School of Business, Economics and Information Systems

Chair of Management Science / Operations and Supply Chain Management

Bachelor's thesis

Optimizing Allocation Location Problems: A Hybrid Approach with Graph Neural Networks and Deep Reinforcement Learning

Chairholder: Prof. Dr. Alena Otto

Supervising tutor: Amir Hossein Hosseini

Edited by: Lukas Bierling

Matriculation number: 88829

Course of studies: Bachelor of Science Wirtschaftsinformatik

Semester: 8

Address: Dorfstrasse 61a 82110 Germering

Telephone: +49 152/33896998

E-mail: bierling.lukas@gmail.com

Place, Date: Passau, 07.08.2024

Statutory Declaration

I herewith declare that I have composed the present thesis myself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references regarding the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance.

Place, Date	Signature

This thesis explores the optimization of dynamic Location Allocation (LA) problems with a focus on Replicated State Machines (RSM) in large-scale global networks. RSM systems, which ensure fault tolerance through the replication of state machines across multiple data centers, face significant latency challenges due to global client distribution and the need for dynamic reconfiguration of active data centers. Traditional heuristic approaches are limited in their scalability and ability to adapt to rapidly changing environments.

To address these challenges, this research proposes a hybrid optimization framework that leverages Graph Neural Networks (GNNs) and Deep Reinforcement Learning (DRL). GNNs are employed to capture the complex graph structure of RSM systems, representing data centers and client interactions. DRL is utilized to train the model in an online learning environment, enabling it to dynamically adjust data center configurations to minimize latency and operational costs.

The thesis first provides a comprehensive review of LA problems and their classification, followed by an exploration of classical and modern heuristic solution methods, including the theoretical foundation of DRL and GNNs. A novel application of the Dynamic Stochastic Facility Location Problem (DSFLP) is developed to frame the RSM optimization challenge within an RL context. The proposed solution is evaluated in a simulated environment, demonstrating its effectiveness in reducing latency and improving system performance compared to traditional methods.

This work contributes to the field of operations research by integrating advanced machine learning techniques with classical optimization problems, offering a scalable and adaptive solution for real-world applications in global network management.

Contents

List of Abbreviations and Symbols

DDFLP	Dynamic Deterministic Facility Location problem
DRL	Deep Reinforcement Learning
DSFLP	Dynamic Stochastic Facility Location problem
DQN	Deep Q-Networks Deep Q-Learning
FLP	Facility Location Problem
LA	Location Allocation
GAT	Graph Attention Networks
GCN	Graph Convolution Networks
GNN	Graph Neural networks
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization
RL	Reinforcement Learning
RSM	Replicated State Machines

List of Figures

List of Tables

1. Introduction

Location-Allocation (LA) problems are critical in many domains, including logistics, telecommunications, and urban planning. These problems involve determining the optimal placement of facilities and the allocation of resources to meet demand most efficiently. Over the years, various methods have been developed to solve LA problems, ranging from classical optimization techniques to advanced heuristic methods. However, as the complexity and scale of these problems have increased, traditional approaches have often fallen short in providing timely and accurate solutions.

In recent years, the advent of advanced computational techniques, particularly in the fields of machine learning and artificial intelligence, has opened new avenues for tackling complex optimization problems like LA. Reinforcement Learning (RL), with its ability to learn optimal policies through interaction with an environment, has shown promise in addressing dynamic and uncertain LA problems. Furthermore, Graph Neural Networks (GNNs), designed to work with data represented as graphs, offer a powerful tool for modeling the relationships between facilities, locations, and customers in LA problems.

This thesis explores the application of RL and GNNs to solve dynamic and stochastic LA problems. The focus is on extending traditional LA problem frameworks, such as the p-median and Facility Location Problems (FLP), to handle dynamic and uncertain environments through RL and GNN-based approaches. By integrating these modern techniques, the thesis aims to address the challenges of scalability and adaptability that are inherent in large-scale, real-world LA problems.

The research specifically investigates the potential of RL and GNNs to optimize the location and allocation of replicated state machines (RSM) in a distributed computing environment. This practical problem exemplifies a dynamic LA problem, where the objective is to minimize latency by strategically placing data centers in response to shifting client demands. The study not only contributes to the theoretical understanding of RL and GNN applications in LA problems but also provides practical insights and solutions for optimizing complex systems in real-time.

2. Allocation Location Problems: Literature Review

This section provides a comprehensive overview of the existing literature on Location-Allocation (LA) problems. It begins with a formal definition of the problem. The discussion then shifts to dynamic LA problems, which are the primary focus of this thesis, followed by a classification schema proposed by various authors. Then, the complexity of LA problems will be shown followed by a presentation of existing heuristic solution approaches. Finally, prevalent methods that use Reinforcement Learning (RL) and/or Graph Neural Networks (GNN) will be examined in the literature.

2.1. Problem Introduction

2.1.1. General Introduction

The LA problem involves the interplay between facilities, locations, and customers, deriving from foundational location problems explored by Location Science. The historical origins of Location Science trace back to the 17th century when Fermat tackled the geometric challenge of minimizing the sum of distances between three points in Euclidean space. However, contemporary Location Science primarily focuses on setups where the goal is to identify the optimal placement for one or more facilities to serve a set of demand points. The definition of "optimal" varies with the problem's context and objectives, such as minimizing distance costs or maximizing customer coverage (Laporte, Nickel, and Gama (2015)). This development was highly influenced by the single warehouse problem elaborated by Alfred Weber in the early 20th century (Weber (1922)) as pointed out by Turkoglu and Genevois (2020). It involves the determination of the minimum transport cost location in a continuous two-dimensional space in which the market and the sites of localized resources are given (Tellier (1972)).
Central to the LA problem is the task of determining the optimal placement and if needed also the number of facilities in available locations to minimize transportation costs to customers while fulfilling their demands (Azarmand and Neishabouri (2009a)). This problem extends beyond basic location problems by integrating the

