.2 Gy larger and 0.8 Gy larger respectively compared to the Full-Plan. The maximum dose is reduced by 4.5 Gy in DQN-plan compared to the full-plan. While the mean dose and the median are increased with negligible values of 0.2 Gy and 0.8 Gy respectively. As for the right lung where the tumor is located, the maximum dose and the median dose are reduced by 3.3 Gy and 0.6 Gy while the average dose is 0.3 Gy higher in DQN-plan.

For the second patient, maximum dose delivered to tumor is 5.34 Gy less in the DQN-Plan. The right lung, heart and esophagus are also much better protected, considering the maximum and mean dose in DQN-Plan. The maximum dose deposited in the right lung is 12.56 Gy lower than the Full plan, while the mean dose is if reduced by around 80%. maximum dose attained by the heart and esophagus are down by 4.33 Gy and 4.59 Gy respectively. As for the left lung, where the tumor is located at, a small reduction of 5.32 Gy is obtained. On the other hand, although the maximum dose delivered to the ribs is almost 1.5 times greater using the DQN-Plan, the average dose is on par with the Full-Plan.

For the last patient, the maximum dose is about 9 Gy less in the tumor in the DQN-Plan. The maximum dose deposited in the left lung is also 20 Gy lower while the max and mean dose in the other organs are similar in both plans. Therefore, the DQN-Plan at maintains the

Table 4.2 Dose-volume parameters for the target volume and critical structures for treatment plans generated by different method for patient 1.

Plan	**Structure**		**Statistics**	
DQN-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.0	58.1	81.3
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	81.1	5.1	8.9
	Left Lung	26.4	2.2	0.5
	Ribs	73.9	**2.9**	2.8
	Hearts	24.4	1.1	0.2
Gurobi-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.0	58.2	**81.2**
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	**80.5**	5.2	7.8
	Left Lung	26.6	2.3	0.4
	Ribs	75.8	3.2	**2.2**
	Hearts	20.5	1.2	0.0
Heuristic-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.0	57.7	85.7
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	85.5	5.0	7.4
	Left Lung	33.0	2.3	1.7
	Ribs	79.3	3.1	2.8
	Hearts	**19.3**	**0.5**	0.0
Random-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.0	56.3	97.5
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	94.7	5.2	9.9
	Left Lung	58.0	2.2	4.3
	Ribs	79.1	3.4	6.5
	Hearts	62.7	0.8	1.5
Full-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.0	**56.3**	85.8
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	84.5	**4.8**	**6.5**
	Left Lung	**17.3**	**2.0**	**0.0**
	Ribs	**72.4**	3.6	2.8
	Hearts	20.5	0.9	**0.0**

dose-volume parameters of the clinical full-Plan. The dose-wash images for different patients comparing DQN method and the clinical full-plan is illustrated in Figure A.3.

The values corroborate the fact that intelligently selecting a subset of nodes results in relatively the same treatment quality. It should be remarked that these similar treatments are obtained by reducing the robotic arm movement time. This reduction of treatment time yields higher comfort for the patients, eliminates the burden on the clinics and allows more patients to be treated in any given time duration.

4.5 Conclusion

In this work, we have proposed a Deep Q-learning algorithm for the Beam Orientation Optimization problem for the Cyberknife system treatment planning. This algorithm generates a set of favorable beams which is tailored to consider patient-specific dosimetric features and beam-related geometric features. This approach also tries to maximally distribute the selected beams around the patient. The proposed Deep Q-learning algorithm has the following advantages over the other methods that can be found in the literature:

- The majority of the previous methods consider only a single feature (geometric or dosimetric) for the individual beam scores. Using this method, we can integrate any number of features into the neural network structure to generate a favorable trajectory.
- By intelligently selecting the beams, we attained treatment plans with much shorter times while maintaining the treatment quality of using all the possible beams.

Finally, we have evaluated our proposed algorithm on three challenging lung cancer cases to demonstrate the effectiveness and efficiency of the algorithm. While the training may take a long time, it should be remarked that the solution time is really short.

Although we have considered identical importance for different features in generating rewards at each step of the Deep Q-learning algorithm, one might tune the weights to achieve an even better treatment quality. For instance, having higher weight on the beam-spread score would lead to an even more scattered beam formation. In addition, as the training is not restricted to a particular patient or cancer type, it would be beneficial to evaluate the possibility of applying the same model to patients suffering from the different types of cancers.

Acknowledgment The Authors would like to thank Dr. Stephane Bedwani and the Department of radiation oncology, Centre hospitalier de l'Universite de Montreal (CHUM), who helped us by preparing and providing the case data and for their illuminating insights. This work has been approved by CHUM research ethics committee with the approval number CER 20.244.

CHAPTER 5 ARTICLE 2: DYNAMIC ROUTING AND WAVELENGTH ASSIGNMENT WITH REINFORCEMENT LEARNING

Peyman Kafaei

Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, Canada

Quentin Cappart

Department of Computer Engineering and Software Engineering, Polytechnique Montréal, Montréal, Canada

Louis-Martin Rousseau

Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, Canada

Hamed Pouya

Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada

Nicolas Chapados

Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, Canada

Abstract With the rapid developments in communication systems, and considering the dynamic nature of them, all-optical-networks are becoming increasingly complex. This study proposes a novel method based on Deep Reinforcement Learning for the routing and wavelength assignment problem in all-optical wavelength-decision-multiplexing networks. We consider dynamic incoming requests, in which their arrival and holding time are not known in advance. The objective is to devise a strategy that minimizes the number of rejected packages due to the lack of resources in the long term. We employ Graph Neural Networks to capture crucial latent information from the graph-structured input to develop the optimal strategy. The proposed Deep Q-Networks algorithm selects a route and a wavelength simultaneously for each incoming traffic connection as they arrive. The results demonstrate that the learned agent outperforms the methods widely used in practice and can be generalized on network topologies that did not participate in training.

Keywords deep reinforcement learning, Routing and wavelengths assignment, Network optimization, Graph neural networks.

History This work is submitted to *Informs Journal of Optimization.* (July 5, 2022)

5.1 Introduction

All-optical-Networks are considered to play a critical role in wide-area backbone networks [96]. In this paper, we investigate the performance of novel learning algorithms on the Routing and Wavelength Assignment problem [97] in the context of Wavelength division Multiplexing networks. Solving RWA corresponds to selecting a route between two end-points of all incoming requests and assigning a wavelength to transmit the data across them. In WDM networks, packages that use the same fiber links cannot use the same wavelengths at the same time. Due to the limited number of wavelengths, incoming traffic requests may be blocked if there are no available wavelengths across the links of the selected route.

RWA problems are conventionally categorized into static and dynamic. The static version is based on the assumption that all the information of the incoming requests such as arrival time, and duration (in case of releasing the requests after a finite time) are known. A set of lightpaths (combination of a route and a wavelength) are set up all at once, and they remain in the network. However, in realistic applications, these assumptions do not hold. A lightpath needs to be generated for each incoming request as it arrives. The admitted traffic connection request may be released after a finite amount of time.

The dynamic arrival and departure of the traffic requests and the uncertainty of the future ones substantially destabilize the problem and reduce the accuracy and efficiency of the conventional optimization methods. In general, the lightpaths already assigned cannot be re-routed to accommodate the new requests that arrive. However, aggressive reconfiguration i.e., lightpath defragmentation has recently been proposed to modify the current lightpaths to reduce the blockage probabilities at the expense of high computational and operational costs [64, 98, 99].

In the settings of this work, wavelengths are assigned to each of the links comprising a route for an incoming request. If all of these links use the same wavelength along the route of the request, the information can transfer from source to destination. Otherwise, the incoming request is blocked. This is known as the wavelength continuity constraint, which makes wavelength-routed networks different from the traditional circuit-switched telephone networks.

Although recently researchers have started applying methods from artificial intelligence in the field of RWA problems, only routing has been under investigation. The wavelength assignment and routing are substantially intertwined and selecting one affects the other. In this paper, an algorithm based on Deep Reinforcement Learning is presented for the dynamic RWA problem. This algorithm selects the route and wavelengths simultaneously.

The algorithm is entirely online and trains via simulation and past experiences, and thereby does not require large data sets. We assume that precise network state information is available at any point in time. The specific contributions are as follows:

1. We propose a Deep Reinforcement Learning algorithm to find the routes and assign wavelengths to incoming traffic requests. Experimental results obtained show that this approach can yield a high-quality solution against standard baselines used to solve this problem for four of the most famous topologies.

2. We exploit the topology of the underlying network as a graph-structured data and apply Graph Neural Networks to encode latent information. This enables us to capture more features and leads to a better performance of the proposed algorithm.

3. The DRL agent proposed in this work has the capability to select routes and assign wavelengths for the dynamic RWA problem simultaneously. This outperforms the conventional methods that mostly focus on the routing part of the problem, for incoming connection requests.

The rest of the paper is organized as follows. Section 5.2 examines the existing literature on the use of Graph Neural Networks and Deep Reinforcement Learning in Routing and Wavelength Assignment as well as the previous algorithms to solve the RWA problem. Section 5.3 discusses the deep reinforcement learning algorithm to solve this problem. The RL environment and the proposed neural networks architecture are introduced in this Section. Experimental results, discussions, and conclusions are presented in Section 5.4.

5.2 Background

In this Section we study previous methods developed to solve the RWA problem.

5.2.1 Routing and Wavelength Assignment

Route planning and resource management are two fundamental tasks in optical networks. This problem is shown to be NP-hard for a set of traffic requests [100]. As the size and complexity of the network increase, opting for conventional Shortest Path First routing algorithms may lead to a high number of blocked requests. In addition to routing, assigning wavelengths to routes results in a more complex problem. RWA is an NP-complete problem [7]. It may be formulated as an Mixed Integer Linear Programming. This approach, however, attains solutions at the expense of complex, expensive, and time-consuming computations even for medium-sized instances [93]. The majority of the literature on RWA has focused on heuristic solutions, such as Genetic Algorithms and Simulated Annealing [101]. These methods are faster but attain sub-optimal solutions [99, 93]. Most of the conventional, exact algorithms are proposed only to find a route, while the heuristics are capable of finding several routes more readily [102]. A taxonomy of the RWA problems and their corresponding information is provided in [103]. Jaumard and Daryalal [104] propose methods to enable solving RWA instances with realistic sizes. To this end, they present an exact method based on column generation decomposition as well as two heuristics for the RWA problem. The authors outline that the algorithm may fail for large instances in a reasonable solution time; however, it provides ϵ-solutions with high accuracy. Daryalal and Bodur [99] formulated the RWA problem as a stochastic optimization problem where uncertainty exists for the future traffic. They studied two perspectives, namely, while granting the connection requests via the RWA problem, and during the defragmentation process by lightpath rerouting. In their study, the distributions of the random variables are known to the decision maker.

In contrast to routing solution approaches, a number of researchers study wavelength assignment techniques. Randhawa and Sohal [105] show that that by increasing the load of the network, the blocking probability increases. Zhou and Yuan [106] compared different wavelength assignment strategies, random-fit, first-fit, and most-used with the effect of imprecise global network information. In their study, they opted for the widest shortest path algorithm for the routing part of the incoming requests.

While several researchers have studied dynamic RWA problems, the majority of them focus on finding optimal routing while the requests arrive in real-time, and only opted for a fixed heuristic for the wavelength assignment [107, 108, 109]. Some also consider traffic grooming by the presence of wavelength conversions throughout the network, to reduce the blocking probability [110]. A dynamic algorithm that considers the length of the k-shortest-paths and free wavelengths is developed by combining the weighted least-congestion routing and first-fit wavelength assignment strategy [111]. It is shown that plugging wavelength converters over the network does not highly contribute to the reduction of the blockage rate as time goes on [112, 113].

Recently, different Machine Learning techniques have been applied to solve this complex problem. In Martin et al. [93], the authors presented supervised learning solution methods based on logistic regression and deep neural networks. They formulated the RWA problem as a classification problem. Several optimal RWA configurations are computed prior to training the model using MILPs and heuristics in the literature. Their method learns relations between features of a problem, such as the topology of the network, capacity, and wavelengths, with one of the corresponding optimal configurations. Although this approach attains solutions quickly, it only considers a few possible configurations for all diverse network configuration inputs. Several researchers leverage Reinforcement Learning to solve the RWA problem. Kiran et al. [62] formulate the path and wavelength selection in Optical Burst Switched (OBS) networks as a multi-armed bandit problem and solve it with a tabular Q-learning method. They provide separate algorithms for path and wavelength selection. The corresponding Q-value of path selection and wavelength selection are updated based on a successful burst in the network. The selection of the route and wavelength are not directly related in their proposed method.

Applying DRL to network optimization problems has received high interest in recent years. Chen et al. [64] propose a DRL framework for a Routing, Modulation, and Spectrum Assignment (RMSA) in Elastical Optical Networks. They deploy advantage actor-critic methods to parameterize the action selection policy, and show a reduction in request blockage following their proposed algorithm. In [65], the authors utilize experience replay to enhance the performance of a proposed Q-learning method to solve the traffic engineering problem in communication networks. Pointurier and Heidari [66] solve the routing problem in Optical Networks with physical impairment by developing a Linear Reward ϵ-penalty method. They disaggregate the wavelength assignment from the routing and opt for a first fitted heuristic. The experimental results show a lower blocking rate in comparison with the shortest path and uniform path selection strategies. By proposing a novel elaborate representation of the states in a DRL algorithm, Suárez-Varela et al. [114] show an increase in the performance of applying DRL to the routing problem in Optical Transport Networks, which outperforms traditional heuristics. A few studies use Graph Neural Networks as a part of the RL approach to solve variants of the RWA problem. Swaminathan et al. [68] developed a GNN-driven RL algorithm, GraphNet, for optimal routing of the incoming requests on Software-Defined Networking to minimize the package delay. Unlike WDM, they do not consider the existence of multiple wavelengths over the fiber links of the network. In order to predict the distribution of per request delay in SDNs, Rusek et al. [67] proposed a novel network-based on message-passing GNNs using Recurrent Neural Networks as aggregation functions. Similar to other algorithms devised for SDN, RL-routing uses RL for routing packages with the objective of throughput and delay of the incoming traffic. Recent comprehensive survey on machine learning methods for optical networks and SDN are presented in [7] and [115].

As discussed, most of the previous studies develop novel methods for the *routing* part of the problem, and resort to accepted heuristics for the *wavelength* assignment. A few other provide new approaches for wavelength assignment, but fail to fully consider the intertwined relation between routing and wavelength assignment, especially in a dynamic setting.

5.3 Methodology

We propose a solution method based on Graph Neural Networks and Deep Reinforcement Learning. We use GNNs to model the problem at each time so that a DRL agent be able to learn how to operate it to achieve an optimization goal. An overview of the proposed framework is presented in Figure 5.1.

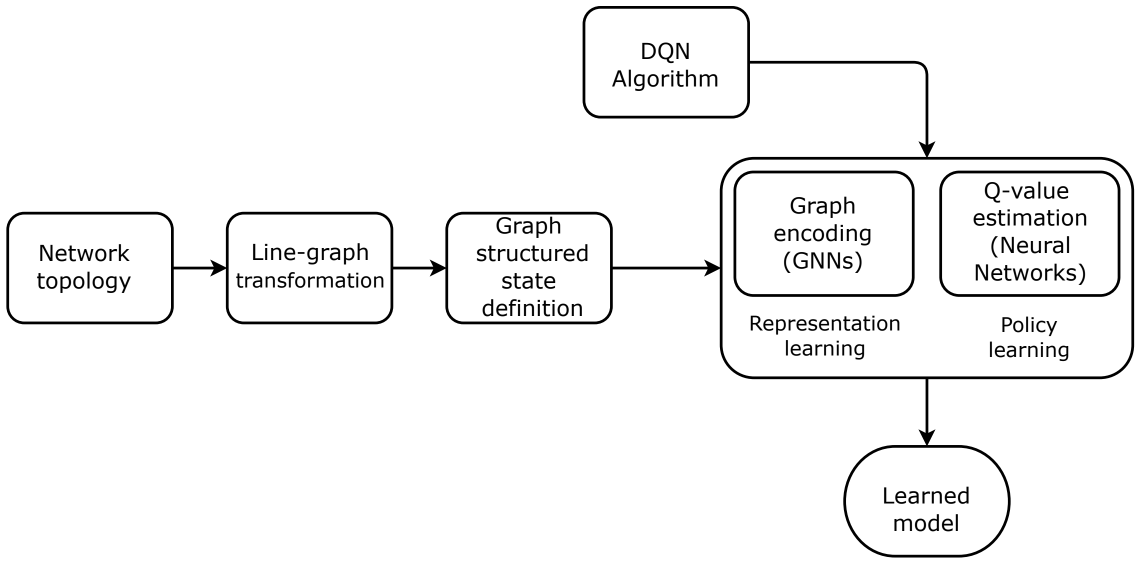


Figure 5.1 Overview of the proposed framework.

5.3.1 Graph Neural Networks

GNNs [16] are novel neural networks that operate over graph-structured inputs. They generate a vectorial representation of an input graph. This is done by an iterative process that associates the latent features of the neighbouring elements (nodes and edges) of a graph. Through this iterative message-passing algorithm, the features of these elements are updated to generate and learn an output vector. We use Graph Attention Networks [23].

The final learned vectorial representation of the graph encodes hidden information of the graph that can be used to solve complex Combinatorial Optimization (CO) problems [2]. Since Network topologies can be directly translated to a graph, using GNNs offers more advantages than applying other neural networks architectures (fully connected neural networks, convolutional neural networks, and etc.). This potential performance is shown in recent works [116, 17].

5.3.2 Deep Reinforcement Learning

In Deep Reinforcement Learning, an agent interacts with the environment by performing actions and observing information about the environment [117]. The information contains the state of the environment and an immediate reward, evaluating the performed action. DRL agents aim to maximize a long-term goal in an optimization problem. They have no prior knowledge of the environment. They learn the optimal policy by exploring the environment and this search is directed by a reward function. At each state $s \in S$, an action $a \in A$ is performed. The state is thereby transitioned to a new state, s', and the environment sends a scalar immediate reward r to the agent. The DRL agent finds a strategy to maximize the cumulative reward when the episode is finished. This problem can be modeled as a Markov Decision Process.

Q-learning is an RL algorithm that learns an optimal policy π, a mapping from states to actions [14], and is used to solve MDPs. Throughout the algorithm, the value of performing an action in a state, called Q-value, is updated according to the rewards generated by the environment. Q-values for different state-action pairs are initiated at zero or randomly generated values and are stored in a table (also known as Q-table). For a more sophisticated problem where the number of states or actions are larger, the Q-table will be substantially large, and updating Q-values will be inefficient. In these situations, Q-values are directly approximated using deep neural networks. Deep Q-Network (DQN) [15] is an algorithm used in such cases. DQN employs the advances of the deep neural networks to closely approximate Q-value functions by receiving the features of the states. The high generalization capabilities of deep neural networks enable the DRL agent to accurately approximate the Q-value for states never seen before during the training.

We consider a WDM optical network topology to be represented by a graph $G = (V, E)$ where V and E correspond to nodes and the fiber links of the physical network. For each fiber link, a set of possible wavelengths Ω indexed by ω exist. The number of possible wavelengths is fixed and denoted by $\mathcal{W} = |\Omega|$. All wavelengths are assumed to have identical capacity. At particular stochastic times τ, traffic demands are requested from the network. A traffic request is represented by a source node n^1, a destination node n^2 and a service time duration δ. We represent a lightpath request for an incoming traffic demand as ℓ_ω^κ where κ denotes the path between the source and the destination (among the k-shortest paths), and ω being the wavelength assigned to it. Feasible lightpaths are generated such that no two paths sharing a link are assigned the same wavelength and all the links throughout its path must use the same wavelength. The latter is called the wavelength continuity constraint. Figure 5.2 depicts a few possible light paths for a sample topology with $|\Omega| = 2$ and an arriving request. In the static RWA problem, a set of predetermined, incoming traffic connection requests are given [104]. The objective of this problem is generally to minimize the number of used resources, i.e., the number of wavelengths used [103].