allocation of customer demand, distinguishing it from simpler location scenarios. Some researchers equate LA problems with the p-median problem, where the objective is to position 'p' facilities so as to minimize the demand-weighted average distance to the nearest facility for demand nodes (Allahbakhsh et al. (2019), M. S. Daskin and Maass (2015)). The p-median problem thus shares considerable similarities with the classic Weber problem (Kazakovtsev (2013)).
However, other experts consider LA problems to encompass a wider range of challenges, including subfields such as, p-median problems, both capacitated and uncapacitated facility location problems (FLPs), p-center problems, and covering problems (M. S. Daskin and Maass (2015), Turkoglu and Genevois (2020)). Given this diversity, it is challenging to arrive at a concise definition of LA problems.
The focus of this thesis will be on the p-median and specifically on dynamic extensions of it. This focus is chosen because the p-median framework aligns closely with the methodologies discussed in this study and provides a practical approach to addressing LA challenges. For this thesis, the p-median framework stands for the general goals of the p-median problem formulation, i.e. minimizing some cost measure while serving demand points, instead of the concrete problem. It serves as the foundation for other extensions that all build up on the general goals of the p-median problem and can be developed out of it. By treating the p-median problem and extensions as synonymous with LA problems, the thesis aims to explore this specific aspect thoroughly, examining both its theoretical underpinnings and practical applications in optimizing facility locations. The following sections will give a mathematical formulation for the p-median problem and its derived extensions. Starting with the static p-median problem, a more relaxed and dynamic version of it will be developed in the following.

2.1.2. P-median Problem

As elucidated by C. ReVelle, Eiselt, and M. Daskin (2008), given potential facility locations $J = \{1, 2, ...\}$ and demand points $I = \{1, 2, ...\}$ the p-median problem involves choosing locations for p facilities based on given demands or weights w_i at nodes, distances d_{ij} between nodes and potential facility sites, and the number of facilities p. The goal is to optimize facility locations to minimize costs or maximize service effectiveness.

The decisions in the model are twofold:

1. **Location Decision**: Determine the optimal locations for the p facilities.
2. **Allocation Decision**: Assign each demand node to a specific facility, ensuring efficient service distribution.

The decision variables are:

- $x_j = 1$ if a facility is established at site j, and 0 otherwise.
- $y_{ij} = 1$ if demand node i is assigned to facility at site j, and 0 otherwise.

The mathematical formulation follows the approach of C. S. ReVelle and Swain (1970) and can be written as:

$$\text{minimize} \quad \sum_{j \in J} \sum_{i \in I} w_i d_{ij} y_{ij} \tag{1.1}$$

$$\text{subject to} \quad \sum_{j \in J} y_{ij} = 1 \quad \forall i \in I, \tag{1.2}$$

$$y_{ij} - x_j \leq 0 \quad \forall i \in I, \forall j \in J, \tag{1.3}$$

$$\sum_{j \in J} x_j = p, \tag{1.4}$$

$$x_j \in \{0, 1\} \quad \forall j \in J, \tag{1.5}$$

$$y_{ij} \in \{0, 1\} \quad \forall i \in I, \forall j \in J. \tag{1.6}$$

The constraints can be explained as:

- **Objective Function (1.1)**: Minimize the total weighted distance between facilities and assigned demand points.
- **Constraint (1.2)**: Ensure that each demand node i is assigned to exactly one facility j, ensuring comprehensive service coverage.
- **Constraint (1.3)**: Guarantee that assignments can only be made to operational facilities, integrating facility status directly into the decision-making process.
- **Constraint (1.4)**: Require that exactly p facilities are operational, aligning with strategic planning objectives.
- **Constraints (1.5) and (1.6)**: Define x_j and y_{ij} as binary variables, where $x_j = 1$ if a facility is established at location j and 0 otherwise, and $y_{ij} = 1$ if demand point i is served by facility j and 0 otherwise, reinforcing the discrete nature of facility location and service assignments.

In the literature, it is noted that Constraint (1.6) could be relaxed to non-negativity constraints. If an optimal solution results in a demand node being assigned to multiple facilities, it indicates equidistance. If the binary nature of the assignment

variables is required, a node can be fully assigned to any one of the facilities arbitrarily (C. ReVelle, Eiselt, and M. Daskin (2008)).
It is important to highlight that there are other, more specialized, formulations of the problem (Barbato et al. (2023), Blanco (2019), Avella, Sassano, and Vasil'ev (2007)). One relevant extension is presented in the following section.

2.1.3. Facility Location Problem

The *Facility Location Problem (FLP)* extends the traditional p-median problem by relaxing the assumption of having a fixed number of facilities. While the p-median problem primarily focuses on determining the optimal locations for a fixed number of facilities, the FLP also addresses the allocation of demand to these facilities. Again, in the FLP, two key decisions must be made:

1. **Location Decisions**: Determining the optimal sites for and number of facilities.

2. **Allocation Decisions**: Assigning user demand to the established facilities.

Each decision incurs costs: fixed costs f_j for setting up facilities and variable assignment costs for serving the demand from these facilities like in the p-median problem. Each facility j has a maximum capacity denoted by q_j. The objective is to minimize the total costs associated with both establishing facilities and allocating demand (Fernandez and Landete (2015)).

The mathematical formulation of the FLP is closely related to that of the p-median problem, and it is expressed as follows:

$$\text{minimize} \quad \sum_{j\in J}\sum_{i\in I} w_i d_{ij} y_{ij} + \sum_{j\in J} f_j x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j\in J} y_{ij} = 1 \quad \forall i \in I, \tag{2.2}$$

$$y_{ij} - x_j \leq 0 \quad \forall i \in I, \forall j \in J, \tag{2.3}$$

$$\sum_{i\in I} c_i y_{ij} \leq q_j x_j \quad \forall j \in J, \tag{2.4}$$

$$x_j \in \{0,1\} \quad \forall j \in J, \tag{2.5}$$

$$y_{ij} \in \{0,1\} \quad \forall i \in I, \forall j \in J. \tag{2.6}$$

The constraints can be explained as:

- **Objective Function (2.1)**: Minimize the total cost, which is the sum of the assignment costs ($\sum_{j \in J} \sum_{i \in I} w_i d_{ij} y_{ij}$) and the facility establishment costs ($\sum_{j \in J} f_j x_j$).
- **Constraint (2.2)**: Ensure that each demand point i is assigned to exactly one facility j.
- **Constraint (2.3)**: Guarantee that assignments can only be made to operational facilities.
- **Constraint (2.4)**: Ensure that the total demand assigned to a facility j does not exceed its capacity q_j.
- **Constraints (2.5) and (2.6)**: Define x_j and y_{ij} as binary variables, where $x_j = 1$ if a facility is established at location j and 0 otherwise, and $y_{ij} = 1$ if demand point i is served by facility j and 0 otherwise.

In essence, the FLP seeks to find the optimal balance between the costs of establishing facilities and the costs of allocating demand, thereby ensuring that all user demands are met efficiently and cost-effectively (Fernandez and Landete (2015)). In contrast to the p-median problem, one can see that constraint 1.4 is replaced with 2.4 accentuating the allocation, i.e. the variable facility selection in the FLP. If we opt to remove the constraint 2.4 we get a slightly modified version of the FLP, which is the *Uncapacitated FLP* (UFLP). In the UFLP setup, each facility does not have a capacity limit anymore.

2.1.4. Dynamic Deterministic Facility Location Problem

Both problems, the p-median and the FLP, cannot take into account dynamically changing environments. Problems that need to be evaluated over multiple time steps where in each time step the positions of facilities and clients changes require more sophisticated problem formulations and solution methods. As explained by An, Norouzi-Fard, and Svensson (2017) based on Eisenstat, Mathieu, and Schabanel (2014), the Dynamic Facility Location Problem tries to solve this downside. They introduce a relaxation to the classic FLP by minimizing the aggregated costs over a fixed time horizon of T time steps. Since the environment dynamics are known for this approach, i.e. the new locations of clients and facilities, we will refer to this problem as the *Dynamic Deterministic Facility Location Problem*

(DDFLP). The problem can be formally depicted as follows with $[T] = \{1, 2, ...T\}$:

$$\text{minimize} \quad \sum_{t\in[T]}(\sum_{j\in J}\sum_{i\in I} w_i^t d_{ij}^t y_{ij}^t + z_{ij}^t g + \sum_{j\in J} f_j^t x_j^t) \tag{3.1}$$

$$\text{subject to} \quad y_{ij}^t - x_j^t \leq 0 \quad \forall i \in I, \forall j \in J, \forall t \in [T]\,, \tag{3.2}$$

$$\sum_{i\in I} c_i^t y_{ij}^t \leq q_j^t x_j^t \quad \forall j \in J, \forall t \in [T]\,, \tag{3.3}$$

$$z_{ij}^t \geq x_{ij}^t - x_{ij}^{t+1} \quad \forall i \in I, \forall j \in J, \forall t \in [T] \setminus T-1, \tag{3.4}$$

$$x_j^t, y_{ij}^t \in \{0, 1\} \quad \forall i \in I, \forall j \in J, \forall j \in J, \forall t \in [T]\,, \tag{3.5}$$

$$z_{ij}^t \in \{0, 1\} \quad \forall i \in I, \forall j \in J, \forall t \in [T-1] \tag{3.6}$$

The constraints can be explained as follows:

- **Objective Function (3.1)**: Minimize the total cost over the entire time horizon $[T]$, which includes the assignment costs ($\sum_{j\in J}\sum_{i\in I} w_i^t d_{ij}^t y_{ij}^t$), relocation costs ($z_{ij}^t g$) with a fixed relocation cost variable g, and facility establishment costs ($\sum_{j\in J} f_j^t x_j^t$).
- **Constraint (3.2)**: Ensure that each demand point i at time t is assigned to an open facility j at time t.
- **Constraint (3.3)**: Ensure that the total demand assigned to a facility j at time t does not exceed its capacity q_j^t.
- **Constraint (3.4)**: Ensure the tracking of facility relocation costs by introducing binary variable z_{ij}^t that accounts for opening and closing facilities between consecutive time steps.
- **Constraints (3.5), (3.6), and (3.7)**: Define x_j^t, y_{ij}^t, and z_{ij}^t as binary variables, where $x_j^t = 1$ if a facility is established at location j at time t and 0 otherwise, $y_{ij}^t = 1$ if demand point i is served by facility j at time t and 0 otherwise, and $z_{ij}^t = 1$ if there is a change in the facility serving demand point i from time t to time $t + 1$ and 0 otherwise.

In summary, the DDFLP aims to achieve an optimal balance between the costs associated with establishing and relocating facilities and the costs involved in meeting demand across a dynamic environment. This method ensures that user demands are met efficiently and cost-effectively over multiple time periods, accounting for the changing locations of both facilities and clients (Eisenstat, Mathieu, and Schabanel (2014) and An, Norouzi-Fard, and Svensson (2017)).
It is crucial to note that if the relocation cost g is set to zero, the optimal facility locations x^t at each time step t will match the solution of the traditional FLP

described in section 1.3. By disregarding relocation costs, each time step can be treated independently, allowing for an optimal solution for each period without considering the impact of relocating facilities. This emphasizes the trade-off between choosing an optimal solution for a given time step and the costs of relocating too often.