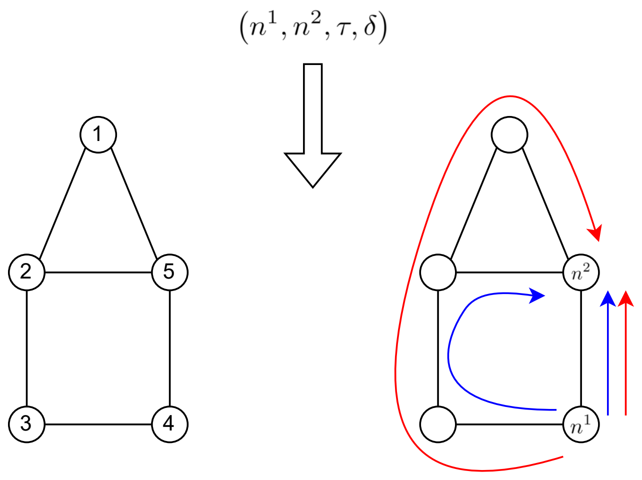


Figure 5.2 Some possible lightpaths for a sample topology with $|\Omega| = 2$ and an arriving request between nodes n^1 to n^2 at arrival time τ and for duration δ.

The dynamic RWA problem is defined as follows. For a WDM network represented by a graph G, the problem is to find a lightpath for an incoming traffic request at the time of its arrival. We assume that no wavelength conversion exists over the network; therefore, the wavelengths assigned must follow the wavelength continuity constraint. The traffic adds and drops over time by assigning new lightpaths and releasing of the wavelength by the end of their corresponding durations. The objective of the problem is to reduce the blockage rate of traffic requests over time.

Requests are generated by uniformly selecting a source destination from the set of the nodes. Each request has an arrival time and a duration. Arrival times follow a Poisson process and the durations are generated by an Exponential Distribution. These values represent the load of the network at each time of the simulation. Load is measured in Erlangs, and is calculated by multiplying the arrival rate and the average duration of the requests. It is therefore the average number of connections measured at any time in the network [107]. No information regarding these distributions and the future is available to the agent at each decision point. This results in a challenging optimization problem for the agent to devise an optimal strategy.

5.3.3 DRL Environment

Here we describe the proposed DQN method to solve the dynamic RWA problem. At the arrival of a new incoming traffic request, the agent receives the state of the environment as a graph. The request contains the source, destination, and the duration of the package. The requests do not have any preference over individual routes or wavelengths. By observing the state, in particular node and edge features, the agent decides on the lightpath to assign to the current traffic request.

The role of the GNN is to receive the state as the graph, process it, and output a new representation. To this end, the GNN aggregates the information of the neighboring nodes and edges of the graph and encodes them with a new representation. This new encoded vectorized representation of the state graph is then forwarded to a fully connected neural network that approximates the $Q(s, a)$ for all the possible actions $a \in A$ (lightpath) to be performed in state s.

Given the underlying network, the links carry the related information such as capacity and the available wavelengths at the current time during the simulation. To reflect this in our implementation, we use the line-graph $\mathcal{L}$ transformation of the original network graph G. In such a transformation, $\mathcal{L}$ has a node for each edge in G, and an edge joins those nodes if the two edges in G share a common node. This conversion facilitates the message passing steps in the GNN. Figure 5.3 illustrates a sample line-graph transformation.

The environment of the proposed DQN algorithm is formally defined as follows.

State Upon receiving a traffic demand (n^1, n^2, δ), and consequently a lightpath request, the agent observes a graph-structured representation of the environment. The GNN module retrieves the state features as well as the graph structure of the state s. The features of

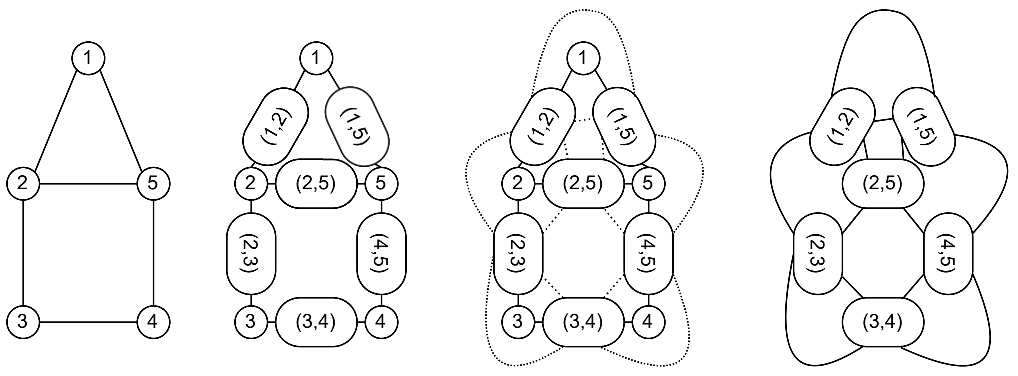


Figure 5.3 Line graph transformation example.

the graph elements illustrate the state. Node features include the wavelength assigned to a particular edge, the number of available wavelengths, the relative popularity of the wavelengths in previous selections, the remaining service times of the already assigned incoming requests, and the edge betweenness centrality [118]. The latter computes the number of all pair shortest paths that go through a particular edge in a graph network. The higher the betweenness centrality, the more likely is the edge to be selected in different paths. State features and their corresponding types are shown in Table 5.1.

Table 5.1 Features of each link in the graph-structured state.

Feature	**Description**	**Type**
f_1	Wavelength assigned to the link	One-hot encoding vector
f_2	Remaining service time of traffic request	Vector
f_3	Betweenness centrality	Scalar
f_4	Popularity of the wavelengths	Vector
f_5	Number of available wavelengths	Scalar

Action Considering all possible paths between any selection of nodes of the underlying network graph results in large action space, even for small networks. There are numerous combination of links to create a path between every two node in a network. For instance, the average action space size considering all routes for the topologies used in this work is around 4500. This means that at each decision point, one of these actions need to be selected. This reduces the capabilities of the proposed algorithm as it needs to approximate Q-values for every possible path and every wavelength. To overcome this challenge, a limited number of paths are selected. For every source-destination node pair in G, the first k shortest paths are generated prior to training the model. Upon arrival of incoming traffic demand, the DQN agent receives the new representation of the state and the information of the current lightpath request (source, destination, and duration). Besides k paths to choose from, the agent needs to select a wavelength assigned to every path. Therefore, the dimension of the possible actions is $\mathcal{W}k + 1$. It should be pointed out that the action state is different based on incoming traffic requests since the source and destination nodes of requests are different. At each decision point, the action a is the decision to either (1) Assign a lightpath to the current request arriving at that time, or (2) reject the request (null action $a_\emptyset$).

$$a \in \{\ell_1^1, \dots, \ell_{\mathcal{W}}^k\} \ \cup \ \{a_\emptyset\} \tag{5.1}$$

Reward The objective of the proposed method is to minimize the blocked traffic connections (or maximize admitted ones). To this end, A positive reward is returned by performing an action (not null action) corresponding to the action a successful provisioning of an incoming traffic connection request. Therefore the reward is constructed as:

$$r = \begin{cases} 1 & \text{if } a \neq a_{\emptyset}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

5.3.4 Efficient Formulation

We observed that a large number of transitions (changing from the current state to the next state) are driven by the null action. As we assume the transition function is deterministic as a result of the selected action, such transitions are trivial. For instance, when all the potential lightpaths for an incoming request are not available, the decision is automatically driven by $a_{\emptyset}$. We propose a framework that only models transitions when $a \neq a_{\emptyset}$. In such a framework, a non-trivial state is defined at a situation where for an incoming request, at least one lightpath be available. Therefore, the cases $a = a_{\emptyset}$ do not contribute to the training process, meaning we ignore the trivial state-action pairs in our algorithm.

5.3.5 Neural Network Architecture

The Neural Network used in this paper is composed of two main modules, namely the GNN and the feed forward neural network that outputs the predicted Q-values. GNN is a process of representation learning of a graph, in particular the state graph in this problem. Therefore, the aim of this module is to learn the representative features of the entire graph by leveraging the node features and graph structure. The algorithm performs 3 steps of message passing and stacks the results. it takes node features and graph structure as an input and outputs a new set of node features. This step is called graph filtering. We use Graph Attention Networks [23] to capture the information on the nodes of the state graph. For each node, the algorithm iterates over its neighbour and aggregates the related information from their features. Since we require to achieve a new graph-level representation, we need to generate features for the entire graph from the node features. Therefore, we apply a graph pooling layer which utilizes the graph structure to guide the pooling process. To summarize, the GNN module takes the state graph as an input and outputs a coarsened graph with fewer nodes as an output. The described Neural Network architecture are represented in Figure 5.4. This abstract, high-level representation of the state graph is then passed through a Feed Forward Neural Network to approximate the Q-values for the current state and action. We use 32 hidden layers, each with 128 hidden units and Rectified Linear Units (ReLU) activation function.

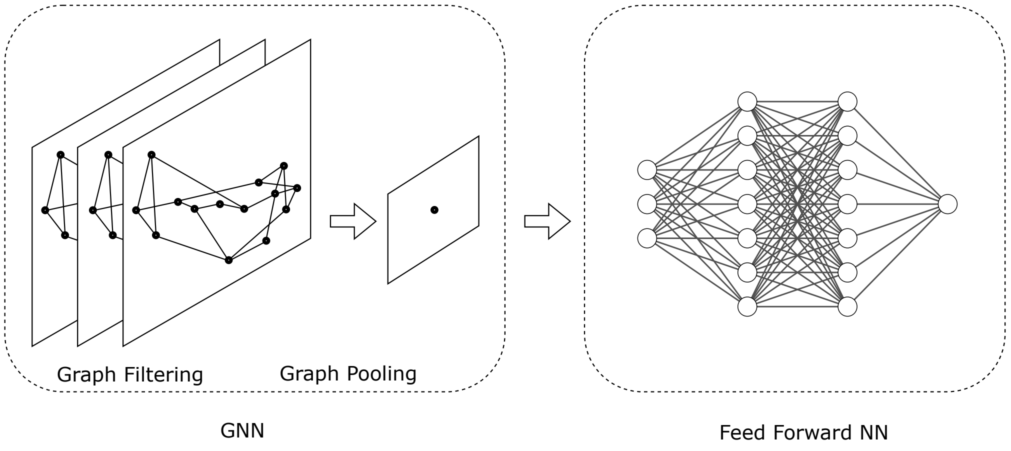


Figure 5.4 Neural Networks architecture is comprised of two modules: Graph Neural Networks and Feed Forward Neural Networks.

5.3.6 Learning Algorithm

The proposed algorithm is represented in Algorithm 2. In the beginning, we create a replay memory buffer to store the experiences at the memory initialization iterations and during the training phase (line 1). By initializing the environment and generating an instance (lines 7, 8), we create a graph G based on the underlying topology where the nodes and edges of the graph correspond to the nodes and the fiber links of the original network, respectively. Centrality betweenness is computed for all the links and $k = 4$ shortest paths are generated for each pair of nodes. As described in 5.3.3, we generate a line-graph $\mathcal{L}$ and the edge feature of G is converted to node features for $\mathcal{L}$. Arrival times and duration of service for each traffic request are created following a Poisson and an Exponential distribution, respectively. This information is not known to the agent, and they only become available as they are introduced at the corresponding arrival time (line 10). In general, the size of the output layer of the feed forward neural networks used in DQN algorithms equals the size of the action space. The value of each output unit in such a structure represents the predicted Q-value for each possible action at the current state of the environment. In this study, however, as the set of possible actions is different at each particular state, we embed the information of these possibilities into the state features. This is called the *action-in* network structure for DQN. The output layer of the feed forward neural network is thus comprised of one unit, which also depicts the Q-value for the current state-action. To this end, at each state s, we allocate routes and wavelengths and compute Q-values (lines 11-15). A ϵ-greedy action selection strategy is devised for selecting action a from the set of possible lightpaths (line 16). The state is transitioned to a new state and a reward signal is sent to the agent (line 21). The current experience tuple (s, a, r, s') is added to the replay memory buffer and replaces a previous one with lowest priority (line 22). The environment awaits the arrival of the successor incoming request and upon arrival, the features of the new state are updated. The algorithm is repeated for a predefined number of episodes. The objective of the proposed algorithm is to learn the weights of neural network framework such that the predicted Q-values accurately estimate the expected long-term collected rewards. In training, we batch samples of size 32 from the memory buffer.

In order to capture the features and structural information of the elements of the graph, we use Graph Attention Networks [32]. For the message passing, we configured 3 layers of graph filtering, and one layer of graph pooling on top of that. The number of features at each layer of graph filtering is a hyperparameter. In this work, we set the number of features as $\{128, 64, 32\}$ for each of these layers, respectively. To further smoothing the final embeddings, apply a max-pooling layer on the output of graph filtering layers. This layer can be used as in independent operation to produce a coarsened representation of the current graph. In this work, however, the output of the the pooling layer is one node with a set of features that are generated by max-pooling. The result vectorial representation of the input graph data is then forwarded to a feed forward neural network for the prediction tasks.

Once Q-values are computed and an action is selected, we update the weights of the neural network in order to minimize the loss between the predicted values and the real values (lines 25-29). Since we do not have access to real values, we approximate them following the target network [117]. To this end, Adam optimizer [90] is used. When the episodes are all finished, we save the weights of the trained model for solving new instances (line 30).

After experimenting with different values for the learning rate during the training phase, a rate of 7×10^{-7} resulted in stable learning. By using larger learning rate in our experiments during the training phase, the performance on the validation set fluctuates after 5000 episodes. This means that the agent cannot reach near optimal solution as the step size in its search strategy is too large, although the search direction could be correct. By setting smaller learning rates, the algorithm converges but very slowly. At each iteration, we use a batch of 32 samples drawn from the experience replay buffer. Prior to the training phase, we store 50000 samples in the experience buffer using Prioritized Experience Replay [119]. Samples with the lowest priority are replaced with the new samples during training.

We select actions following the ϵ-greedy strategy. For the initial 200 iterations of training, the agent explores the environment with the exploration rate of $\epsilon = 1$. This rate is then decayed with a rate of 0.995 after every episode until it reaches the minimum value of $\epsilon = 0.01$. We also update the weights of the target network every 50 iteration by cloning the values from the policy network of our DQN agent. The weights of the policy network are initialized following Xavier initialization [95].

Algorithm 2 DQN algorithm for the dynamic RWA Problem

```
1:  $memory\_buffer \leftarrow \{\}$
2:  $total\_reward \leftarrow 0$
3:  $\mathbf{w}$ initial weights of the Neural Network
4:  $k$-shortest-paths initiated for all source-destination pairs prior to training
5:
6:  for episode in $0, \ldots,$ num_episodes do
7:      $environment \leftarrow initialize\_environment$
8:      $instance \leftarrow generate\_instance$
9:      for $i$ in $0, \ldots,$ num_requests do
10:         $s \leftarrow (n^1, n^2, \delta, \tau,$ current_time$)$
11:         for $\kappa$ in $1, \ldots, k$ do
12:             for $\omega$ in $1, \ldots, |\Omega|$ do
13:                 $q\_values[\kappa][\omega] \leftarrow compute\_q\_values$
14:             end for
15:         end for
16:         $a \leftarrow epsilon\_greedy\ (q\_values,\ s)$    ▷ action $a$ is comprised of a route and a
    wavelength (lightpath)
17:         if $a = a_\emptyset$ then
18:             Reject current request
19:             skip to the next request
20:         end if
21:         $s', r \leftarrow make\_step\ (s, a)$
22:         $memory\_buffer \leftarrow (s, a, r, s')$
23:         $total\_reward \leftarrow total\_reward + r$
24:     end for
25:     for $j$ in $0, \ldots,$ batch_size do
26:         $e \leftarrow get\_sample$
27:         $\mathcal{L}_j(w) \leftarrow$ loss of $e$
28:     end for
29:     update the weight vector $\mathbf{w}$
30:     save $\mathbf{w}$
31: end for
```

5.3.7 Solving Algorithm

Once the model is trained, we can use the saved model to solve new instances following the Algorithm 3. To this end, we take the best model observed during training. This is the model that resulted in the highest reward on the validation set (line 1). We initiate the total cumulative reward (line 2), the environment (line 4), and the instance (line 5). For each incoming connection request, a state graph is generated (line 7) and Q-values are computed for the corresponding action space (line 8-12). We use a greedy strategy to select the lightpath resulting in the largest estimated Q-value (line 13). The state is thereby transitioned into a new state and a positive reward is returned (line 14), and accumulated reward is updated (line 19). Once all the requests arrive, the accumulated reward, i.e., number of admitted incoming traffic connections is returned (line 21).

Algorithm 3 Solving Algorithm.

```
1: $\mathbf{w} \leftarrow$ best model saved during training
2: $total\_reward \leftarrow 0$
3:
4: $environment \leftarrow initialize\_environment$
5: $instance \leftarrow generate\_instance$
6: for $i$ in $0, \ldots,$ num_requests do
7:     $s \leftarrow (n^1, n^2, \delta, \tau,\ current\_time)$
8:     for $\kappa$ in $1, \ldots, k$ do
9:         for $\omega$ in $1, \ldots, |\Omega|$ do
10:            $q\_values[\kappa][\omega] \leftarrow compute\_q\_values$
11:        end for
12:    end for
13:    $a \leftarrow$ argmax($q\_values,\ s$)
14:    $s', r \leftarrow make\_step\ (s, a)$
15:    if $a = a_\emptyset$ then
16:        Reject current request
17:        skip to the next request
18:    end if
19:    $total\_reward \leftarrow total\_reward + r$
20: end for
21: return $total\_reward$
```

5.4 Experimental Results

In this section we evaluate the performance of the proposed algorithm, and corresponding results are discussed here as well.