2.1.5. Dynamic Stochastic Facility Location Problem

In many real-world applications, it is challenging to obtain upfront knowledge of environmental dynamics, meaning the positions of facilities and clients evolve unpredictably over time. The DDFLP does not capture these stochastic dynamics. To address this, researchers have proposed the *Dynamic Stochastic Facility Location Problem (DSFLP)*, as suggested by Eisenstat, Mathieu, and Schabanel (2014). Unlike the deterministic case, the stochastic nature of this problem means the locations of clients and sometimes also facilities at step t are unknown, making it impossible to model using the same approach.
The DSFLP uses Markovian approaches, as depicted by the work of Farahani, Abedian, and Sharahi (2009a) based on Rosenthal, J. A. White, and Young (1978). Formally, the problem includes the following components:

- X_t: Facility locations at time t (decision variable).
- A_t: Client locations at time t (stochastic variable).
- N: Set of possible locations for both facilities and clients, $1, ..., n$.
- F: Facility relocation cost matrix, $n \times n$.
- G: Service cost matrix, aggregating costs $w_i d_{ij}$ from the previous problem formulation, $n \times n$.
- P: Markov transition matrix for client locations, $n \times n$.
- B: Discount factor.

The process is as follows:

- The decision-maker observes (X_{t-1}, A_{t-1}) and chooses X_t.
- The relocation cost $F(X_{t-1}, X_t)$ is incurred.
- The probabilistic client locations A_t are realized.
- The service cost $G(X_t, A_t)$ is incurred.

This process repeats, and the goal is to find a policy for choosing facility locations to minimize the expected present worth of all costs:

$$\text{minimize} \quad \mathbb{E}\left[\sum_{t=1}^{\infty}\left[F(X_{t-1}, X_t) + G(X_t, A_t)\right] B^{t-1}\right] \tag{4}$$

The objective is to minimize the expected value of the discounted sum of relocation and service costs over time. This optimization considers the trade-off between selecting optimal facilities at each time step and the associated relocation costs, both immediate and future (discounted). The use of the expected value emphasizes the stochastic nature of this approach, accounting for uncertainty in the evolving environment. It's important to note that the literature does not provide a problem formulation in which the assignment of customers to facilities is explicitly modeled as part of the decision-making process. This omission is likely due to the stochastic nature of the problem, where the actual positions of clients at time step t are unknown, making it impractical to include constraints like 3.2 and 3.3 which rely on the concrete knowledge of position of clients at t.

In summary, the p-median problem and its extensions, such as the FLP, Dynamic DDFLP, and DSFLP, address the optimal placement and allocation of facilities to minimize costs and maximize service effectiveness. The p-median problem focuses on selecting p facilities to minimize weighted distances, while the FLP extends this by incorporating facility establishment costs and capacity constraints. The DDFLP further evolves to handle dynamic environments over multiple time steps, optimizing both establishment and relocation costs. The DSFLP introduces stochastic elements to account for uncertainty in client and facility locations over time. Each formulation offers unique insights and solutions to complex logistical challenges in varying contexts and undermines the development of relaxed versions of the classic p-median problem while following the general p-median framework.

2.2. Classification of LA problems

To provide more precise definitions within the field of LA problems, the literature presents various classification schemas. Although these schemas may show some differences in detail, they generally adhere to a similar framework. This discussion will primarily utilize and summarize the classification schemas provided by Turkoglu and Genevois (2020), "Introduction to Location Theory and Models" (2013), and Azarmand and Neishabouri (2009b) to offer a well-founded overview of the essential aspects of LA problems. These classifications will aid in understanding the various dimensions and approaches that characterize the study and

implementation of LA models.

1. Space: This refers to the underlying environment used to model the problem. According to Turkoglu and Genevois (2020), LA problems can be categorized into discrete and continuous spaces. In continuous spaces, the potential locations for facilities and demands can be anywhere on a plane, allowing for flexible modeling, such as positioning video cameras (C. ReVelle, Eiselt, and M. Daskin (2008)). The plane is mostly a one-dimensional or two-dimensional coordinate space. (Plastria (2001)). In contrast, in discrete spaces, both facilities and demands are restricted to a predefined and finite set of locations, as detailed by Turkoglu and Genevois (2020) and Plastria (2001). Azarmand and Neishabouri (2009b) introduces a third type, network space, where facilities and demands can be located anywhere along the edges of a network, with nodes merely marking the intersection of edges (Plastria (2001)). This type of space serves as a hybrid between continuous and discrete models, offering a limited continuous setting that is restricted to network edges. Azarmand and Neishabouri (2009b) further clarify that network-based problems may be either continuous or discrete, depending on whether the links are considered as a continuous range of potential locations or if only the nodes are considered suitable for facility placement. Plastria (2001) explain that the term of network problems is often used in a very restrictive sense, where only nodes of the networks are target sites. Therefore, these kind of network problems can be also framed as pure discrete problems and the network is only used to model distances. We will stick to that interpretation in this thesis.

2. Graph and Tree: Problems on networks can be distinguished between problems that occur on trees and others that have to be formulated more general on graphs. A tree is a network in which there is at most one path from any node to any other node. In other words, a tree is an acyclic graph or agraph with no cycles. Especially relevant to researchers in this field are minimum spanning trees where there is exactly one path between any node and any other node. If such a tree has N nodes, it will have N+1 edges. Tree problems are often easier solvable. However, while there are some real world problems that can be modelled as trees, e.g. parts of power transmission and telecommunication networks, complex network structures often require a general graph structure ("Introduction to Location Theory and Models" (2013)).

3. Number of Facilities: In single facility settings, only the location of one facility has to be determined. Conversely, in multi-facility problem the aim is to locate simultaneously more than one facility (Azarmand and Neishabouri (2009b)). Both can be assigned to the type of exogenous problems, because

the number is a priori fixed. Furthermore, the number facilities can also be variable, and be made an model output itself. Problems of this kind are called endogenous models (Turkoglu and Genevois (2020)). It is worth noting that single-facility problems are much easier to solve than multi-facility problems ("Introduction to Location Theory and Models" (2013)).

4. Static vs Dynamic: This classification focuses on the temporal aspect of the problem. Static problems, also known as single-period location problems, involve a scenario where the parameters remain constant throughout a single period. On the other hand, dynamic location problems, which can also be referred to as multi-period problems, involve multiple discrete time planning horizons, with parameters that change over these periods, as described by Turkoglu and Genevois (2020). Dynamic problems arise because of the more realistic environment they provide as pointed out by Plastria (2001): *"Existing competition will most probably alter its strategy when it loses part of or even all of its market share to a newcomer, implying that the competitive environment changes. This leads to dynamic models which aim at describing the action/reaction cycles of the competing actors."* It is important to mention, that in the realm of dynamic problems exist also deeper classification approaches. The authors there introduce refined categorization such as the cause of change, number of changes, source of change (demand site or facility site) or the time horizon (finite or infinite) (Farahani, Abedian, and Sharahi (2009b)).

5. Distance Metric: Depending on the space of the problem, different distance metrics have to be applied to compute distances between elements in the model. A very common technique for problems modelled as networks is to use the shortest distance between any pair of points ("Introduction to Location Theory and Models" (2013)). In discrete and continous spaces, as common distance metric like the Euclidean, Manhattan Distance or Chebyshev can be used (Turkoglu and Genevois (2020), "Introduction to Location Theory and Models" (2013)).

6. Deterministic vs Probabilistic: This classification differentiates how model inputs are handled based on certainty. In deterministic models, all inputs are assumed to be precisely known and remain constant, whereas in probabilistic models, inputs are uncertain and subject to variability. For instance, demand may fluctuate over time, requiring predictions that introduce uncertainty into the model (Azarmand and Neishabouri (2009a)). Probabilistic models may incorporate uncertainty through various methods; some integrate probability distributions into conventional mathematical frameworks, while others employ queuing models to handle these distributions (Owen and

M. S. Daskin (1998)). Typically in real-world applications, model inputs are based on forecasts and are inherently uncertain. Thus, models are categorized as deterministic if the inputs are considered or assumed certain, or probabilistic if the inputs are acknowledged as uncertain (Klose and Drexl (2005))

7. Capacitated vs Uncapacitated: A significant number of LA models, for example the p-median and FLP, operate under the assumption that facilities have unlimited capacity, classifying these as uncapacitated problems. Conversely, other facility location models account for limited capacity by restricting the allocation of demand to facilities, thus categorizing these as capacitated models (Klose and Drexl (2005)).

Some authors propose additional classification criteria for LA problems (Turkoglu and Genevois 2020; Azarmand and Neishabouri 2009a). However, the criteria outlined here offer a foundational orientation and allow us to frame our problems broadly within this classification framework. Other criteria, such as multi-objective settings, are either not relevant to our specific problem or are already encompassed to some extent by the existing criteria.

2.3. Solution Approaches

2.3.1. General Introduction

The p-median problem is well-known to be NP-hard, as first demonstrated by Kariv and Hakimi (1979). Extensions such as the static and dynamic FLP also exhibit NP-hardness, adding further complexity (Fernandez and Landete (2015)). Given this complexity, heuristic methods are often employed to find solutions. In the following sections, common heuristic approaches for solving the LA problem will be presented.

2.3.2. Classical Heuristic Approaches

According to Nenad Mladenovic et al. (2007) classical heuristics can be divided into three groups:

- **Constructive heuristics** (CH): These start with an empty solution and repeatedly extends the current solution until a complete solution is obtained (Salhi (1997))). Examples are greedy heuristics (Whitaker (1983)), stingy heuristics (Salhi and Atkinson (1995)), dual ascent heuristics (GalvÃ£o (1980)) and composite heuristics (Salhi (1997))

- **Local search** (LS): These move from one solution to another within the search space by making local changes until an optimal solution is found or a specified time limit is reached (Michiels, Aarts, and Korst (2018)). Examples are: alternate heuristics (Maranzana (1964)) and interchange heuristics (Hansen and N. Mladenovic (1997))
- **Mathematical programming** (MP): Combined with heuristics, they aim to explore how theoretical optimization methods can be applied practically to generate good, though not necessarily optimal, solutions to complex problems (Ball (2011)). Examples are: dynamic programming heuristics(Hribar and M. S. Daskin (1997)), Lagrangian relaxation heuristics (Beasley (1993)) and aggregate heuristics (Bowerman, Calamai, and Brent Hall (1999))

For a detailled explanation of every heuristic refer to Nenad Mladenovic et al. (2007).

2.3.3. Metaheuristics

Metaheuristics are iterative master processes that guide and modify subordinate heuristics to efficiently generate high-quality solutions, either by working on a single solution or a collection of solutions per iteration. These subordinate heuristics can range from complex algorithms to simple local searches or construction methods (Voß (2008)). The metaheuristic family encompasses various techniques, such as tabu search (Salhi (2002)), genetic search (Alp, Erkut, and Drezner (2003)), heuristic concentration (Rosing et al. (1998)) and more.
Although these metaheuristic techniques have proven effective, the rapid advancements in artificial intelligence have opened up new possibilities. Modern metaheuristics, particularly those based on neural networks (NNs), offer powerful alternatives for tackling complex optimization problems (Nenad Mladenovic et al. (2007)).