5.4.1 Setup

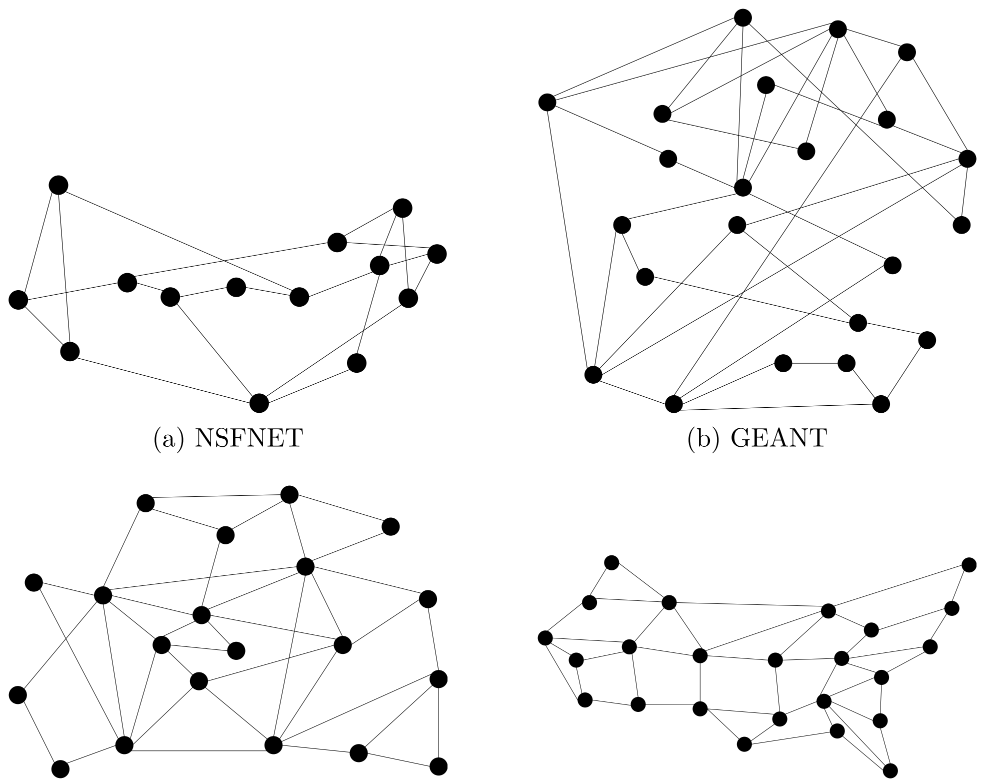
Our model is implemented in Python 3.8 using PyTorch 1.11.0. Training is carried out on one GPU (NVIDIA V100 Volta, 32GB memory). Prior to training, we create a fixed *validation* *set* to keep track of the performance of the training and compare with baselines. This validation set is comprised of 100 randomly created instances. The mathematical programming formulation of the model is solved by Gurobi 9.5.0 with a system containing 16 GBs of RAM with 4 core Intel Core i7 processors. Random instances are generated by producing random arrival times and service hold time for incoming traffic requests. In this work, we generate arrival times based on a Poisson process with a mean of 0.6. Service hold duration for the requests follow an Exponential distribution with the mean of 100. These values are selected after several experiments, in order to maintain the load of the network at an acceptable level. Having a very low load or a very high load will result in situations where the majority of the traffic requests are admitted or rejected. Prior to training, we solve these instances with widely used baselines in the literature. For routing, we select Shortest path selection and Alternate Routing methods [9]. As for the wavelength selection, First fit heuristic is selected. We also developed a simple random selection strategy to evaluate the other methods against it.

In addition to these baselines, we assumed a case where we have perfect information of the arrival time and duration for all the incoming traffic requests. We formulated this problem with a mathematical programming model presented in Appendix. Although this problem is in essence different with the dynamic RWA problem as we do not have stochasticity, its solution is an upper-bound for the problem studied in this work.

5.4.2 Experiments

Figure 5.5 represents the National Science Foundation Network (NSFNET) [120], the Gigabit European Academic Network (GEANT), the European Optical Network (EON), and the USA network. We trained an RL agent for each topology. For solving new instances, we chose the model that was learned on the NFSNET topology, as the network has a smaller size and lower average node degrees. We apply the learned model on all graph topologies. Table 5.2 depicts different topologies and their corresponding information which we used in this work. For each incoming traffic connection requests, the source and destination are selected randomly. The incoming traffic connection requests arrive following a Poisson process and each request stays on the network for a duration that follows an Exponential distribution.

In Figure 5.6, we demonstrate the performance of the learning algorithm for NSFNET (a), GEANT (b), EON (c), and USA (d) . We show the performance of the applied model at every 100 episode on the validation set. The results of applying the other baselines are also illustrated. These training curves suggest that the learning algorithm is able to infer the crucial information from the state (using GNNs), and take actions that result in maximum



(a) NSFNET

(b) GEANT

(c) EON

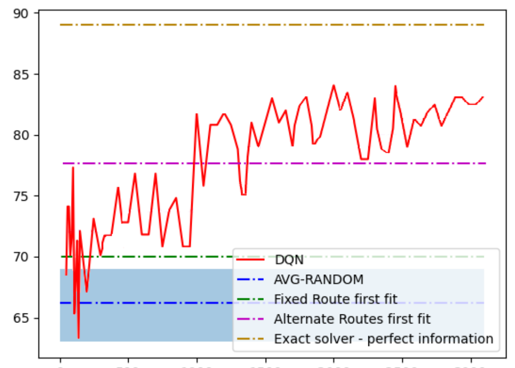
(d) USA

Figure 5.5 Network topologies.

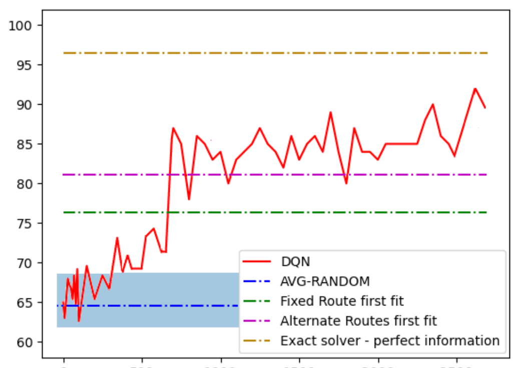
Table 5.2 Different topologies.

Topology	**Nodes**	**Links**	**Avg. Degree**
NSFNET	14	21	3.0
GEANT	24	37	3.1
EON	20	36	3.6
USA	24	44	3.7

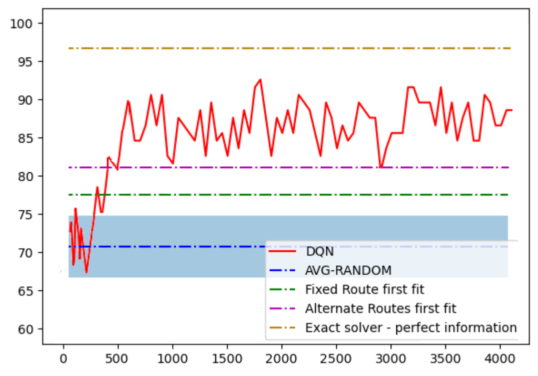
cumulative rewards. We can observe that after approximately 1000 episodes, the learning remains relatively stable. It should be pointed out that the ϵ parameter started to decay at iteration 200.



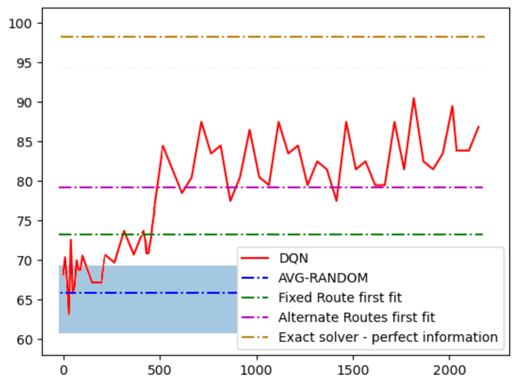
(a)



(b)



(c)



(d)

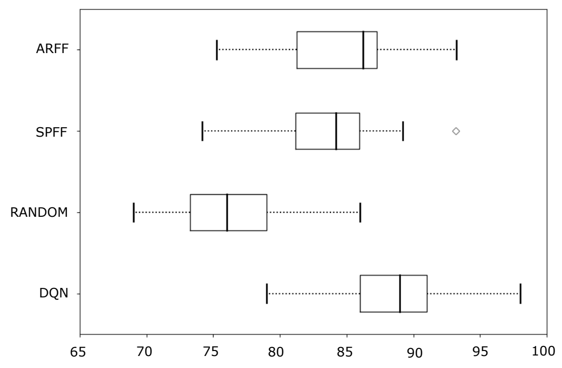
Figure 5.6 Evaluating the performance of the DQN algorithm during training for (a) NSFNET , (b) GEANT , (c) EON , and (d) USA topologies.

We ran experiments to evaluate the performance of the proposed algorithm and the learned model on new instances that did not participate in training. Since the objective is to minimize the number of rejected incoming traffic connection requests, we check the percentage of the blocked requests. In the first set of experiments, the generated instances are from the NSFNET topology. These include the instances used during training, or for solving after the model is learned. In the second set of experiments, we use the same learned model using NSFNET, but we apply this to instances generated from the other topologies (GEANT, EON, USA) that have not been seen during training.

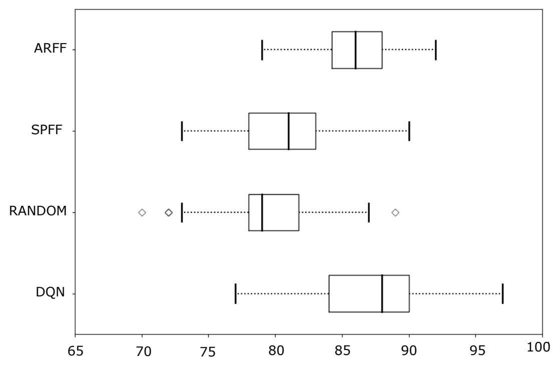
To this end, we run four separate simulation sessions. We randomly create 100 instances of the problem. For each instance, we generate arrival time and service time duration following a Poisson and an Exponential distribution, respectively. Within all the steps that the model is saved during the training phase, we select the one that revealed to be the best performing on the validation set to solve for the new instances. To be able to compare these results, we solve these instances with the accepted methods which are widely used in practice and in previous studies. For routing, shortest path and alternate routing are chosen. To assign the wavelengths, we opted for the first fit method which favors selecting more wavelengths against resulting in congestion and therefore blockage of the incoming requests. We also solved these new instance using a random strategy. At each decision point for an incoming traffic connection, the lightpath comprising of a route and a wavelength is generated randomly.

Figure 5.7 shows the percentage of blocked traffic connection requests for the described simulation scenarios. As demonstrated, the rate of blocking for the proposed method is less than the other baselines which is the goal of this algorithm. The random lightpath selection strategy shows the worst performance. Although for some cases the random selection could work better than other baselines, the median and the mean of the results for this strategy is lower than others. Although selecting the shortest paths at each decision point appears to be a logical approach, it only considers a myopic view of the problem. Always performing a greedy approach (by selecting the paths with smaller number of fiber links) may fall in local optimal points. As the results suggest, opting for the alternate routes proves to be better. Comparing the median and the mean of these two methods reveal the slightly better performance of the latter. The DQN-algorithm however, outperforms other approaches. In the best case, less than 3% of the requests are blocked. The median of the results of the DQN algorithm is around 3% better than the one of the next best performing approach (alternate routing), and about 13% better than random policy.

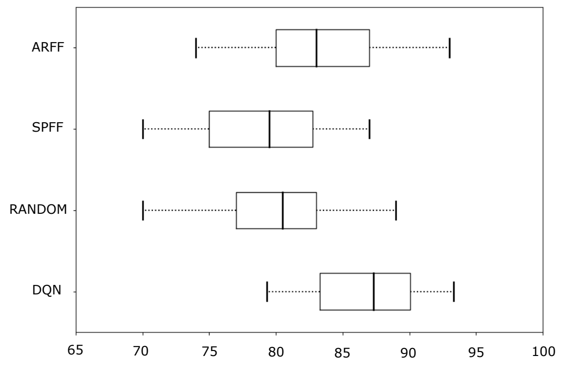
For the results presented above, we apply the model learned on the NSFNET topology to the newly created instances that even though did not participate in training, have the same underlying topology. To determine the generalization capability of the algorithm, we solve new instances from a different network topology (GEANT, EON, USA) and compare them with their corresponding baselines. Similar to the previous experiment, we generate 100 new instances following the new topology, and solve them with the proposed DQN agent that is learned over NFSNET. The results are illustrated in Figure 5.7(b-d). For GEANT network topology, random selection policy shows the worst performance. For this relatively bigger and more dense network topology, the relative performance of the alternate routing against the shortest path is better than the previous experiment. This is demonstrated via the difference in the median of these two methods across the 100 test instances. The DQN agent trained on NSFNET topology performs well on these instances compared to the best baseline. Although the median value is almost the same (still better in DQN-algorithm), the proposed RL algorithm has an edge in the maximum value. Random selection results in a slightly better performance compared to Shortest path first fit strategy for the instances created from the EON network in 5.7(c). DQN method and Alternate Routing First Fit demonstrate relatively similar behaviour in terms of the best solutions, however, the median in the DQN method is around 7 units better than that of the ARFF which in practice results in more profits.



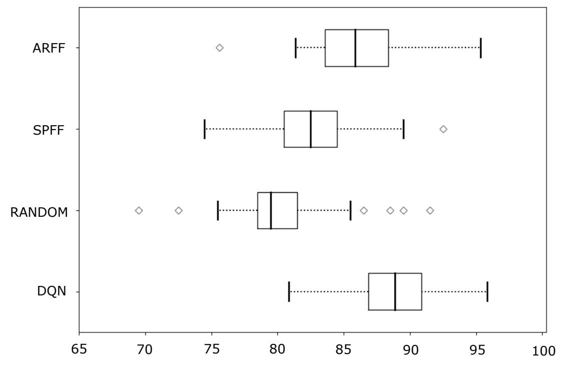
(a)



(b)



(c)



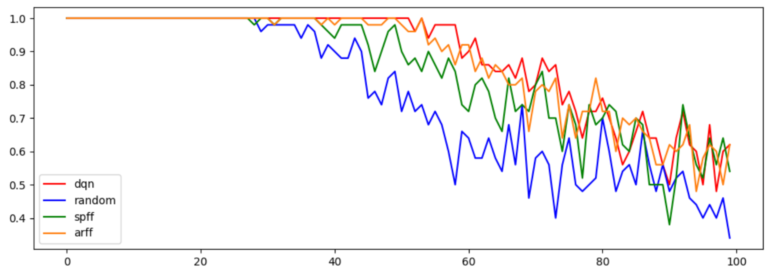
(d)

Figure 5.7 Comparison of performance of the DQN-algorithm and the baselines with the model trained on NSFNET. The results are shows for new instances that are generated following the (a) NSFNET , (b) GEANT , (c) EON , and (d) USA topologies.

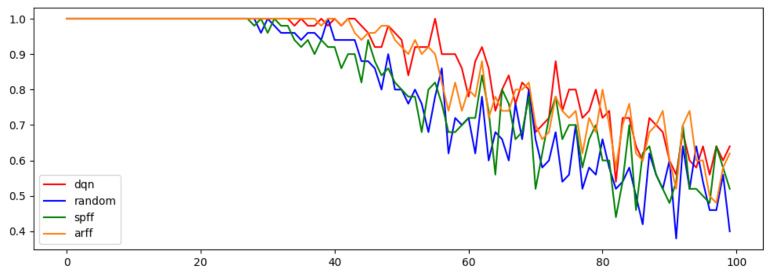
To observe the characteristic of action selection between the different methods, we evaluate their corresponding action selection behavior. In Figure 5.8, we represent the average number of admitted incoming requests over the instances created at the test phase. Similar to above, the DQN agent is trained on NSFNET topology and the performance of the algorithm over test instance from NSFNET, GEANT, and EON topologies are illustrated in Figure 5.8(a), (b), and (c), respectively. The x-axis shows the number of the request and the values of the curves is the average of times a particular request is admitted. As expected, as the time goes on, considering the load, the network gets more congested and less possibilities exist. However, the long-term approach of the DQN method results in more average accepted traffic. The random strategy reveals to be the worst performing one. The alternate routing approach has an edge over the shortest path because of the relatively myopic approach of the latter. Figure 5.9 demonstrates the length of the selected lightpaths for different methods for instances from NSFNET (a), GEANT (b), and EON (c). It could be observed that the random strategy selects lightpaths with relatively longer routes compared to other method; however, after arrival of around half of the incoming traffic, the capacities rapidly depleted and a decrease in admittance is observed. The length of the accepted routes also decrease. On the contrary, always opting to take shortest routes (greedily) results in suboptimal solutions. As this Figure suggests, the rejected connection request rapidly increase for he SPFF method.

5.4.3 Conclusion

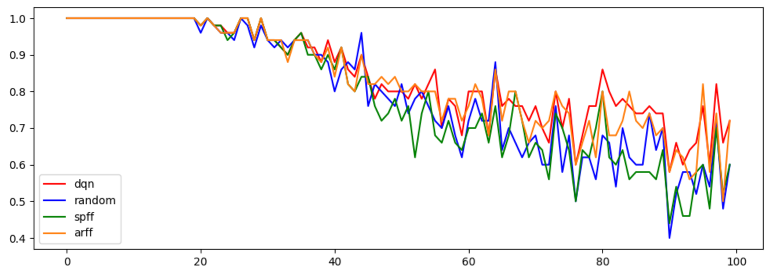
This paper studies a dynamic RWA algorithm based on Deep Reinforcement Learning that simultaneously selects routes and wavelengths. By inferring the information from the elements and the structure of the network, it devises a strategy to minimize the blockage of the incoming requests in the long term. Simulations show that the proposed algorithm performs superior to widely used solution methods to solve the dynamic RWA problem. We demonstrate the generalization capability of the proposed model. The GNN module of the training learns a new representation of the graph-structured input regardless of the topology of the graph. In other words, computations within GNN module only takes into account node and edge features, which is similar in size for any network topology. Therefore, we can train the model using a single topology, and solve instances arising from variety of topologies (not present during training) which works better than the current methods used in practice.



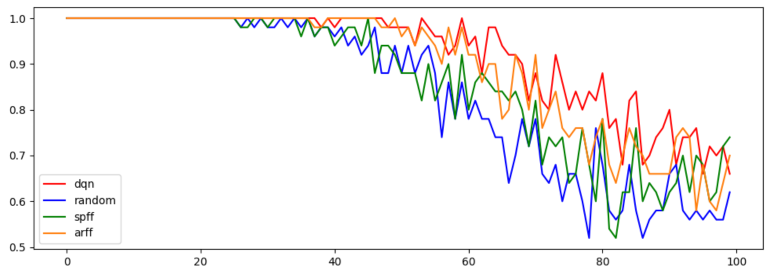
(a)



(b)

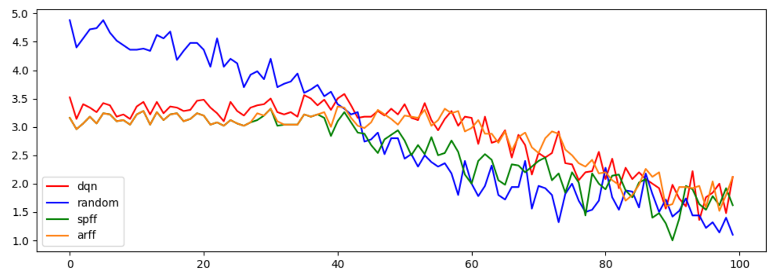


(c)

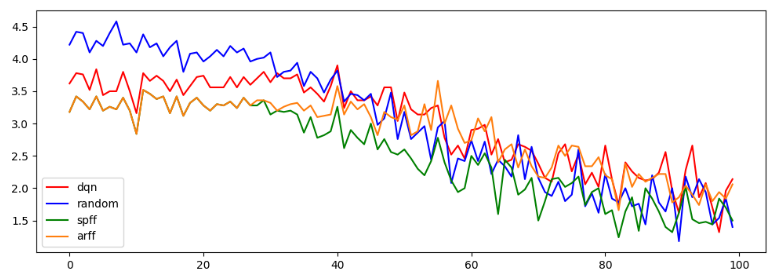


(d)

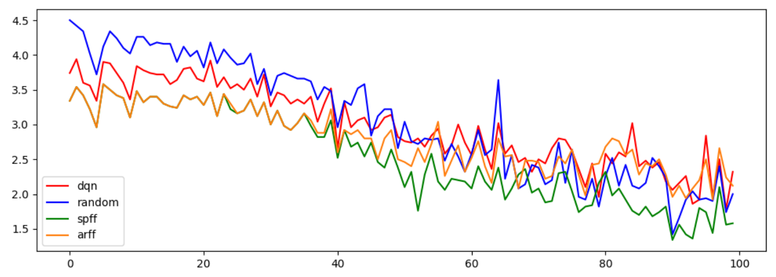
Figure 5.8 Average admitted traffic for each request over test set instances. (a) NSFNET, (b) GEANT, (c) EON, and (d) USA.