2.3.4. Modern Heuristics Based on Neural Networks

NNs are inspired by the biological neurons of the brain and consist of interconnected computational "neurons" arranged in layers. These networks use weighted signal channels (synaptic weights) to process inputs through a nonlinear activation function, allowing them to approximate arbitrary nonlinear functions. In recent years, artificial (multi-layer) neural network (NN) models have become crucial tools for solving complex computational problems, including complex multi-parameter problems (Lachhwani (2020)). As pointed out by Hornik, Stinchcombe, and H. White (1989), NNs are universal function approximators. This makes them especially suitable for operation research problems. They offer a heuristic alternative

(Burke and Ignizio (1992)) capable of deriving near-optimal solutions. NNs are usually calibrated using gradient based algorithms. They use a large amount of data to construct a multidimensional function that captures relations in that data. The model parameters are adjusted at every optimization step until convergence is reached. The most commonly used algorithm is called backpropagation, which *refers to the traditional manner in which learning proceeds in artificial neural networks, through which connection weights are calibrated to maximize training accuracy."* (Rohlfs (2023)). Refer to Figure 2.1 for a simplified visual depiction of layers in an NN. In operations research, for example, Dominguez Merino and Muñoz Perez (2002) introduced the application of a two-layer NN to approximately solve the p-median problem with fairly good results.

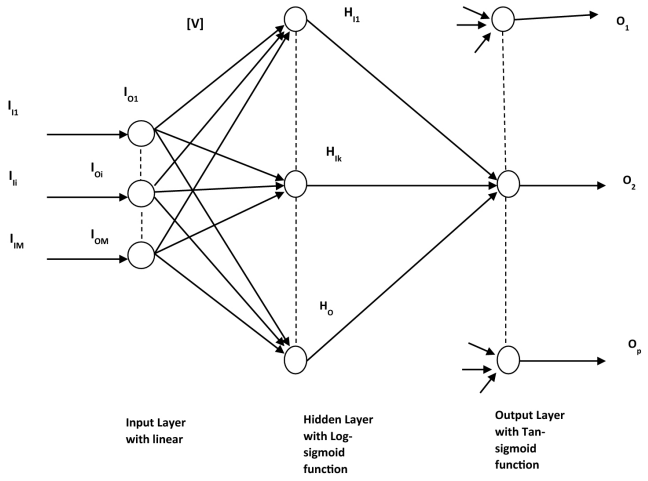


Figure 2.1.: The Figure illustrates the simplest architecture of a NN, a multi-layered feed-forward neural network. The input layer consists of neurons $I_{i1}, I_{i2}, \ldots, I_{iM}$ with linear activation functions. The hidden layer consists of neurons $H_{11}, H_{1k}, \ldots, H_0$ with log-sigmoid activation functions. The output layer consists of neurons $O_1, O_2, \ldots, O_p$ with tan-sigmoid activation functions. Each neuron in a layer is connected to every neuron in the subsequent layer, forming a fully connected network. By incorporating non-linear activation functions in the hidden and output layers, the network is capable of approximating non-linear functions, allowing it to model complex relationships (Lachhwani (2020)).

As we delve deeper into neural network-based heuristics, it is important to recognize the diversity of NN architectures that have been developed to address various data structures. Each of these architectures offers unique advantages and is suited

for different types of problems. Here, we will explore some of the most prominent NN architectures used in modern heuristics and are applied to the field of operations research:

- **Recurrent NNs (RNNs)**: While feed-forward NNs are well suited for numerical, table-organized data it is not able to capture autoregressive dependencies in the data. Therefore, RNNs were developed. They are able to process sequential data by constructing a latent representation of the dependence of the data that is propagated through the network (Caterini and Chang (2018)). Dominguez and Munoz (2008) used a RNN for the LA problem.

- **Convolutional NNs (CNNs)**: In contrast to feed-forward NNs, CNNs have a unique architecture where hidden layer neurons are only connected to a subset of neurons in the previous layer, creating sparse connectivity. This allows CNNs to learn features implicitly and efficiently handle spatial data. The deep architecture of CNNs results in hierarchical feature extraction, where early layers detect simple patterns such as edges or color blobs, intermediate layers identify shapes and object parts, and final layers recognize complete objects. This hierarchical approach makes CNNs particularly effective for image and video processing tasks (Aloysius and Geetha (2017)). For example, Matis and Tarabek (2023) have shown how CNNs can be used for LA problems.

- **Graph NNs (GNNs)**: Graphs are flexible mathematical objects that can represent many entities and knowledge from different domains. GNNs are mathematical models that can learn functions over graphs and are a leading approach for building predictive models on graph-structured data. This combination has enabled GNNs to advance the state of the art in many disciplines by effectively capturing relationships and dependencies in complex, non-Euclidean data structures (Corso et al. (2024)). They will be explained in a more detailed way later. Many problems in operations research are modeled as graphs. For example, the LA problem in the network space. Thus, GNNs offer a well-foundated method to use the approximative power of NNs, while exploiting the structural characteristics of graphs. For instance, C. Wang et al. (2023) and Liu, X. Yan, and Yaochu Jin (2023) make use of GNNs to solve the LA problem.

2.3.5. Modern Heuristics Based on Reinforcement Learning

While, in theory, NNs should be able to approximate functions arbitrarily well, they require a lot of data to converge, i.e. being trained. This data has to be

collected a-priori. Hence, the model is trained in an offline manner, where data is acquired before calibrating the model's parameter. This is called offline learning. It is inefficient in time and space costs and is hardly scalable for large-scale applications since the model has to be recalibrated from scratch when new data arrives (Hoi et al. (2021)). Additionally, data collection in operation research is traditionally difficult. For example, some institutions may not be interested in or are secretive about the disclosure of their data (Simpson, Genovese, and Rais Mohamad Mokhtar (2022)).

Conversely, online-learning methods do not need the data to be collected before. Algorithms using online learning use freshly received data at every time step to learn and update the best predictor for future time step. The model is updated instantly for every new data instance overcoming the drawbacks of offline learning (Hoi et al. (2021)).

While there exist a lot of algorithms that use online learning, we will focus on reinforcement learning (RL) in this thesis. RL involves an agent interacting with an environment through actions based on the state and reward signals, aiming to optimize a sequential decision-making policy for maximum cumulative rewards. Rewards or penalties are assigned based on the agent's actions, and RL algorithms iteratively update the agent's policy using these outcomes to improve future decisions. The theoretical framework of RL is built on *Markov Decision Processes* (MDPs). This means, that the conditional probability distribution of future states of a stochastic process has the Markov property, i.e. it depends only on the current state (Shakya, Pillai, and Chakrabarty (2023a)). The decision-making is modeled with defined states and actions, enabling the prediction and optimization of actions for the best possible results (Shakya, Pillai, and Chakrabarty (2023b)). This ongoing adaptability and iterative learning process align seamlessly with the principles of online learning, allowing RL to efficiently integrate into environments where data evolves or accumulates continuously.

Deep RL (DRL) integrates the advanced capabilities of deep learning architectures, like RNNs, CNNs, and GNNs, in the RL framework to model and utilize the underlying data in the most effective manner. These architectures enable DRL to capture complex dependencies and patterns in data, enhancing the agent's ability to make informed decisions based on learned experiences. By embedding these neural network models, DRL can effectively process and learn from a variety of data types and structures like sequential, spatial, or graph-based. This integration not only boosts the learning capacity of RL agents but also extends their applicability to a broader range of complex real-world problems, where adaptability and efficient data utilization are crucial (Arulkumaran et al. 2017).

RL has increasingly proven to be a valuable heuristic in operations research (Schneckenreither and Haeussler (2019), Wan, T. Li, and J. M. Wang (2023)) due to

its effectiveness in solving hard combinatorial optimization problems (Y. Yan et al. (2022)) and often providing faster, higher-quality solutions compared to standard heuristics, as highlighted by Q. Wang and Tang (2021). Another significant advantage of RL is its ability to model complex systems that are difficult to represent with traditional optimization models or heuristic search rules, by constructing simulated environments where the RL algorithm can operate (Y. Yan et al. (2022)). Furthermore, once trained, RL algorithms can deliver real-time solutions for operational systems in seconds (Y. Yan et al. (2022)), making them highly valuable for practical problems that require immediate deployment of solutions. Finally, RL is also already used specifically for LA problems (C. Wang et al. (2023), Guo, Xu, and Yaohui Jin (2023), Yu et al. (2021), Klar, Glatt, and Aurich (2021)).
Given these advantages, this thesis will investigate the potential of RL and DRL, enhanced by the capabilities of GNNs, for optimizing the LA problem. The next chapter will explore these topics in depth, demonstrating how the combination of RL and GNNs can effectively address the challenges in LA.

2.4. Research Gap and Contribution

The literature extensively explores, classifies, and provides traditional solution approaches to the LA problem. This is evident in the literature review table A.1 in *Table A.1* in the appendix A, where the majority of cited works are sourced from established operations research journals. These references primarily focus on well-established descriptions of LA problems and classic heuristic methods.
Consequently, the application of RL to LA problems is relatively underexplored. Most of the literature focussing on RL or GNNs for LA problems are either very recent (2020-2024)[1] or are often still preprints and not reviewed yet. Although some innovative approaches have emerged, they remain in their early stages. For instance, Yu et al. (2021) employ RL to optimize resource allocation in humanitarian logistics following a disaster. A more recent study by Guo, Xu, and Yaohui Jin (2023) integrates GNNs with RL to address the FLP. Their model aims to suggest optimal facility relocations to minimize transportation costs. +
Despite this lack of research and literature, there is a general trend, noted by C. Wang et al. (2023), of increasing academic and industrial interest in applying machine learning to traditional NP-hard problems. Deep learning's perceptual capabilities combined with RL's reasoning abilities can effectively tackle large-scale combinatorial optimization problems. This growing interest highlights a significant research gap: the need for more sophisticated and practical applications of DRL and GNNs in solving LA problems.
The primary contribution of this thesis is to address this gap by investigating

[1] See therefore also the *Table A.2* and *Figure A.1* in the appendix A

whether DRL in conjunction with GNNs can serve as a viable alternative to traditional heuristics for solving the traditional NP-hard LA problem. This research will focus on a real-world scenario involving *Replicated State Machines* (RSM), which can be conceptualized as a dynamic LA problem. The necessity for rapid decision-making in this context, along with its ease of simulation, presents an ideal opportunity for this investigation.
By developing a framework that incorporates DRL and GNNs, this thesis aims to demonstrate the potential of modern GNN-based DRL approaches to enhance traditional heuristics for LA problems. Specifically, this research will contribute by:

1. Building the theoretical foundation to apply RL algorithms on the DSFLP, establishing the path to apply them also on other LA problems

2. Proposing a DRL-based framework that leverages GNNs for solving dynamic LA problems.

3. Applying this framework to a practical, real-world problem involving RSM, to assess the framework's effectiveness in scenarios requiring quick decision-making and data generalization.

4. Building a reusable and adjustable simulation environment for the RSM system that can be reused for other RL algorithms as it implements common RL interfaces

These contributions will not only address the identified research gap but also offer a new perspective on the application of advanced machine learning techniques in operations research.