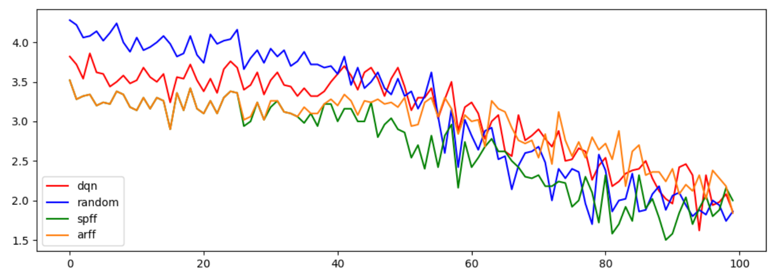
(a)



(b)



(c)



(d)

Figure 5.9 Average length of the lightpath for each request over test set instances. (a) NSFNET, (b) GEANT, (c) EON, and (d) USA.

CHAPTER 6 DYNAMIC MULTI-RESOURCE CONSTRAINED SCHEDULING WITH GNN-BASED DEEP Q-NETWORKS

6.1 Introduction

Today's production systems must cope with sophisticated customer demands and the requirements of the shareholders. Companies need to address these challenges and be capable of manufacturing in highly uncertain conditions and try to shorten production times to increase throughput and productivity. All these concerns can be addressed through formulating and solving scheduling problems. In this setting, we consider a scheduling problem that arises from a real use case in the industry. We consider a setting, where jobs require additional resources besides machines to be processed. In this setting, each job is completed by the use of some raw materials, which must be processed on machines. To transfer and add the materials to machines, one of the three available resources is required. The type of resource is defined by the job in a deterministic manner. In other words, each job requires either a robotic arm, a liquid dispenser, or a human operator in addition to a machine. Since the number of these resources is limited (one resource per type), and the fact that arrival times of the jobs are stochastic, leads to a challenging scheduling problem.

Many approaches have been taken in the literature to address scheduling and production optimal control. A very comprehensive literature review is presented in [121]. The scheduling problems in general are NP-hard, therefore finding exact solutions can be very expensive. Therefore, many researchers apply metaheuristics and heuristics to solve these problems. Various types of scheduling problems such as flow shop or job shop scheduling problems are vastly studied. Methods such as particle swarm optimization, and evolutionary algorithms applied to these problems are presented in [122]. The potential of machine learning to deal with these issues in production systems is discussed by Kang et al. [123]. In dynamic situations, having the perfect information about the future production system can lead to an increase in the performance of any dynamic scheduling approach. However, this real-time knowledge of the production is not available in practical cases. In such cases, the capabilities of methods from machine learning and reinforcement learning can be exploited [124, 125]. In order to respond to the dynamic changes in the production line from a cloud manufacturing perspective, Chen et al. [126] proposed a scheduling method based on reinforcement learning to select tasks in real time.

Machine learning techniques include supervised, unsupervised, and reinforcement learning (RL). The first two categories deal with finding innate patterns among historical data, mostly for further predictions and decision-making. Reinforcement learning, however, revolves around training an agent via trial-and-error and interactions with the environment without any prior knowledge to find the optimal policy regarding the objective to achieve [11]. The agent is thereby capable to adapt to highly uncertain conditions. As a result of this flexibility and the characteristics of modern production systems, RL is well suited for problems arising in this domain. Based on recent studies, RL has the ability to outperform state-of-the-art heuristics [127].

The contribution of this particular scheduling setting is as follows: We propose a Deep Reinforcement Learning algorithm to assign machines and corresponding resources (robotic arms or a human operator) to jobs after they have arrived in the system. Experimental results suggest high-quality solutions compared to conventional baselines. Using the GNN-based deep reinforcement learning algorithm enables us to find a schedule of the jobs that arrive dynamically with an unknown distribution (to the scheduling agent) that outperforms other baselines.

The remainder of this work is organized as follows. Section 6.2 provides background on the methods of machine learning used to solve the scheduling problems. Section 6.3 discusses reinforcement learning and Graph Neural Networks that will be used in this work. We define the problem in Section 6.4. Section 6.5 discusses the environment, Neural networks architecture, and the learning algorithm in detail. Experiments and the results are represented in Section 6.6 and finally, the conclusion of this work is in Section 6.7.

6.2 Related Work

In this section, we provide previous studies that exploit the recent advances in Machine Learning and Artificial Intelligence in the context of scheduling problems.

Recently there has been a significant increase in the use of machine learning in the field of production systems. In particular, many researchers developed methods based on neural networks and reinforcement learning. Panzer and Bender [128] publish a detailed survey of the use of deep neural networks in the field of production systems. They discuss that of the most active discipline is scheduling. The first application of neural networks in this field dates back to [129]. Their study showed higher performance and lower costs in comparison with the manual system design for the job shop scheduling problem. For a dynamic distributed job shop scheduling problem, Zhou et al. [130] minimized the completion time of all the tasks, for a set of randomly arriving instances. They considered queue times for the incoming jobs and proposed a deep reinforcement learning algorithm for this problem. Deep RL algorithms are shown to be more efficient and faster compared to conventional heuristics in different domains with particular objectives, such as reducing makespan for medical mask production [131] in times of COVID-19 pandemic, reducing tardiness for repair scheduling operations [132], increasing profitability in chemical scheduling by dealing with shifting demands and fluctuating costs [133], or reducing costs in paint job scheduling for the automotive industry [134] by using deep reinforcement learning algorithms for scheduling.

Discipline-specific scheduling problems have been addressed by a number of researchers. To address the volatility in the mold injection industry, Deep Q-networks are used to reduce the weighted tardiness of this dynamic environment. This method demonstrated better performance than the current dispatching rules [135]. Lin et al. [136] propose a multi-class Deep Q-Networks algorithm for a special use-case, semiconductor manufacturing, to reduce the makespan. Using reinforcement learning methods based on policy gradients, especially proximal policy optimization (PPO), has attracted many researchers. By considering several features such as setup times, Park et al. [137] developed an algorithm based on PPO that works better compared to the conventional heuristics such as shortest setup time or shortest processing time, in terms of minimizing the makespan for semiconductor manufacturing. A better generalization was observed by embedding Graph Neural Networks in the learning algorithm for the job shop scheduling problem [138]. This new algorithm was proven to be superior to scenario-specific configured algorithms as well. It is shown that few studies have been adopted in real environments [128]. They discuss that developing and implementing of scheduling policies based on reinforcement learning for established production environments is critical.

Aydin and Öztemel [74] proposed a variant of the Q-learning algorithm to train an agent to dynamically select scheduling rules. The action set used in their work consists of a few heuristic rules adopted in for the scheduling problems in the literature. At each decision point, the set of possible actions are jobs that satisfy the Shortest Processing Time (SPT), Cost Over Time (COVERT), and Critical Ratio (CR). They show the superiority of their algorithm compared to individual options. By using the average tardiness as the reward signal, Wei and Zhao [139] proposed a Q-learning algorithm to select scheduling rules to minimize the mean tardiness of the final schedule. They both used a tabular version of Q-learning. Similar to the work of [74], other researchers developed heuristics to create priority lists as actions in their Reinforcement Learning frameworks. Zhang et al. [140] used Weighted Shortest Processing Time (WSPT), Weighted Modified Due Date (WMDD), Weighted Cost Over Time (WCOVERT), and Ranking Apparent Tardiness cost with setups (RATCS) as actions to solve the parallel machine scheduling with constraints. Mao et al. [73] presented a deep reinforcement learning approach to deal with the dynamic resource management. to this end, similar to [72], they utilized graphical features of the current schedules to identify different states. They used convolutional neural networks to capture the critical information of the images during their algorithm. Since scheduling problems can be well defined over graphs, Park et al. [138] adopted a PPO algorithm to train GNN for the Job Shop Scheduling Problem to minimize the total makespan. Similar to the methods mentioned above, they opted for a few heuristic rules as possible actions. These limited options may lead to a decrease in the performance of the algorithms.

6.3 Technical Background

In this section, we introduce graph Neural Networks and Reinforcement Learning.

6.3.1 Graph Neural Networks

Graph Neural Networks (GNNs) [16] is a machine learning architecture designed to operate over graph-structured data. For any problem that can be represented as a graph, the aim of the GNNs is to compute a vectorial representation of each node of the input graph. This is done by aggregating the features of the neighboring nodes and edges. The new vector representation of the input graph embeds crucial graph structures that help to solve Combinatorial Optimization problems. Cappart et al. [2] and Vesselinova et al. [141] present a review of key advances in how GNNs have been recently used as building blocks for Combinatorial Optimization problems, either as solvers or by enhancing exact solvers. The potential of GNNs is shown in multiple domains where data can be represented as a graph structure [20, 21, 142]. GNNs may aim to learn good features for each node of the graph, such that node-focused tasks be facilitated. On the contrary, for graph-focused tasks, they target on learning a new abstract representation of the entire graph in which learning node features is an intermediate step. In both these cases, particular node features and the structure of the graph are taken into consideration. The process of generating a new representation for node features is called a graph filtering operation. Generally, several consecutive stacks of graph filtering layers are used to get the final node features. To obtain a new graph representation, pooling operations can be employed on top of the graph filtering layers [18].

In the proposed approach, the GNN receives the graph-structured input of the orders and generates a vectorized representation of the state by applying consecutive graph filtering operations. The nodes of the graph represent the tasks and the edges show the order of the tasks corresponding to a specific job. This new representation of the state encodes latent information between the nodes and edges of the state graph. For each node, the features of the neighboring nodes and edges are aggregated through a message passing mechanism. The GNN is trained in an end-to-end fashion to learn the state vector. This new representation is then ready to be forwarded to a feed-forward neural network to estimate the values of the actions at the current state.

6.3.2 Deep Reinforcement Learning

Reinforcement Learning (RL) algorithms learn a policy to maximize the expected value of an objective function of an optimization problem. Unlike supervised learning, there is no labeled data in the context of the RL. The RL agent learns this policy by iteratively interacting with the environment. Therefore, there is no need for previous expert knowledge about the dynamics of the environment. The agent explores the state and action space and the search is directed by a reward function. Let $\mathcal{S}$ and $\mathcal{A}$ represent the set of states and actions, respectively. By performing an action $a \in \mathcal{A}$, the state is transitioned to $s' \in \mathcal{S}$ and a reward r is received by the agent. The promise here is to find the policy that maximizes the expected total reward during its lifetime by a sequence of states $s_t \in \mathcal{S}$ with $t = 1, \ldots, \Theta$. Such sequence is called an episode and s_Θ denoted the terminal state.

Q-learning [14] is an RL algorithm to learn the value of performing an action at a specific state, called Q-values. The Q-learning agent learns a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to maximize the expected cumulative rewards in the successive steps until termination. The initial Q-values for every possible state-action pair, $Q(s, a)$, are set to zero (or generated randomly) in the form of a table. During training, these values are updated by performing actions and receiving the corresponding reward from the environment. Q-values denoted the expected cumulative reward by performing action a at state s following the learned policy π.

For many problems, the number of all possible state-action pairs can be exponentially large. Therefore, storing and accessing Q-values in a table would be inefficient and would hinder the learning process. To resolve this, Deep Q-Learning [15] is developed to approximate Q-values by leveraging the generalization capabilities of Deep Neural Networks. The key idea is provide an estimator $\widehat{Q}(s, a, \mathbf{w}) \approx Q(s, a)$, where $\mathbf{w}$ is the learned vector parameterizing $\widehat{Q}$.

During the training, to create a balance between exploring different states and exploiting the known best actions, we follow a softmax action selection strategy [143]. Unlike ϵ-greedy action selection, when exploring, the softmax strategy does not choose the actions with equal probability. These actions are weighted based on their estimates. It should be noted that the greedy action still has the highest selection probability.

6.4 Problem Definition

The problem in this study is characterized by three sets, namely jobs, machines, and additional resources, which are discussed as follows.

Jobs

Jobs arrive dynamically to the system. They are independent and arrive following an unknown stochastic process. Let j denote the index of the arriving job. We assume that the interval time between the arrival of two consecutive jobs follows a probability distribution $P(t)$:

$$|\alpha_j - \alpha_{j-1}| \sim P(t) \tag{6.1}$$

where α_j is the arrival time of job j. The set of all jobs is J. In this work, we do not consider any precedence among jobs. Arrived jobs will queue up to be processed in the system, while the size of the queue is assumed to be infinite. Each job j has an arrival time α_j, processing time p_j, deadline d_j, and a expiration time e_j. Once a job arrives, it will start at a start time $\sigma_j \geq \alpha_j$. If job j reaches its corresponding expiration time e_j, it will be removed from the system and will be thrown away, and cannot fulfill the customers' order any more.

Machines

Each task requires a machine to be processed on. In this context, we consider parallel machines that are all capable of performing similar tasks. In addition, there is no difference in terms of processing power among them; therefore, they are identical. In the current problem, once a job is assigned to a machine, it needs to be processed on that particular machine until the job is finished (or expired) and leaves the system. We consider $\mathcal{M}$ machines. The processing time of job j represented by p_j is independent of the machines, solely relies on the particular job, and is deterministic.

Resources

Besides machines, any job requires another resource ρ_r for being processed. These resources are renewable, meaning their availability is constrained at a given time. However, this is not true for their consumption. The requirement of each one of them is directly reflected in the job description. The resources are non-preemptive, meaning once they are assigned to a job, they will be unavailable for other jobs until the current one is finished.

Objective

In static scheduling problems, a complete information set of jobs are available and the scheduler assigns start times to different tasks subject to the problem constraints. In a dynamic case, however, jobs arrive over time until the time horizon is finished. The scheduling agent has no idea about future events at any decision time.

The objective of the problem is to minimize the costs that are incurred to the system based on all the decisions made during the scheduling horizon. In our problem, the costs have two sources: (1) tardiness costs, and (2) expiration costs. If the processing of a job is not finished within its start time and deadline, we incur a penalty proportional to the extra time the job stays in the system. In addition, if this extra time is passed the job's corresponding expiration, it will leave the system with an expiration penalty and we lose the order. The objective function is similar to the one presented in [72] and as follows:

$$\begin{aligned} \min \quad z &= \mathcal{P}_t + \mathcal{P}_e \\ &= \sum_{j \in J} \omega_j \, (\ell_j - d_j)^+ + \sum_{j \in J} \omega_j \, \phi_j p_j \end{aligned} \tag{6.2}$$

where $(x)^+ = \max(0, x)$, ω_j is the weight of job j, ℓ_j is the time job j leaves the system and ϕ_j is binary parameter equal to 1 when job j is expired.

Constraints

Several constraints control the solution space of the problem we study here. Firstly, once a job is assigned to a machine, it needs to be carried out on the same machine, until the job is finished or the job is expired. Secondly, two jobs cannot be scheduled at the same time on the same machine. Lastly, the start times $\sigma_j, j \in J$ must be larger than the arrival time of job j.

At each decision point, the set of all jobs can be divided into four categories based on their status: (1) eligible to be scheduled, (2) finished, (3) currently being scheduled, and (4) not eligible to be scheduled. A job is eligible if it is arrived to the system, regardless of the availability of required machines and resources. The agent schedules one eligible job at each time step. We assume that for each job, the type of the required resource is known upon arrival. More specifically, a resource profile of each task is denoted by $r_j \in \mathcal{R}$ designating the resource type. As we do not consider a constrained buffer, orders can wait to be processed.

6.5 Methodology

The RL agent implements a DQN algorithm [15] and outputs the Q-values. The objective of the algorithm is to predict the $Q(s, a)$ of selecting the next task (a) to carry out at the current state (s).

6.5.1 Reinforcement Learning Environment

The reinforcement learning method is defined by states, actions, and rewards functions which are described as follows.

State

In order to take informed decisions, the agent needs to observe the state of the environment at the decision points. The proposed model is designed to find a scheduling strategy based on this observation. The state of the environment is illustrated by the state graphs node and edge features. We consider a complete, undirected, acyclic graph $G = (V, E)$. The set of jobs constructs V. An edge exists between every pair of jobs in this graph. A sample input graph with six jobs is demonstrated in Figure 6.1. We consider a complete graph, and this representation suggests that if all the jobs are available, i.e., arrived in the system, they can potentially be selected in any order one after each other, given the availability of machines and resources.

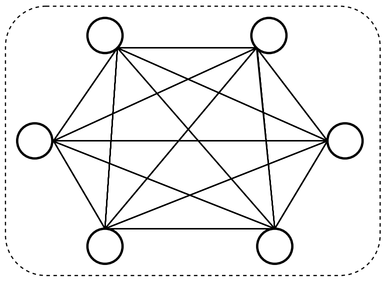


Figure 6.1 A sample graph-structured input is comprised of six jobs.

Each element of the graph contains features. For every node (job), we keep track of its status, i.e., currently being processed, finished, not scheduled yet, or eligible for selection at the current decision point. This information is stored as a one-hot encoding vector. Similarly, every job requires a particular resource, which is also represented by a one-hot encoding vector. Table 6.1 provides a description of the features of a state. All of the numeric features are standardized.

Table 6.1 Features of job j in the input graph at time t.

Feature	**Definition**	**Description**
x_1	$\{0,1\}$	Status of the task
x_2	p_j	Processing time
x_3	d_j	Deadline
x_4	$(d_j - t)^+$	Remaining time
x_5	ω_j	Weight
x_6	$\{0,1,2\}$	Required resource
x_7	$\{0,1\}$	Tardy job

Action Set

In this section, we describe how the scheduling actions are represented within our proposed algorithm. The size of the action set equals the total number of jobs to be scheduled. Although this would be a large number, at each decision point, many jobs will be unavailable, or masked, either because they have already been selected, or they are simply not available as they have not arrived in the system yet. In addition, decisions must follow the natural constraints of the system. For instance, if the required resource for a particular job is not valid at the current time, that job will be masked. At each state s_t, the action a_t is the next job to be selected. Each job can be processed by a specific resource. A job is available only if it has not been already selected. As the information about the upcoming jobs is not known and based on the limited number of resources, there could be cases where no action can be selected as the resources may be already in use. We add a particular null action a_ϕ to the action set at each state. However, since the transition to a new state is deterministic in this study, the outcome of taking a_ϕ is trivial (the current state) and no reward is collected. Therefore, we ignore these state-action pairs during the training and updates of the network.

It should be pointed out that although we have different types of resources, the resources of each type are completely identical. Therefore, assigning any one of them to the corresponding job will be the same and no decision is required here.

Reward Function

The goal of reinforcement learning is to devise a strategy that maximizes the expected cumulative reward. Therefore, the definition of the reward function is crucial to guide the learning algorithm. At each decision time t, the agent will receive a reward $R(s_t, a_t)$ in a certain state s_t after executing any possible action a_t. In general, the reward function corresponds to the objective of the problem. The goal of this problem is to minimize the costs of the schedule, while the RL agent tries to maximize the total collected rewards, so the value of the reward should be negatively correlated with the overall penalties.

Decision points are when an action (a non-null action) needs to be taken. In our framework, these points in time equal the start time of any of the arrived jobs that are eligible to be scheduled. When a job arrives (before its start time), all the related information about processing time, deadline, and expiration time are available. Therefore, once a job is selected to be scheduled on a particular machine using its required resource, we can compute its completion time c_j as:

$$c_j = \sigma_j + p_j \tag{6.3}$$

By comparing the completion time with the deadline and the expiration time, we can deduce the values of the penalties incurred by selecting this job (taking a particular action).

Let $\mathcal{J}_{(t-1,t)}$ be the set of jobs in the scheduling system from time $t-1$ to t. For each job $j \in \mathcal{J}_{(t-1,t)}$, the partial cost pc_j is:

$$pc_j = \begin{cases} 0, & c_j \leq d_j \\ -\omega_j\left(c_j - d_j\right), & d_j < c_j < e_j \\ -\omega_j\Big[\left(e_j - d_j\right) + p_j\Big], & c_j \geq e_j. \end{cases} \tag{6.4}$$

and the reward is:

$$R(s_t, a_t) = \sum_{j \in \mathcal{J}_{(t-1,t)}} pc_j \tag{6.5}$$

6.5.2 Neural Networks Architecture

To capture all the critical information and predict an accurate estimate of the Q-values, we use two separate neural network architectures for this problem: GNNs, and feed-forward neural networks (policy network).

We use Graph Attention Network introduced by Veličković et al. [32] to not only utilize the feature and structural information of the input graph data, but to consider the importance of each piece of information to make a decision at each state of the algorithm. Graph Attention Networks computes a normalized attention mechanism that realizes the importance of an element's features into obtaining a new representation of the graph data. To this end, we generated a Graph Attention Network with 3 layers of graph filtering. In each of these layers, the features of every node is updated by combining its own features and the aggregated features of its direct neighbor nodes following Equation 2.6. The size of the features at each layer is a hyperparameter that is selected as $\{128, 64, 32\}$, respectively. It should be pointed out that the attention mechanism used in Equation 2.6 contains two learnable set of parameters that are updated within the main training algorithms. The output of the Graph Attention Network is a new embedding, a new vectorial representation, of the input graph data.

These new embeddings are then forwarded to a fully connected, feed-forward neural network that models the Readout function. The objective of this network (called the policy network) is to output a prediction of taking different actions at the current state. As noted in Section 6.5.1, we mask the tasks that cannot be selected at the current state and the action is chosen following a softmax strategy among the available ones. This neural network architecture is illustrated in Figure 6.2.

6.5.3 Learning Algorithm

In this study, we use the Q-learning algorithm to solve the scheduling problem. This algorithm finds a policy (action selection strategy) π^* such that:

$$\pi^* = \operatorname{argmax}_\pi Q^\pi(a, s) \qquad \forall s \in S, \forall a \in A \tag{6.6}$$

where $Q^\pi(a, s)$ represents the value of selecting action a at state s following an arbitrary policy π. For relatively smaller problems, this policy can be found using exhaustive exploration by Dynamic Programming. In a large-size practical problem where the jobs arrive in a dynamic fashion, this will be expensive to compute. To address this issue, the Q-learning algorithm [14] provides an estimation of the Q-values at each state-action pair by iterative updates following:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_t(s_t, a_t) + \max_{a \in A_t} Q(s_{t+1}, a) - Q(s_t, a_t)\right] \tag{6.7}$$

in which α is a step size (learning rate) and the Q-values are initiated and updated using a Q-table. As the size of the problems expands, maintaining a Q-table will be difficult. Neural networks are thereby used to approximate these values [89]. We parameterize these values by a vector $\mathbf{w}$ such that $Q(s, a, \mathbf{w}) \approx Q(s, a)$. We use the Adam optimizer [90] to minimize the square loss between the approximated and the real Q-value:

$$L(\mathbf{w}) = \left[R(s_t, a_t) + \max_{a \in A_t} \widehat{Q}(s_{t+1}, a, \mathbf{w}) - \widehat{Q}(s_t, a_t, \mathbf{w})\right]^2 \tag{6.8}$$

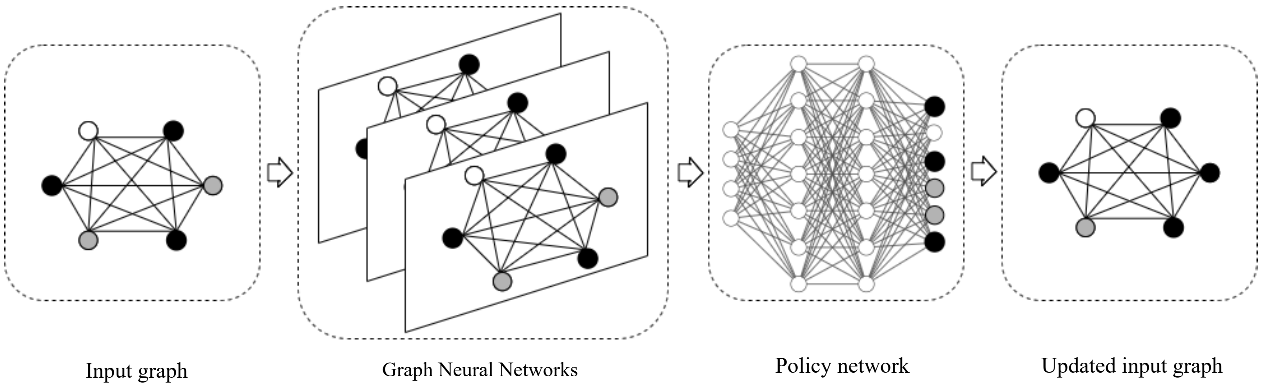


Figure 6.2 An overview of the proposed algorithm. An input graph (with node and edge features) at each state is forwarded into the GNN for the purpose of representation learning. Already selected tasks are shaded in black. The new vector representation is then forwarded to the Policy network to predict the Q-values. The next action is then selected among the eligible ones (shaded in grey) and the input graph and its corresponding features are updated.

The agent interacts with the environment in the simulation model and based on past experiences, figures out the best actions at different states that yield maximum expected cumulative rewards, or in this problem, minimize the costs. The process is represented in Algorithm 4. In the beginning, we initialize the memory buffer (line 1) and set the total accumulated rewards to zero (line 2). At each episode, we initialize the environment by setting the features of the elements of the graphs (line 5) and the initial state is generated (line 6) . At this point, no job has been started, and the eligible job (the first job arriving in the system) is identified. Processing time, deadline, and expiration time are also set. To restrict the size of the action space, we determine the set of available actions (line 8), i.e., jobs that have already arrived and are not scheduled yet. If at the current time t, there are not such jobs, we directly go to the next steps. Otherwise, an action is selected following an ϵ-greedy strategy (line 12). By taking this action, the state is transitioned towards a new one,, and a reward is returned to the agent (line 13) following equation 6.5. Now that the action is taken, the time is updated (line 15) and based on that, we update the status of the jobs, update their remaining time, and for those that are finished, we free up their corresponding resources (line 16). Once all the jobs are scheduled, we sample a subset of them (a mini-batch) from the reply memory buffer (line 17). At this point, the weights of the feed-forward neural network (policy network) are updated in order to minimize the loss function using the Adam optimizer [90] (line 20-24). To this end, at particular iterations, a snapshot of the policy network is cloned (structure and weights). This new network is called a target network. For the subsequent iterations, the output of this fixed network resembles the actual Q-values compared to the predicted ones following the policy network. At the end of each iteration, we save the model i.e., updated weights. (line 25).

Algorithm 4 DQN algorithm

```
1: $memory\_buffer \leftarrow \{\}$
2: $total\_reward \leftarrow 0$
3:
4: for each episode do
5:     $env \leftarrow$ env_init()
6:     $s \leftarrow$ state_init()
7:     while jobs arrive do
8:         $eligibles \leftarrow$ get_valid_actions($s$)
9:         if no job is eligible then
10:            wait for the next state
11:        else
12:            $a \leftarrow$ get_action($s, eligibles$)        ▷ $\epsilon$-greedy action selection strategy
13:            $r, s' \leftarrow$ env_step($a, s$)
14:        end if
15:        $t \leftarrow t + 1$
16:        update_status()
17:        $memory\_buffer \leftarrow (s, a, r, s')$
18:        $total\_reward \leftarrow total\_reward + r$
19:    end while
20:    for $j$ in $0, \ldots, batch\_size$ do
21:        $e \leftarrow$ get_sample()
22:        $\mathcal{L}_j(w) \leftarrow$ loss of $e$
23:    end for
24:    update the weight vector $\mathbf{w}$
25:    save $\mathbf{w}$
26: end for
```

6.6 Experiments

In this section, we provide an analysis of the experiments of applying our proposed method to the problem. The results are discussed here as well.

6.6.1 Setup

We implemented our proposed algorithm in Python 3.8 using PyTorch 1.11.0 library [144]. For training, we used a single GPU (NVIDIA V100 Volta, 32GB memory).

In order to evaluate the performance of the training, we sample a number of instances prior to training as the validation set. At predetermined iterations of the training, we apply the latest trained model on the validation set and output the results as the training goes on. We also solve this validation set with the baseline algorithms.

6.6.2 Baselines

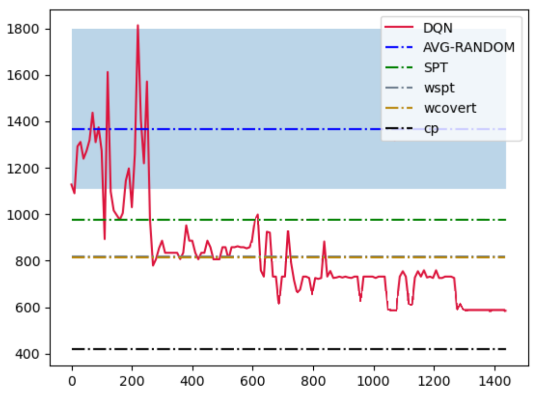
We evaluate the performance of the training by comparing it to three heuristic rules: SPT, WSPT [145], WCOVERT [146]. Shortest Processing Time favors the selection of the jobs that are already arrived and have the shortest processing time among others, as long as their corresponding required resources are available. Weighted Shortest Processing Time considers a weight on top of the shortest processing times. For each job j, we compute the value of $\frac{\omega_j}{p_j}$. If all weights are equal, the WSPT equals the SPT rule. In addition to these heuristics, Weighted Cost Over Time computes the degree of criticality of jobs used for determining job sequence. For each job j, WCOVERT is calculated as $\frac{\omega_j}{p_j} \max[0, 1 - \frac{\max[0, d_j - t - p_j]}{p_j}]$. We also generate a method that randomly selects a job among the ones that have already arrived and their corresponding resources are available.

We also formulated a modified version of this problem as a Constraint Programming model, implemented in MiniZinc, and solved it with OR-Tools 9.3.10497 FlatZinc solver with free search. To this end, we considered a problem with the perfect information where we know exactly when the jobs arrive and how long their corresponding processing times are in a deterministic way. The optimal solution to this problem will be better (lower) than the one with the online arrival of the jobs.

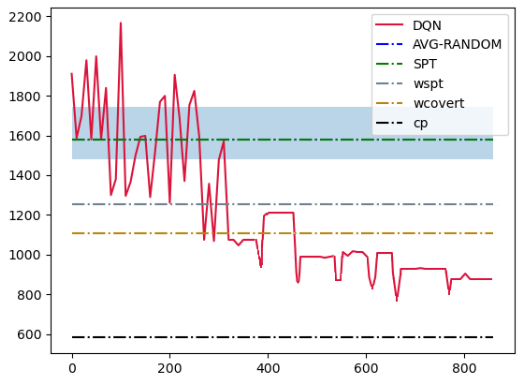
6.6.3 Results and Discussion

Figure 6.3 shows the training process of the model in three experiments. The results of applying the model on the held-put validation set are also illustrated. For the first 100 epochs of training, we only allow exploration, following the ϵ-greedy action selection strategy. After the exploration rate is decayed until it reaches a small value favoring exploitation of the best actions at each state. For a more complex setting, Figure 6.3 (c), the result of applying the trained model on the held-up validation set marginally passes the best baseline during the time limit for the training since we limit the training time to 24 hours.

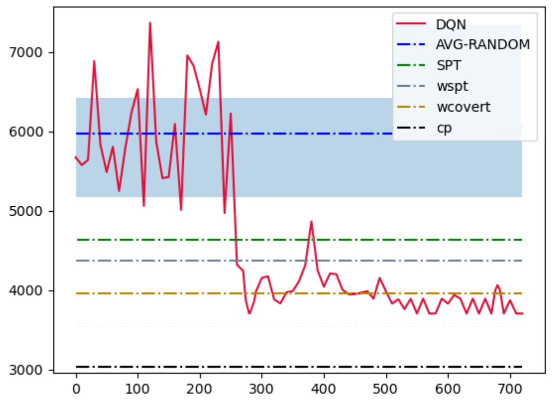
Hyperparameter values for the RL-scheduler are selected following a series of preliminary experiments. We perform 3 layers of graph filtering operations. The policy network contains 32 hidden layers, each with the size of 128 hidden units. we employ the Rectified Units (ReLu) as the activation function of the hidden layers and no activation at the output layer. The



(a)



(b)



(c)

Figure 6.3 Average training rewards (penalties) for different configurations. (a) 50 jobs, 5 machines, (b) 100 jobs, 5 machines, and (c) 100 jobs, 20 machines.

memory replay buffer stores 5000 experience tuples and at each training iteration, batches of 32 samples are selected. Prioritized experience replay [147] is selected for our implementation where samples with the lowest priority are replaced with the new samples during training. The learning rate is set to 10^{-6}. For selecting actions and in particular the exploration strategy, we opt for the softmax action selection. This approach was revealed to be superior to the ϵ-greedy strategy where it assigns a similar probability to different actions during exploration [92]. The softmax strategy selects the best action with the highest probability (exploitation); however, the other actions during exploration are weighted based on their values. At the beginning of the training, the temperature hyperparameter is set to 0 and is increased as the training goes on, thereby increasing exploitation. The network is trained for 24 hours for all the instances.

Since this problem is based on a real practical problem, we consider some parameters corresponding to the real scenarios there. For each type of resource, we have one available unit. However, the number of machines and number of orders can be flexible based on the cases in practice.

Once the model is trained, we apply it to new instances that have not been observed by the agent during the training phase. New orders are represented as a graph, and the features and the environment are initialized. The graph is forwarded to the neural network structure illustrated in Figure 6.2. At each state, the objective is to take the action that yields the maximum Q-value. To this end, the next action is selected following a greedy policy $\pi = \text{argmax}_{a \in A} \widehat{Q}(s,a)$ among possible actions (eligible tasks). it should be noted that the best model during the training is saved and is used to solve the new orders.

Table 6.2 shows the performance of different algorithms on problems identified with the number of jobs and machines, 50 jobs (with 5 machines), and 100 jobs (with 5 and 20 machines). As described earlier, the number of resources is fixed throughout the experiments. There are also two distinct robotic arms and one human operator. For each instance, we calculated the tardiness, expiration, and total penalty incurred by applying each method. As expected, randomly scheduling the jobs results in the highest cost for all the instances. As it can be observed from the results, besides the CP model, solving the problem with the proposed method results in lower total costs compared to the other baselines. To evaluate the performance of the DQN methods against the case with the perfect information modeled as a Constraint Programming model, we computed the gap between these values across different problem instances. The gap is calculated as $\frac{obj_{cp} - obj_{dqn}}{obj_{dqn}}$ and represented in Table 6.2. However, it should be pointed out that the Constraint Programming is indeed the best performing method among the baselines as it considers the case with the perfect information and provides a lower bound for the dynamic problem. This measure demonstrates the distance to the case with the perfect information.

Table 6.2 Comparison of total penalty for different methods.

Jobs - Machines	**Cost**	**RANDOM**	**SPT**	**WSPT**	**WCOVERT**	**CP**	**DQN**	**GAP (CP)**
50 -5	**Expiration**	1888	655	519	553	272	384	
	Tardiness	1290	945	824	759	912	877	
	Total	3178	1600	1343	1312	1184	**1261**	7%
100 -5	**Tardiness**	450	322	401	356	139	415	
	Expiration	352	215	189	232	73	150	
	Total	802	**537**	590	588	212	565	167%
100 - 20	**Tardiness**	237	198	217	260	163	194	
	Expiration	271	255	129	86	38	130	
	Total	508	453	346	346	201	**324**	61%

Running times for different methods are also displayed in Table 6.3. It can be viewed that the time required to find a solution for Random, SPT, WSPT, and WCOVERT are very close in different problem settings. When the number of jobs increases, in particular with 150 jobs, CP takes a rather long time to find the optimal solution, a bit more than 16 minutes. As for the proposed methods, the time required to get the solution is closer to the heuristic baselines as well which shows the efficiency of the learned model.

Table 6.3 Comparison of calculation times (seconds).

Jobs	**50**	**100**	**100**
Machines	**5**	**5**	**20**
RANDOM	5	11	18
SPT	7	12	21
WSPT	7	12	25
WCOVERT	11	12	27
CP	11	15	291
DQN	34	38	48

6.7 Conclusion

This work studies GNN-based deep reinforcement algorithm for a particular scheduling problem with constraints on the resources arising from a real use-case. In this problem, jobs arrive dynamically following an unknown distribution, with stochastic processing times where their distribution is also unknown. The algorithm captures the hidden information within the features and the relations of a graph-structured input and devises a near-optimal policy to reduce the total penalty as a result of late delivery and expiration of the orders arriving in a dynamic fashion into the system. Experimental results show the proposed algorithm outperform the baselines in a shorter computation time. However, it should be pointed out that the agent takes a rather long time for the it to learn a good model. The reinforcement learning agent learns a model to assign a particular resource at the decision points to jobs that have already arrived and available to be scheduled.

CHAPTER 7 GENERAL DISCUSSION

In this dissertation, we studied the application of Graph Neural Networks and deep reinforcement learning on three different Combinatorial Optimization problems arising in three particular applications, namely, healthcare, network optimization, and production system using robotic arms.

In Chapter 4, we presented a method based on Deep Reinforcement Learning to solve the beam orientation optimization and trajectory optimization simultaneously for the CyberKnife system in radiation therapy treatment planning. Because of the intrinsic complexity of this problem, beam orientation optimization is generally solved sequentially and does not consider the influences of the beam fluence maps. This results in a sub-optimal solution. Employing a machine learning method enables us to solve this large-scale problem efficiently. We show that we can maintain the quality of the treatment compared with the ones actually carried out in clinics. This problem is similar to the Traveling Salesman Problem with profits, therefore it can be represented as a graph. We utilized GNNs to capture the latent relational information among the underlying graph elements. This efficient algorithm reduces treatment time and allows centers to admit more patients at a given time window, which results in less waiting time for patients who are already in pain.

In the second article in Chapter 5, we utilized a similar approach and applied it to a problem arising in network optimization, the online Routing and Wavelength Assignment Problem. Unlike the first article in Chapter 4, some information is stochastic and not available to the RL agent. The online aspect of the problem where incoming traffic requests may arrive in the network following an unknown distribution would make the problem really difficult to be solved with exact algorithms. Unlike methods in the literature that solve routing and resource allocation sequentially, our proposed method simultaneously finds the beneficial combination in each decision point in time in order to maximize the expected overall number of admitted traffic requests. We formulated this problem under perfect information as a MILP and solved it with commercial solvers as a baseline.

In Chapter 6, similar to the previous one, we develop a GNN-based deep Q-learning algorithm to solve the online multi-resource constrained scheduling problem for a robotic production. As a result of the dynamic arrival of jobs, and the unknown values of processing time, solving this problem is challenging. To compare the results, we formulated the problem with Constraint Programming considering that we have perfect information on the stochastic parameters. We also solved the dynamic problem with three of the prevalent heuristics and compared the results of the proposed method against them.

Throughout this dissertation, we showed that the use of Deep Reinforcement Learning can benefit solving different Combinatorial Optimization problems that are difficult to be solved using conventional solution methods. Although the proposed methods do not reach the exact global optimal solution, they outperform other methods in terms of being efficient and achieving acceptable, near-to-optimal results.

CHAPTER 8 CONCLUSION AND RECOMMENDATIONS

In this dissertation, we studied the application of advanced machine learning methods such as Deep Reinforcement Learning and Graph Neural Networks in three different Combinatorial Optimization problems. In the following, we first present the contributions achieved by this dissertation. Then we highlight the limitations of the proposed solutions and possible future research directions.

8.1 Summary of Works

We addressed the simultaneous Beam Orientation Optimization and trajectory optimization for Cyberknife in the Chapter 4. Most of the previous studies consider one particular type of beam score, rank beams based on that criteria, and select a subset of the beams starting from the one with the highest score in a greedy fashion. These scores are either dosimetric-driven (related to the dose distribution of the selected beam) or geometric-driven (related to the physical and anatomical features of the patient). We, on the other hand, propose an algorithm based on DRL which enables us to construct reward signals based on different beam score approaches, either individual or universal scores. Latent information between beams can be captured by applying a GNN framework. By following a DQN method, we are capable of finding the best subset of the beams and the order to traverse them (trajectory) at the same time, such that the maximum expected cumulative reward is achieved while minimizing the treatment time. Although the training phase is time-consuming, once a model is trained, it solves new instances in a few seconds. The results on patients suffering from lung cancer demonstrate that shorter treatment time (35% reduction) is achieved and the quality of the treatment is maintained. The treatment time reduction is truly beneficial. It leads to a better treatment experience for individual patients and diminishes the discomfort and the side effects of inadvertent movements for them. Furthermore, it creates the opportunity to treat a higher number of patients in a given time period at the radiation therapy centers, which is a major hurdle in clinics.

As for the second essay in this dissertation described in Chapter 5, we apply a GNN-based DQN algorithm to the dynamic Routing and Wavelength Assignment problem. We generate Deep Reinforcement Learning agents that do not have prior expertise in the environment and have no information on the future incoming traffic connection admitted to the network. Most if not all of the previous studies focus on developing different methods for the routing part. They subsequently resort to choosing an acceptable simple heuristic (such as first-fit, least-used, or most-used wavelengths) for assigning a wavelength to the selected route. We propose an algorithm that finds the best route-wavelength combination (i.e., lightpath) at the same time. The status of the previously used wavelengths affects the selection of a route and vice-versa. We also considered the challenging dynamic traffic connection case, where the requests may arrive at arbitrary times and, if accepted, they are released after spending their corresponding service time in the network.

In the last essay in Chapter 6, we addressed a challenging multi-resource constrained scheduling problem arising from a real company where robotic arms and a human operator are used to move raw material to machines in order to process the jobs. To generate a solution method applicable to this practical problem, we captured and embedded practical constraints to the scheduling problem. A DRL algorithm based on deep Q-learning is adopted to tackle this problem. As this scheduling problem can be represented as a graph, we employed a GNN framework using an attention mechanism in the embedding phase. To simulate the practical case and similar to the second essay, we consider a dynamic case, where the orders arrive following a Poisson distribution. The agent has no information about the orders and the environment. We consider the case that jobs have an expiration time after which they may not be delivered to customers.

8.2 Limitations and Future Research

In the first article represented in Chapter 4, which discusses the application of Deep learning methods in Radiation Therapy treatment planning, the major limitation has been the lack of access to patient data. As discussed in the article as well, the results of the trained model were only applied to the three patients suffering from lung cancer. Although this small number is customary in publications in this field, having access to more data can always be illuminating in terms of the performance of the algorithm. This would not hinder the agent from learning a (near) optimal policy, but it is likely that could affect the discussion of the results. Therefore, we believe having access to a larger, more diverse data set with tumors in different structures would better explain the performance of the proposed algorithm. In addition, by assigning different weights in the reward construction step, one may achieve better solutions in terms of the different objectives to follow. Finally, a similar framework might be adapted to IMRT or possibly VMAT treatment planning to find the best beams or arcs, respectively.

In the Routing and Wavelength Assignment problem discussed in the second work described in Chapter 5, we assign the route and the wavelength at the same time. However, there are more different resources in the WDM networks, such as spectrum that could be considered. Considering that additional information could illustrate a more realistic situation. As discussed in the article and following the recommendation of the previous studies in the literature, we consider 4-*shortest-paths* between every node pair in the underlying network topology. Although it is demonstrated that following this rule would result in acceptable results in practical networks such as NSFNet or GEANT, we are inevitably restricting the solution space. Increasing this number entails more computational time and requires more infrastructure for the training purpose.

The third work presented in Chapter 6 is developed to solve a simplified version of a problem in a real company. We propose a solution for a dynamic multi-resource constrained scheduling problem. In this setting, a job requires one type of additional resource (either one type of robotic arm or a human operator) for processing. The dynamic nature of the problem makes it challenging to compare with the global optimal solutions. Therefore, we resort to heuristic baselines to evaluate the performance of the algorithm. We also formulated the problem under perfect information and solved it with CP to have a lower bound on the solutions. Future studies can focus on the cases where a limited buffer exists for the incoming orders, where an order is immediately rejected if at the decision time the buffer would be full. We may also consider cases where jobs are comprised of a number of operations, making the problem more realistic.

REFERENCES

[1] Richard M Karp. Reducibility among Combinatorial Problems BT - Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, an. pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi: 10.1007/978-1-4684-2001-2{_}9. URL `https://doi.org/10.1007/978-1-4684-2001-2_9`.

[2] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. 2 2021. URL `http://arxiv.org/abs/2102.09544`.

[3] L. J. Verhey. Comparison of three-dimensional conformal radiation therapy and intensity-modulated radiation therapy systems. *Seminars in Radiation Oncology*, 9 (1):78–98, 1 1999. ISSN 1053-4296. doi: 10.1016/S1053-4296(99)80056-3.

[4] P Francescon, W Kilby, J M Noll, L Masi, N Satariano, and S Russo. Monte Carlo simulated corrections for beam commissioning measurements with circular and MLC shaped fields on the CyberKnife M6 System: a study including diode, microchamber, point scintillator, and synthetic microdiamond detectors. *Physics in Medicine and Biology*, 62(3):1076–1095, 2 2017. ISSN 0031-9155. doi: 10.1088/1361-6560/aa5610. URL `https://iopscience.iop.org/article/10.1088/1361-6560/aa5610`.

[5] James L. Bedford, Peter Ziegenhein, Simeon Nill, and Uwe Oelfke. Beam selection for stereotactic ablative radiotherapy using Cyberknife with multileaf collimation. *Medical Engineering & Physics*, 64:28–36, 2 2019. ISSN 1350-4533. doi: 10.1016/J.MEDENGPHY.2018.12.011.

[6] Linda Rossi, Sebastiaan Breedveld, Ben J M Heijmen, Peter W J Voet, Nico Lanconelli, and Shafak Aluwini. On the beam direction search space in computerized non-coplanar beam angle optimization for IMRT—prostate SBRT. *Physics in Medicine and Biology*, 57(17):5441–5458, 9 2012. ISSN 0031-9155. doi: 10.1088/0031-9155/57/17/5441. URL `https://iopscience.iop.org/article/10.1088/0031-9155/57/17/5441`.

[7] Rentao Gu, Zeyuan Yang, and Yuefeng Ji. Machine learning for intelligent optical networks: A comprehensive survey. *Journal of Network and Computer Applications*, 157:

102576, 2020. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2020.102576. URL `https://www.sciencedirect.com/science/article/pii/S1084804520300503`.

[8] R Ramaswami and K N Sivarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking*, 3(5):489–500, 1995. ISSN 1558-2566. doi: 10.1109/90.469957.

[9] Yang Wang, Xiaojun Cao, and Yi Pan. A study of the routing and spectrum allocation in spectrum-sliced Elastic Optical Path networks. In *2011 Proceedings IEEE INFOCOM*, pages 1503–1511, 2011. ISBN 0743-166X. doi: 10.1109/INFCOM.2011.5934939.

[10] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 4 2021. ISSN 0377-2217. doi: 10.1016/J.EJOR.2020.07.063.

[11] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 1998. ISSN 1045-9227. doi: 10.1109/tnn.1998.712192.

[12] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 10 2021. ISSN 0305-0548. doi: 10.1016/J.COR.2021.105400.

[13] Yunhao Yang and Andrew Whinston. A Survey on Reinforcement Learning for Combinatorial Optimization. 8 2020. doi: 10.48550/arxiv.2008.12248. URL `http://arxiv.org/abs/2008.12248`.

[14] Christopher J C H Watkins and Peter Dayan. Q-Learning. *Machine Learning*, 8(3-4): 279–292, 1992. ISSN 0885-6125.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. 12 2013. doi: 10.48550/arxiv.1312.5602. URL `https://arxiv.org/abs/1312.5602`.

[16] 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. doi: 10.1109/TNN.2008.2005605.

[17] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep Reinforcement Learning meets Graph

Neural Networks: exploring a routing optimization use case. 10 2019. URL `http://arxiv.org/abs/1910.07421`.

[18] Yao Ma and Jiliang Tang. *Deep learning on graphs*. Cambridge University Press, 2021. ISBN 110893482X.

[19] Luis C. Lamb, Artur Garcez, Marco Gori, Marcelo Prates, Pedro Avelar, and Moshe Vardi. Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective. 2 2020. doi: 10.48550/arxiv.2003.00330. URL `http://arxiv.org/abs/2003.00330`.

[20] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. Interaction Networks for Learning about Objects, Relations and Physics. In D Lee, M Sugiyama, U Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/3147da8ab4a0437c15ef51a5cc7f2dc4-Paper.pdf`.

[21] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. ISBN 2640-3498.

[22] Dmitry B Kireev. ChemNet: a novel neural network based method for graph/property mapping. *Journal of chemical information and computer sciences*, 35(2):175–180, 1995. ISSN 0095-2338.

[23] Petar Veličković, Arantxa Casanova, Pietro Liò, Guillem Cucurull, Adriana Romero, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[24] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In I Guyon, U Von Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf`.

[25] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. 11 2016. URL `https://arxiv.org/abs/1611.08402`.

[26] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In D Lee, M Sugiyama, U Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf`.

[27] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 1 2021. ISSN 21622388. doi: 10.1109/TNNLS.2020.2978386.

[28] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. 11 2015. doi: 10.48550/arxiv.1511.05493. URL `https://arxiv.org/abs/1511.05493`.

[29] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. 9 2016. doi: 10.48550/arxiv.1609.02907. URL `http://arxiv.org/abs/1609.02907`.

[30] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In I Guyon, U Von Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf`.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. 10 2017. URL `https://arxiv.org/abs/1710.10903`.

[33] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 1 2020. ISSN 2666-6510. doi: 10.1016/J.AIOPEN.2021.01.001.

[34] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, 2015.

[35] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. 11 2016. URL `http://arxiv.org/abs/1611.09940`.

[36] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 2017.

[37] Hanjun Dai, Bo Dai, and Le Song. Discriminative Embeddings of Latent Variable Models for Structured Data. In Maria Florina Balcan and Kilian Q Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2702–2711, New York, New York, USA, 4 2016. PMLR. URL `https://proceedings.mlr.press/v48/daib16.html`.

[38] Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! 3 2018. URL `https://arxiv.org/abs/1803.08475`.

[39] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018. ISBN 9783319930305. doi: 10.1007/978-3-319-93031-2{_}12.

[40] Mohammadreza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V. Snyder. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, 2018.

[41] Antoine François, Quentin Cappart, and Louis-Martin Rousseau. How to Evaluate Machine Learning Approaches for Combinatorial Optimization: Application to the Travelling Salesman Problem. 9 2019. doi: 10.48550/arxiv.1909.13121. URL `https://arxiv.org/abs/1909.13121`.

[42] G K Bahr, J G Kereiakes, H Horwitz, R Finney, J Galvin, and K Goode. The Method of Linear Programming Applied to Radiation Treatment Planning. *Radiology*, 91(4): 686–693, 10 1968. ISSN 0033-8419. doi: 10.1148/91.4.686. URL `https://doi.org/10.1148/91.4.686`.

[43] Chieh Hsiu Jason Lee, Dionne M. Aleman, and Michael B. Sharpe. A fast beam orientation optimization method that enforces geometric constraints in IMRT for total marrow irradiation. *International Transactions in Operational Research*, 2015. ISSN 14753995. doi: 10.1111/itor.12093.

[44] Sifeng Lin, Gino J. Lim, and Jonathan F. Bard. Benders decomposition and an IP-based heuristic for selecting IMRT treatment beam angles. *European Journal of Operational Research*, 251(3):715–726, 6 2016. ISSN 03772217. doi: 10.1016/j.ejor.2015.12.050.

[45] Humberto Rocha, Joana Matos Dias, Tiago Ventura, Brígida da Costa Ferreira, and Maria do Carmo Lopes. Beam angle optimization in IMRT: are we really optimizing what matters? *International Transactions in Operational Research*, 2019. ISSN 14753995. doi: 10.1111/itor.12587.

[46] Mark Bangert and Jan Unkelbach. Accelerated iterative beam angle selection in IMRT. *Medical Physics*, 2016. ISSN 00942405. doi: 10.1118/1.4940350.

[47] Joel Mullins, Marc-André Renaud, Monica Serban, and Jan Seuntjens. Simultaneous trajectory generation and volumetric modulated arc therapy optimization. *Medical Physics*, 47(7):3078–3090, 7 2020. ISSN 0094-2405. doi: 10.1002/mp.14155. URL `https://doi.org/10.1002/mp.14155`.

[48] A. B. Pugachev, A. L. Boyer, and L. Xing. Beam orientation optimization in intensity-modulated radiation treatment planning. *Medical Physics*, 2000. ISSN 00942405. doi: 10.1118/1.599001.

[49] Yong-Jie Li. Prior Knowledge Helps Improve Beam Angle Optimization Efficiency in Radiotherapy Planning. 11 2018. doi: 10.48550/arxiv.1811.01834. URL `https://arxiv.org/abs/1811.01834`.

[50] Andrei Pugachev and Lei Xing. Pseudo beam's-eye-view as applied to beam orientation selection in intensity-modulated radiation therapy. *International Journal of Radiation Oncology Biology Physics*, 2001. ISSN 03603016. doi: 10.1016/S0360-3016(01)01736-9.

[51] Lulin Yuan, Wei Zhu, Yaorong Ge, Yuliang Jiang, Yang Sheng, Fang Fang Yin, and Q. Jackie Wu. Lung IMRT planning with automatic determination of beam angle configurations. *Physics in Medicine and Biology*, 2018. ISSN 13616560. doi: 10.1088/1361-6560/aac8b4.

[52] Philippe Meyer, Vincent Noblet, Christophe Mazzara, and Alex Lallement. Survey on deep learning for radiotherapy. *Computers in Biology and Medicine*, 98:126–146, 7 2018. ISSN 0010-4825. doi: 10.1016/J.COMPBIOMED.2018.05.018.

[53] Huan Wang, Peng Dong, Hongcheng Liu, and Lei Xing. Development of an autonomous treatment planning strategy for radiation therapy with effective use of population-based prior data. *Medical Physics*, 44(2):389–396, 2 2017. ISSN 0094-2405. doi: https://doi.org/10.1002/mp.12058. URL `https://doi.org/10.1002/mp.12058`.

[54] Dan Nguyen, Troy Long, Xun Jia, Weiguo Lu, Xuejun Gu, Zohaib Iqbal, and Steve Jiang. Dose prediction with U-net: a feasibility study for predicting dose distributions from contours using deep learning on prostate IMRT patients. *arXiv preprint arXiv:1709.09233*, 2017.

[55] Guy Amit, Thomas G. Purdie, Alex Levinshtein, Andrew J. Hope, Patricia Lindsay, Andrea Marshall, David A. Jaffray, and Vladimir Pekar. Automatic learning-based beam angle selection for thoracic IMRT. *Medical Physics*, 2015. ISSN 00942405. doi: 10.1118/1.4908000.

[56] Peng Dong, Hongcheng Liu, and Lei Xing. Monte Carlo tree search -based non-coplanar trajectory design for station parameter optimized radiation therapy (SPORT). *Physics in Medicine & Biology*, 63(13):135014, 7 2018. ISSN 1361-6560. doi: 10.1088/1361-6560/aaca17. URL `https://iopscience.iop.org/article/10.1088/1361-6560/aaca17`.

[57] Gregory Smyth, Jeffrey C. Bamber, Philip M. Evans, and James L. Bedford. Trajectory optimization for dynamic couch rotation during volumetric modulated arc radiotherapy. *Physics in Medicine and Biology*, 2013. ISSN 13616560. doi: 10.1088/0031-9155/58/22/8163.

[58] Gregory Smyth, Philip M. Evans, Jeffrey C. Bamber, Henry C. Mandeville, Liam C. Welsh, Frank H. Saran, and James L. Bedford. Non-coplanar trajectories to improve organ at risk sparing in volumetric modulated arc therapy for primary brain tumors. *Radiotherapy and Oncology*, 2016. ISSN 18790887. doi: 10.1016/j.radonc.2016.07.014.

[59] Vasant Kearney, Joey P Cheung, Christopher McGuinness, and Timothy D Solberg. CyberArc: a non-coplanar-arc optimization algorithm for CyberKnife. *Physics in Medicine & Biology*, 62(14):5777–5789, 6 2017. ISSN 1361-6560. doi: 10.1088/1361-6560/aa6f92. URL `https://iopscience.iop.org/article/10.1088/1361-6560/aa6f92`.

[60] Sebastian Troia, Alberto Rodriguez, Ignacio Martín, Jose Alberto Hernández, Oscar Gonzalez De Dios, Rudolfo Alvizu, Francesco Musumeci, and Guido Maier. Machine-Learning-Assisted Routing in SDN-Based Optical Networks. In *2018 European Conference on Optical Communication (ECOC)*, pages 1–3, 2018. doi: 10.1109/ECOC.2018.8535437.

[61] A Belbekkouche, A Hafid, and M Gendreau. A Reinforcement Learning-Based Deflection Routing Scheme for Buffer-Less OBS Networks. In *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*, pages 1–6, 2008. ISBN 1930-529X. doi: 10.1109/GLOCOM.2008.ECP.500.

[62] Y V Kiran, T Venkatesh, and C Siva Ram Murthy. A Reinforcement Learning Framework for Path Selection and Wavelength Selection in Optical Burst Switched Networks. *IEEE Journal on Selected Areas in Communications*, 25(9):18–26, 2007. doi: 10.1109/JSAC-OCN.2007.028806.

[63] Y V Kiran, T Venkatesh, and C S R Murthy. A Multi-Agent Reinforcement Learning Approach to Path Selection in Optical Burst Switching Networks. In *2009 IEEE International Conference on Communications*, pages 1–5, 2009. ISBN 1938-1883. doi: 10.1109/ICC.2009.5198632.

[64] Xiaoliang Chen, Baojia Li, Roberto Proietti, Hongbo Lu, Zuqing Zhu, and S J Ben Yoo. DeepRMSA: A Deep Reinforcement Learning Framework for Routing, Modulation and Spectrum Assignment in Elastic Optical Networks. *J. Lightwave Technol.*, 37(16): 4155–4163, 8 2019. URL `http://jlt.osa.org/abstract.cfm?URI=jlt-37-16-4155`.

[65] Zhiyuan Xu, Jian Tang, Jingsung Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven Networking: A Deep Reinforcement Learning based Approach. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1871–1879, 2018. doi: 10.1109/INFOCOM.2018.8485853.

[66] Yvan Pointurier and Fariba Heidari. Reinforcement learning based routing in all-optical networks with physical impairments. In *2007 Fourth International Conference on Broadband Communications, Networks and Systems (BROADNETS '07)*, pages 928–930, 2007. doi: 10.1109/BROADNETS.2007.4550535.

[67] Krzysztof Rusek, Jose Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020. ISSN 1558-0008 VO - 38. doi: 10.1109/JSAC.2020.3000405.

[68] Avinash Swaminathan, Mridul Chaba, Deepak Kumar Sharma, and Uttam Ghosh. GraphNET: Graph Neural Networks for routing optimization in Software Defined Networks. *Computer Communications*, 178:169–182, 2021. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2021.07.025. URL `https://www.sciencedirect.com/science/article/pii/S0140366421002887`.

[69] Sridhar Mahadevan and Georgios Theocharous. Optimizing Production Manufacturing Using Reinforcement Learning. In *FLAIRS conference*, volume 372, page 377, 1998.

[70] 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. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. 12 2017. doi: 10.48550/arxiv.1712.01815. URL `http://arxiv.org/abs/1712.01815`.

[71] Yufei Ye, Xiaoqin Ren, Jin Wang, Lingxiao Xu, Wenxia Guo, Wenqiang Huang, and Wenhong Tian. A New Approach for Resource Scheduling with Deep Reinforcement Learning. 6 2018. doi: 10.48550/arxiv.1806.08122. URL `https://arxiv.org/abs/1806.08122`.

[72] Peng Cheng Luo, Huan Qian Xiong, Bo Wen Zhang, Jie Yang Peng, and Zhao Feng Xiong. Multi-resource constrained dynamic workshop scheduling based on proximal policy optimisation. *International Journal of Production Research*, pages 1–19, 9 2021. ISSN 0020-7543. doi: 10.1080/00207543.2021.1975057. URL `https://doi.org/10.1080/00207543.2021.1975057`.

[73] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pages 50–56, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450346610. doi: 10.1145/3005745.3005750. URL `https://doi.org/10.1145/3005745.3005750`.

[74] M. Emin Aydin and Ercan Öztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2-3):169–178, 11 2000. ISSN 0921-8890. doi: 10.1016/S0921-8890(00)00087-7.

[75] Yi Chi Wang and John M. Usher. Application of reinforcement learning for agent-based production scheduling. *Engineering Applications of Artificial Intelligence*, 18(1):73–82, 2 2005. ISSN 0952-1976. doi: 10.1016/J.ENGAPPAI.2004.08.018.

[76] Zhicong Zhang, Li Zheng, Na Li, Weiping Wang, Shouyan Zhong, and Kaishun Hu. Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning. *Computers & Operations Research*, 39(7):1315–1324, 7 2012. ISSN 0305-0548. doi: 10.1016/J.COR.2011.07.019.

[77] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. 7 2017. doi: 10.48550/arxiv.1707.06347. URL `https://arxiv.org/abs/1707.06347`.

[78] Laura Masi, Margherita Zani, Raffaela Doro, Silvia Calusi, Vanessa Di Cataldo, Ivano Bonucci, Samantha Cipressi, Giulio Francolini, Pierluigi Bonomo, and Lorenzo Livi. CyberKnife MLC-based treatment planning for abdominal and pelvic SBRT: Analysis of multiple dosimetric parameters, overall scoring index and clinical scoring. *Physica Medica*, 2018. ISSN 1724191X. doi: 10.1016/j.ejmp.2018.11.012.

[79] H. Edwin Romeijn, Ravindra K. Ahuja, James F. Dempsey, and Arvind Kumar. A column generation approach to radiation therapy treatment planning using aperture modulation. *SIAM Journal on Optimization*, 2005. ISSN 10526234. doi: 10.1137/040606612.

[80] Ruijie Yang, Jianrong Dai, Yong Yang, and Yimin Hu. Beam orientation optimization for intensity-modulated radiation therapy using mixed integer programming. In *IFMBE Proceedings*, 2007.

[81] Mark Bangert and Uwe Oelfke. Spherical cluster analysis for beam angle optimization in intensity-modulated radiation therapy treatment planning. *Physics in Medicine and Biology*, 2010. ISSN 00319155. doi: 10.1088/0031-9155/55/19/025.

[82] R. Lee MacDonald and Christopher G. Thomas. Dynamic trajectory-based couch motion for improvement of radiation therapy trajectories in cranial SRT. *Medical Physics*, 2015. ISSN 00942405. doi: 10.1118/1.4917165.

[83] Peng Dong, Percy Lee, Dan Ruan, Troy Long, Edwin Romeijn, Yingli Yang, Daniel Low, Patrick Kupelian, and Ke Sheng. 4π non-coplanar liver SBRT: A novel delivery technique. *International Journal of Radiation Oncology Biology Physics*, 2013. ISSN 03603016. doi: 10.1016/j.ijrobp.2012.09.028.

[84] Marco Langhans, Jan Unkelbach, Thomas Bortfeld, and David Craft. Optimizing highly noncoplanar VMAT trajectories: The NoVo method. *Physics in Medicine and Biology*, 2018. ISSN 13616560. doi: 10.1088/1361-6560/aaa36d.

[85] Qihui Lyu, Victoria Y. Yu, Dan Ruan, Ryan Neph, Daniel O'Connor, and Ke Sheng. A novel optimization framework for VMAT with dynamic gantry couch rotation. *Physics in Medicine and Biology*, 2018. ISSN 13616560. doi: 10.1088/1361-6560/aac704.

[86] Marc André Renaud, Monica Serban, and Jan Seuntjens. On mixed electron-photon radiation therapy optimization using the column generation approach. *Medical Physics*, 2017. ISSN 00942405. doi: 10.1002/mp.12338.

[87] James L. Bedford, Simeon Nill, and Uwe Oelfke. Dosimetric accuracy of delivering SBRT using dynamic arcs on Cyberknife. *Medical Physics*, 47(4):1533–1544, 2020. ISSN 00942405. doi: 10.1002/mp.14090. URL https://aapm.onlinelibrary.wiley.com/doi/abs/10.1002/mp.14090.

[88] Richard Bellman. Dynamic programming and Lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

[89] Martin Riedmiller. Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In João Gama, Rui Camacho, Pavel B Brazdil, Alípio Mário Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005*, pages 317–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31692-3.

[90] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[91] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. 11 2015. URL https://arxiv.org/abs/1511.05952.

[92] Quentin Cappart, Emmanuel Goutierre, David Bergman, and Louis-Martin Rousseau. Improving Optimization Bounds Using Machine Learning: Decision Diagrams Meet Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33011443.

[93] Ignacio Martin, Sebastian Troia, Jose Alberto Hernandez, Alberto Rodriguez, Francesco Musumeci, Guido Maier, Rodolfo Alvizu, and Oscar de Dios. Machine Learning-Based Routing and Wavelength Assignment in Software-Defined Optical Networks. *IEEE Transactions on Network and Service Management*, 16(3):871–883, 2019. doi: 10.1109/TNSM.2019.2927867.

[94] Gurobi Optimization LLC. Gurobi Optimizer reference manual. *Www.Gurobi.Com*, 2019.

[95] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[96] Armando Pinto. Optical Networks: A Practical Perspective, 2nd Edition. *Journal of Optical Networking*, 1(6):219–220, 2002. URL `http://opg.optica.org/jon/abstract.cfm?URI=jon-1-6-219`.

[97] R Ramaswami and K N Sivarajan. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking*, 3(5):489–500, 1995. doi: 10.1109/90.469957.

[98] Hamed Pouya. *New Models and Algorithms in Telecommunication Networks.* PhD thesis, Concordia University, 11 2018. URL `https://spectrum.library.concordia.ca/id/eprint/984939/`.

[99] Maryam Daryalal and Merve Bodur. Stochastic RWA and Lightpath Rerouting in WDM Networks. 12 2020. URL `http://arxiv.org/abs/2012.10084`.

[100] Miriam Di Ianni. Efficient delay routing. *Theoretical Computer Science*, 196(1-2): 131–151, 4 1998. ISSN 0304-3975. doi: 10.1016/S0304-3975(97)00198-9.

[101] Xiang Zhou, Wei Lu, Long Gong, and Zuqing Zhu. Dynamic RMSA in elastic optical networks with an adaptive genetic algorithm. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 2912–2917, 2012. doi: 10.1109/GLOCOM.2012.6503559.

[102] Arturo Rodriguez, Washington Fernández, and Leonardo Ramírez. RWA: Novel Heuristic Algorithm for Optical Networks with Dynamic Traffic BT - Advanced Computer and Communication Engineering Technology. pages 1–10, Cham, 2016. Springer International Publishing. ISBN 978-3-319-24584-3.

[103] Hui Zang, Jason P Jue, and Biswanath Mukherjee. A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks. *Optical networks magazine*, 1(1):47–60, 2000.

[104] Brigitte Jaumard and Maryam Daryalal. Efficient Spectrum Utilization in Large Scale RWA Problems. *IEEE/ACM Transactions on Networking*, 25(2):1263–1278, 2017. doi: 10.1109/TNET.2016.2628838.

[105] Rajneesh Randhawa and J S Sohal. Blocking probability analysis of survivable routing in WDM optical networks with/without sparse-wavelength conversion. *Optik*, 121(5): 462–466, 2010. ISSN 0030-4026. doi: https://doi.org/10.1016/j.ijleo.2008.08.007. URL `https://www.sciencedirect.com/science/article/pii/S0030402608002222`.

[106] Jun Zhou and Xin Yuan. A study of dynamic routing and wavelength assignment with imprecise network state information. In *Proceedings. International Conference on Parallel Processing Workshop*, pages 207–213, 2002. ISBN 1530-2016 VO -. doi: 10.1109/ICPPW.2002.1039732.

[107] Hui Zang, J P Jue, L Sahasrabuddhe, R Ramamurthy, and B Mukherjee. Dynamic lightpath establishment in wavelength routed WDM networks. *IEEE Communications Magazine*, 39(9):100–108, 2001. ISSN 1558-1896 VO - 39. doi: 10.1109/35.948897.

[108] Gao Huimin, Yang Wen, Wang Liang, and Liu Zhengtang. Research on dynamic routing and wavelength assignment algorithm for optical networks. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 221–224, 2017. ISBN VO -. doi: 10.1109/IAEAC.2017.8054010.

[109] Svetoslav Duhovnikov, Dominic A Schupke, Gottfried Lehmann, Thomas Fischer, and Franz Rambach. Dynamic RWA for All Optical Networks Using Linear Constraints for Optical Path Feasibility Assessment. In *2006 European Conference on Optical Communications*, pages 1–2, 2006. ISBN 1550-381X VO -. doi: 10.1109/ECOC.2006.4801328.

[110] Neeraj Mohan and Priyanka Kaushal. Dynamic routing and wavelength assignment for efficient traffic grooming. *Journal of Optical Communications*, 2020. doi: doi: 10.1515/joc-2019-0309. URL `https://doi.org/10.1515/joc-2019-0309`.

[111] Xiaowen Chu, Bo Li, and Zhensheng Zhang. A dynamic RWA algorithm in a wavelength-routed all-optical network with wavelength converters. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, volume 3, pages 1795–1804, 2003. ISBN 0743-166X VO - 3. doi: 10.1109/INFCOM.2003.1209202.

[112] Brigitte Jaumard, Christophe Meyer, and Xiao Yu. When is wavelength conversion contributing to reducing the blocking rate? In *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.*, volume 4, pages 2078–2083, 2005. doi: 10.1109/GLOCOM.2005.1578031.

[113] Brigitte Jaumard, Christophe Meyer, and X Yu. How much wavelength conversion allows a reduction in the blocking rate? *J. Opt. Netw.*, 5(12):881–900, 12 2006. doi: 10.1364/JON.5.000881. URL `http://jon.osa.org/abstract.cfm?URI=jon-5-12-881`.

[114] José Suárez-Varela, Albert Mestres, Junlin Yu, Li Kuang, Haoyu Feng, Albert Cabellos-Aparicio, and Pere Barlet-Ros. Routing in optical transport networks with deep reinforcement learning. *J. Opt. Commun. Netw.*, 11(11):547–558, 11 2019. doi: 10.1364/JOCN.11.000547. URL `http://jocn.osa.org/abstract.cfm?URI=jocn-11-11-547`.

[115] R Amin, E Rojas, A Aqdus, S Ramzan, D Casillas-Perez, and J M Arco. A Survey on Machine Learning Techniques for Routing Optimization in SDN. *IEEE Access*, 9: 104582–104611, 2021. ISSN 2169-3536 VO - 9. doi: 10.1109/ACCESS.2021.3099092.

[116] 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 *Proceedings of the 2019 ACM Symposium on SDN Research*, SOSR '19, page 140–151, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367103. doi: 10.1145/3314148.3314357. URL `https://doi.org/10.1145/3314148.3314357`.

[117] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018. ISBN 0262352702.

[118] Linton C Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, 8 1977. ISSN 00380431. doi: 10.2307/3033543. URL `http://www.jstor.org/stable/3033543`.

[119] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[120] Kimberly C Claffy, Hans-Werner Braun, and George C Polyzos. Tracking long-term growth of the NSFNET. *Communications of the ACM*, 37(8):34–45, 1994. ISSN 0001-0782.

[121] Alexandre Dolgui, Dmitry Ivanov, Suresh P Sethi, and Boris Sokolov. Scheduling in production, supply chain and Industry 4.0 systems by optimal control: fundamentals, state-of-the-art and applications. *International Journal of Production Research*, 57 (2):411–432, 2019. doi: 10.1080/00207543.2018.1442948. URL `https://doi.org/10.1080/00207543.2018.1442948`.

[122] Kaizhou Gao, Zhiguang Cao, Le Zhang, Zhenghua Chen, Yuyan Han, and Quanke Pan. A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. *IEEE/CAA Journal of Automatica Sinica*, 6(4):904–916, 2019. doi: 10.1109/JAS.2019.1911540.

[123] Ziqiu Kang, Cagatay Catal, and Bedir Tekinerdogan. Machine learning applications in production lines: A systematic literature review. *Computers & Industrial Engineering*, 149:106773, 11 2020. ISSN 0360-8352. doi: 10.1016/J.CIE.2020.106773.

[124] Haejoong Kim, Dae-Eun Lim, and Sangmin Lee. Deep Learning-Based Dynamic Scheduling for Semiconductor Manufacturing With High Uncertainty of Automated Material Handling System Capability. *IEEE Transactions on Semiconductor Manufacturing*, 33(1):13–22, 2020. doi: 10.1109/TSM.2020.2965293.

[125] Yun Geon Kim, Seokgi Lee, Jiyeon Son, Heechul Bae, and Byung Do Chung. Multi-agent system and reinforcement learning approach for distributed intelligence in a flexible smart manufacturing system. *Journal of Manufacturing Systems*, 57:440–450, 2020. ISSN 0278-6125. doi: https://doi.org/10.1016/j.jmsy.2020.11.004. URL `https://www.sciencedirect.com/science/article/pii/S0278612520301916`.

[126] Shengkai Chen, Shuiliang Fang, and Renzhong Tang. A reinforcement learning based approach for multi-projects scheduling in cloud manufacturing. *International Journal of Production Research*, 57(10):3080–3098, 2019. doi: 10.1080/00207543.2018.1535205. URL `https://doi.org/10.1080/00207543.2018.1535205`.

[127] Bingran Li, Hui Zhang, Peiqing Ye, and Jinsong Wang. Trajectory smoothing method using reinforcement learning for computer numerical control machine tools. *Robotics and Computer-Integrated Manufacturing*, 61:101847, 2 2020. ISSN 0736-5845. doi: 10.1016/J.RCIM.2019.101847.

[128] Marcel Panzer and Benedict Bender. Deep reinforcement learning in production systems: a systematic literature review. *International Journal of Production Research*, pages 1–26, 9 2021. ISSN 0020-7543. doi: 10.1080/00207543.2021.1973138. URL `https://doi.org/10.1080/00207543.2021.1973138`.

[129] Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

[130] Longfei Zhou, Lin Zhang, and Berthold K.P. Horn. Deep reinforcement learning-based dynamic scheduling in smart manufacturing. *Procedia CIRP*, 93:383–388, 1 2020. ISSN 2212-8271. doi: 10.1016/J.PROCIR.2020.05.163.

[131] Chen Xin Wu, Min Hui Liao, Mumtaz Karatas, Sheng Yong Chen, and Yu Jun Zheng. Real-time neural network scheduling of emergency medical mask production during COVID-19. *Applied Soft Computing*, 97:106790, 12 2020. ISSN 1568-4946. doi: 10.1016/J.ASOC.2020.106790.

[132] Jorge A. Palombarini and Ernesto C. Martinez. Closed-loop Rescheduling using Deep Reinforcement Learning. *IFAC-PapersOnLine*, 52(1):231–236, 1 2019. ISSN 2405-8963. doi: 10.1016/J.IFACOL.2019.06.067.

[133] Christian D. Hubbs, Can Li, Nikolaos V. Sahinidis, Ignacio E. Grossmann, and John M. Wassick. A deep reinforcement learning approach for chemical production scheduling. *Computers & Chemical Engineering*, 141:106982, 10 2020. ISSN 0098-1354. doi: 10.1016/J.COMPCHEMENG.2020.106982.

[134] Jinling Leng, Chun Jin, Alexander Vogl, and Huiyu Liu. Deep reinforcement learning for a color-batching resequencing problem. *Journal of Manufacturing Systems*, 56: 175–187, 7 2020. ISSN 0278-6125. doi: 10.1016/J.JMSY.2020.06.001.

[135] Seunghoon Lee, Yongju Cho, and Young H Lee. Injection Mold Production Sustainable Scheduling Using Deep Reinforcement Learning, 2020.

[136] Chun-Cheng Lin, Der-Jiunn Deng, Yen-Ling Chih, and Hsin-Ting Chiu. Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network. *IEEE Transactions on Industrial Informatics*, 15(7):4276–4284, 2019. ISSN 1941-0050 VO - 15. doi: 10.1109/TII.2019.2908210.

[137] In-Boem Park, Jaeseok Huh, Joonkyung Kim, and Jonghun Park. A Reinforcement Learning Approach to Robust Scheduling of Semiconductor Manufacturing Facilities. *IEEE Transactions on Automation Science and Engineering*, 17(3):1420–1431, 2020. ISSN 1558-3783 VO - 17. doi: 10.1109/TASE.2019.2956762.

[138] Junyoung Park, Jaehyeong Chun, Sang Hun Kim, Youngkook Kim, and Jinkyoo Park. Learning to schedule job-shop problems: representation and policy learning using

graph neural network and reinforcement learning. *International Journal of Production Research*, 59(11):3360–3377, 6 2021. ISSN 0020-7543. doi: 10.1080/00207543.2020.1870013. URL `https://doi.org/10.1080/00207543.2020.1870013`.

[139] Yingzi Wei and Mingyang Zhao. Composite rules selection using reinforcement learning for dynamic job-shop scheduling. In *IEEE Conference on Robotics, Automation and Mechatronics, 2004.*, volume 2, pages 1083–1088, 2004. doi: 10.1109/RAMECH.2004.1438070.

[140] Zhicong Zhang, Li Zheng, and Michael X Weng. Dynamic parallel machine scheduling with mean weighted tardiness objective by Q-Learning. *The International Journal of Advanced Manufacturing Technology*, 34(9):968–980, 2007. ISSN 1433-3015. doi: 10.1007/s00170-006-0662-8. URL `https://doi.org/10.1007/s00170-006-0662-8`.

[141] Natalia Vesselinova, Rebecca Steinert, Daniel F Perez-Ramirez, and Magnus Boman. Learning Combinatorial Optimization on Graphs: A Survey With Applications to Networking. *IEEE Access*, 8:120388–120416, 2020. ISSN 2169-3536 VO - 8. doi: 10.1109/ACCESS.2020.3004964.

[142] Peyman Kafaei, Quentin Cappart, Marc-Andre Renaud, Nicolas Chapados, and Louis-Martin Rousseau. Graph neural networks and deep reinforcement learning for simultaneous beam orientation and trajectory optimization of Cyberknife. *Physics in Medicine &amp; Biology*, 66(21):215002, 2021. ISSN 0031-9155. doi: 10.1088/1361-6560/ac2bb5. URL `http://dx.doi.org/10.1088/1361-6560/ac2bb5`.

[143] Michel Tokic and Günther Palm. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual conference on artificial intelligence*, pages 335–346. Springer, 2011.

[144] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, and Luca Antiga. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[145] Wayne E Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi: https://doi.org/10.1002/nav.3800030106. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030106`.

[146] Ari P.J. Vepsalainen and Thomas E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):1035–1047, 1987. ISSN 00251909. doi: 10.1287/MNSC.33.8.1035.

[147] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

APPENDIX A SUPPLEMENTS OF CHAPTER 3

A.1 Mathematical Programming Model

The Mathematical Programming model which is implemented in Gurobi is represented as follows in (A.1)-(A.7). Let $G = (V, E)$ be a graph where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices and E is a set of edges. V corresponds to the nodes at which the robotic arm delivered a beam toward the patient. An edge $e \in E$ represents the path the robotic arm traverses from one node to the other. Let a score f_i be associated to each node $v_i \in V$ and a distance d_e associated to each edge. A beam spread measure is denoted by $s_{ij} = K\,(1 - \cos\alpha_{ij})^{-1}$ for every pair of the nodes (beams) i and j. α_{ij} is the angle separation between the pair of beams [51]. Distance d_e, score f_i and spread measure s_{ij} corresponds to r_{dist}, r_{dose}, and r_{spread} described in Section 4.3.5.

We associate a binary variable x_e to every edge $e \in E$, equal to 1 if the edge is traversed and 0 otherwise. Another binary variable y_i is associated with every $v_i \in V$, equal to 1 if and only if the corresponding node is used in the solution.

$$\min \sum_{e \in E} d_e\, x_e + \sum_{v_i \in V} f_i\, y_i + \sum_{v_i \in V} \sum_{v_j \in V} s_{ij}\, y_i\, y_j \tag{A.1}$$

$$\sum_{e \in \delta(\{v_i\})} x_i = 2\, y_i \qquad v_i \in V \tag{A.2}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad S \subseteq V, 2 \leq |S| \leq |V| - 2 \tag{A.3}$$

$$\sum_{v_i \in V} y_i = \mathcal{N} \tag{A.4}$$

$$y_1 = 1 \tag{A.5}$$

$$x_e \in \{0, 1\} \qquad e \in E \tag{A.6}$$

$$y_i \in \{0, 1\} \qquad v_i \in V \tag{A.7}$$

Constraints (A.2) are the degree constraints. Constraints (A.3) eliminates subtours form the solutions. As discussed before, the number of selected nodes is illustrated by $\mathcal{N} = 25$ and enforced in constraint (A.4). The resting point of the robotic arm is denoted by node 1 as is imposed by constraint (A.5).

A.2 Tables and Figures

In this section, we present the DVH diagrams and the tables discussing the dose-volume parameters of different methods for patients 2 and 3.

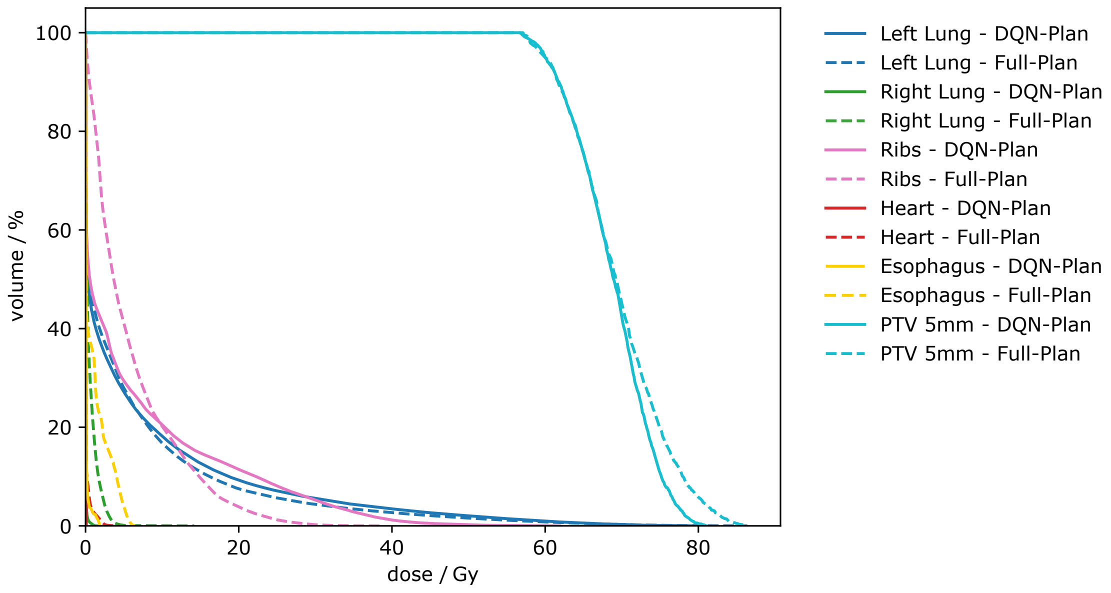


Figure A.1 Comparison between DVH diagrams for patient 2.

Table A.1 Dose-volume parameters for the target volume and critical structures for treatment plans generated by different method for patient 2.

Plan	**Structure**		**Statistics**	
DQN-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.00	58.60	**80.98**
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	**1.59**	**0.01**	0.00
	Left Lung	**80.98**	5.87	9.26
	Ribs	61.83	5.94	11.45
	Hearts	**0.06**	**0.00**	0.00
	Esophagus	**1.88**	0.07	0.00
Gurobi-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.00	58.50	84.00
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	6.94	0.76	0.00
	Left Lung	84.00	5.96	8.69
	Ribs	56.32	5.48	**0.00**
	Hearts	6.35	0.22	0.00
	Esophagus	5.26	0.88	0.00
Heuristic-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.00	58.09	84.35
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	7.72	0.65	0.00
	Left Lung	84.35	5.85	8.70
	Ribs	57.71	**5.53**	10.75
	Hearts	3.39	6.43	0.00
	Esophagus	9.43	1.27	0.00
Random-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.00	58.80	87.60
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	3.65	0.07	0.00
	Left Lung	87.60	6.45	9.47
	Ribs	58.27	5.62	10.46
	Hearts	4.04	0.02	0.00
	Esophagus	4.55	**0.62**	0.00
Full-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	60.00	**58.1**	86.32
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	14.15	0.54	**0.0**
	Left Lung	86.32	**5.46**	**7.52**
	Ribs	**38.92**	5.90	3.86
	Hearts	4.39	0.09	**0.00**
	Esophagus	6.47	1.05	**0.00**

Table A.2 Dose-volume parameters for the target volume and critical structures for treatment plans generated by different method for patient 3.

Plan	**Structure**		**Statistics**	
DQN-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	50.0	**43.0**	90.2
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	90.2	13.05	**20.1**
	Left Lung	**11.1**	**0.04**	**0.0**
	Ribs	86.9	6.2	14.7
	Hearts	57.4	3.1	4.9
Gurobi-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	50.0	45.5	**82.8**
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	**82.1**	12.1	23.7
	Left Lung	22.4	2.7	0.6
	Ribs	76.6	6.9	14.9
	Hearts	**44.4**	3.1	**4.7**
Heuristic-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	50.0	44.3	84.6
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	84.7	12.3	22.4
	Left Lung	34.6	2.6	5.9
	Ribs	**76.5**	**5.3**	9.9
	Hearts	54.4	4.8	10.8
Random-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	50.0	43.6	86.0
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	86.1	13.7	24.1
	Left Lung	43.2	3.3	7.5
	Ribs	82.2	6.2	11.2
	Hearts	69.4	7.7	15.7
Full-Plan		**Dose 95% (Gy)**	**Dose 98% (Gy)**	**D max (Gy)**
	Target	50.0	43.7	99.1
		D max (Gy)	**D mean (Gy)**	**V 20Gy (%)**
	Right Lung	99.1	**9.5**	**16.9**
	Left Lung	31.6	0.8	0.3
	Ribs	87.9	6.9	**11.0**
	Hearts	55.2	3.3	4.88

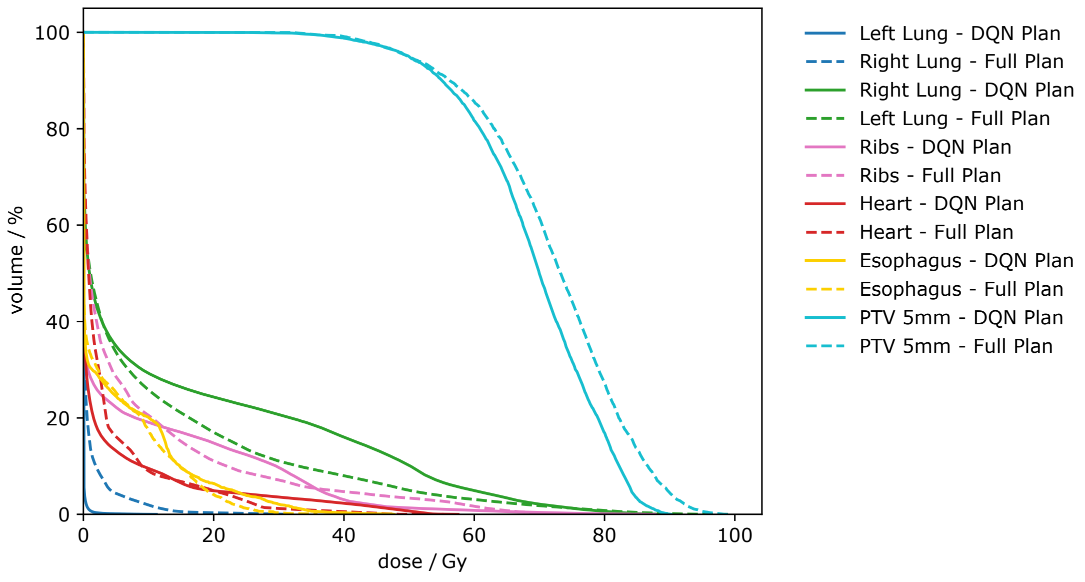


Figure A.2 Comparison between DVH diagrams for patient 3.

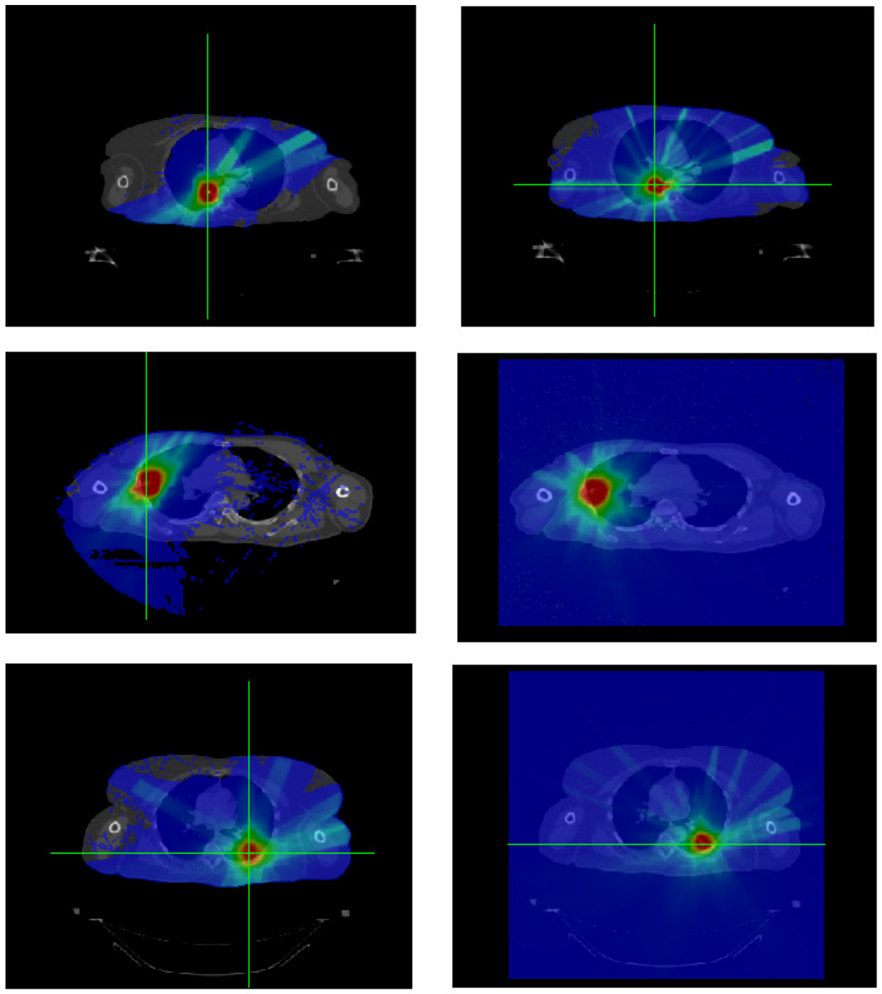


Figure A.3 Dose-wash images. Left column represents the DQN-plan and right column is the Clinical full-plan. Rows 1, 2, and 3 correspond to patients 1, 2, and 3, respectively.

APPENDIX B SUPPLEMENTS OF CHAPTER 4

B.1 Mathematical Programming Model under Perfect Information

In order to develop another baseline for the Dynamic RWA problem, we formulated it as a mathematical programming model. However, we considered a case where we have perfect information about the incoming traffic requests at the beginning. This means that for each traffic request, we know exactly when they will arrive and how long they will stay on the network. Solving this problem gives an upper-bound to the the case where our DRL agent solves the problem.

let $G = (V, E)$ be a graph generated from an underlying topology of the network. V is the set of nodes of the network. $\ell \in E$ denotes a fiber link of the network, joining a pair of nodes. Let D be the set of all incoming traffic requests, and Δ_d be the set of all possible lightpaths for a traffic request $d \in D$. We define a binary variable $x_{d\delta}, \forall d \in D, \forall \delta \in \Delta_d$ that is 1 if traffic request d is assigned to lightpath δ. In addition, a binary variable y_{dt} represents if request d is on the network at period $t \in \mathcal{T}$ where $\mathcal{T}$ is the set of all periods of the horizon. For each request d, we have complete information of the their arrival time arr_d, and duration dur_d. We define a parameter $q_{\ell\delta}$ equals to 1 if fiber link ℓ is assigned to lightpath δ and 0 otherwise. In addition, parameter $r_{\omega\delta}$ is 1 if wavelength $\omega \in \Omega$ is used in lightpath δ and 0 otherwise. The corresponding mathematical programming model for the described problem with perfect information on the requests is illustrated in B.1 to B.6:

$$\max \quad \sum_{d \in D} \sum_{\delta \in \Delta_d} x_{d\delta} \tag{B.1}$$

$$\sum_{\delta \in \Delta_d} x_{d\delta} \leq 1 \qquad \forall d \in D \tag{B.2}$$

$$dur_d \sum_{\delta \in \Delta_d} x_{d\delta} = \sum_{t \in \{arr_d, \ldots, arr_d + dur_d + 1\}} y_{dt} \qquad \forall d \in D \tag{B.3}$$

$$\sum_{d : arr_d \in \{t - dur_d, t\}} \sum_{\delta \in \Delta_d} q_{\ell\delta}\, r_{\omega\delta}\, x_{d\delta}\, y_{dt} \leq 1 \qquad \forall t \in \mathcal{T},\ \forall \ell \in E,\ \forall \omega \in \Omega \tag{B.4}$$

$$x_{d\delta} \in \{0, 1\} \qquad \forall d \in D,\ \forall \delta \in \Delta_d \tag{B.5}$$

$$y_{dt} \in \{0, 1\} \qquad \forall d \in D,\ \forall t \in \mathcal{T}. \tag{B.6}$$

Constraints B.2 ensures that at maximum one lightpath is selected for each incoming traffic request. As we consider a dynamic RWA, where the traffic requests are released after their service time is finished, we need to keep track of the times where the requests are utilizing the network resources. Constraints B.3 are thereby defined to this end. In Constraints B.4 we make sure that each fiber link and each wavelength is at maximum used in one lightpath for the requests currently on the network at each time period.

APPENDIX C SUPPLEMENTS OF CHAPTER 5

C.1 Constraint Programming model

We consider a variant of the problem presented in Chapter 6 where we have know all the characteristics of the problem in a deterministic way. In other words, all the information about the time of arrival and the processing time of the every job is completely known to the scheduler. We then formulated this problem with the *perfect information* as a Constraint Programming model. Table C.1 introduces the parameters and notations used.

Table C.1 Notations and parameters used in the CP model.

Notation	Description
J	Set of jobs
R	Set of resources
a_j	Arrival times of job j
s_j	Start times of job j
p_j	Processing times of job j
d_j	Delivery times of job j
δ_j	Expiration times of job j
e_j	Expiration costs of job j
w_j	Importance of job j
$rr_{r,j}$	Required resource of type r for job j
rc_r	Capacity of resource type r

C.1.1 Variables

Jobs (activities) are fundamental elements in the implementation of the scheduling problem in CP. The proposed model handles the following variables. $start\_times_j$ is the variable denoting the time where the processing on job j starts. $expired_j$ is a variable that represents the partial cost whether job j has passed its expiration time.

$$s_j : \text{start time of the job } j \tag{C.1}$$

$$e_j : \text{expiration penalty of job } j \tag{C.2}$$

C.1.2 Constraints

This section introduces the different constraints that comprise the CP formulation.

Start time constraints. Since the jobs arrive dynamically into the system, any job needs to be started after its arrival time.

$$s_j \;\geq\; a_j, \qquad \forall j \in J \tag{C.3}$$

Resource usage constraints. It is required that a set of jobs given by particular start times, processing times and resource requirements, never used more than a global resource capacity at any one time. We used the special *cumulative* global constraint to this end.

$$\textit{cumulative}\Big([\ s_j \mid j \in J\ ],\ [\ p_j \mid j \in J\ ],\ [\ rr_{r,j} \mid j \in J\ ],\ rc_r\Big), \quad \forall r \in R \tag{C.4}$$

C.1.3 Objective Function

As described in Chapter 6, we aim to minimize the total amount of penalty incurred as a result of tardiness and expiration of jobs following the equation 6.2.

$$\min\ \textit{objective} = \sum_{j \in J} w_j\ \max\Big(0,\ s_j + p_j - d_j\Big) + e_j \tag{C.5}$$

where

$$e_j = \begin{cases} \omega_j\ p_j, & \text{if } s_j + p_j > \delta_j \\ 0, & \text{otherwise.} \end{cases} \tag{C.6}$$