3. Reinforcement Learning

Generally speaking, RL involves an agent interacting with an environment E at time step t through actions a_t based on the environment state s_t and reward signals r_t, aiming to optimize a sequential decision-making policy π for maximum cumulative rewards. RL uses the formal framework of MDP. Refer to Figure 3.1 for a simplified visualization of this process. The goal of the agent is to maximize the expected discounted sum of rewards, i.e. the return G_t. Formally, the return G_t can be depicted as (Sutton and Barto (2018a)):

$$G_t = r_{t+1} + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ... = r_{t+1} + \gamma G_{t+1} \tag{5}$$

The state value function $v^\pi(s_t)$ and action value function $q^\pi(s_t, a_t)$ determine how advantageous it is for the agent to be in a particular state or to perform an action in that state. The value functions are useful in RL for estimating optimal policies. These value functions are primarily based on expected future rewards or returns. The expected return for a given policy π is $v^\pi(s_t)$ if the agent starts from state s_t and then follows the policy π. For a given policy, the state value function can be expressed as:

$$v^\pi(s) = \mathbb{E}_\pi \left[G_t \mid s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s \right] \forall s \in S \tag{6}$$

where S is the set of all possible states, i.e. the state space.
If an agent starts from the state s and performs an action a and follows the policy π thereafter, then the expected return is known as the action value function $q^\pi(s, a)$ and expressed as:

$$q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid s_t, a_t \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s, a \right] \forall s \in S, a \in A \tag{7}$$

where A is the set of all possible actions, i.e. the action space.
The state value $v^\pi(s)$ of a state would theoretically represent the average reward obtained by the agent for visiting it a large number of times. Similarly, the agent's average reward for repeatedly performing a fixed action a in a given state s would

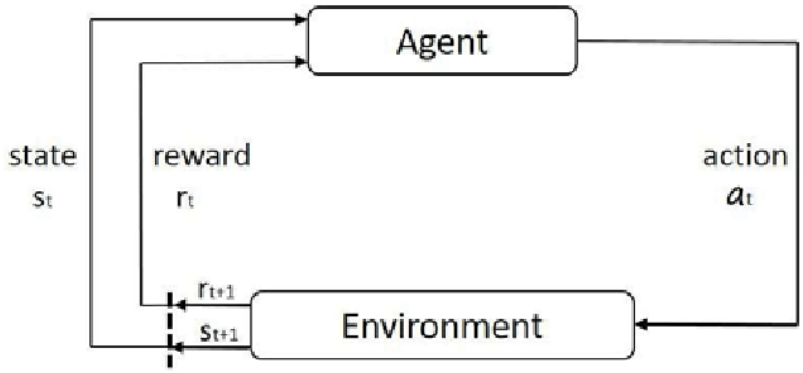


Figure 3.1.: RL cycle of an agent interacting with an environment (Shehab, Khader, and Alia (2019))

be the state's action value. Mathematically we can calculate the value functions using equations 6 and 7 by recursively unrolling them as :

$$v^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v^{\pi}(s') \right] \tag{8}$$

$$q^{\pi}(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \sum_{a} \pi(a'|s') q^{\pi}(s', a') \right] \tag{9}$$

Equations 8 and 9 are known as *Bellman Equations* which represent the recursive relationship between the value functions of current and successor states (Shakya, Pillai, and Chakrabarty (2023a)).
The optimal policy π^* can then be derived using optimal value functions $v^*(s)$ and $q^*(s, a)$ as follows (Sutton and Barto (2018a)):

$$v^*(s) = \max_{\pi} v^{\pi}(s) = \max_{a} \left(r + \gamma \sum_{s',r} p(s', r|s, a) v^*(s') \right) \tag{10}$$

$$q^*(s, a) = \max_{\pi} q^{\pi}(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \max_{a'} (q^*(s', a')) \right] \tag{11}$$

This forms the foundation of any RL algorithm. We will make special use of the state value function $v^{\pi}(s)$ later to derive the theoretical justification to apply RL on DSFLP.

4. Graph Neural Networks

4.1. Introduction

As previously discussed, there are various types of neural networks, each utilizing different data structures. Graph Neural Networks (GNNs) are a specific class of neural networks designed to handle data organized in graph structures (Khemani et al. (2024)). Let $G = (V, E)$ be a graph consisting of vertices V and edges E. Let $x_v \in \mathbb{R}^{d_V}$ and $x_e = x_{(u,v)} \in \mathbb{R}^{d_E}$ be the corresponding node and edge features (between node v and u) of dimension d_V and d_E. Node features are attributes or properties associated with each node in a graph, while edge features are attributes or properties associated with the connections (edges) between pairs of nodes. The core concept of GNNs is that nodes in a graph symbolize objects or concepts, while edges represent the relationships between them. Consequently, for each node $v \in V$, we can construct a hidden state $h_v \in \mathbb{R}^{d^{(h)}}$, that is based on the information contained in the neighborhood of v, denoted $N(v)$ with dimension $d^{(h)}$.
Formally speaking, let f_w be a function parametrized by w that expresses the dependence of a node v on its neighborhood $N(v)$. We then can depict $h_v^{(1)} \in \mathbb{R}^{d^{(1)}}$, the first hidden state of node v as (Scarselli et al. (2009)):

$$h_v^{(1)} = f_w(x_v, \{(x_{(v,u)}, x_u) \mid \forall u \in N(v)\} \tag{12}$$

where $d^{(1)}$ is the dimension of first the hidden state, x_v is the feature vector of node v and $(x_{(v,u)}, x_u)$ are the feature vectors of the edge between node v and u and the node u itself for all nodes in the neighborhood of v, i.e. $N(v)$. As shown, the hidden state $h_v^{(1)}$ uses the edge and node feature information of neighboring nodes to derive the new representation for the node v. Performing this transformation for every node in the graph results in a new graph representation G', where the node features x_v are replaced by $h_v^{(1)}$ at each node. This new graph G' can then be used for further processing. For example, one could train a classifier on top of each node representation $h_v^{(1)}$ predicting classes for each node or construct another hidden state $h_v^{(2)}$ based on $h_v^{(1)}$ instead of x_v.

4.2. Message Passing

While introducing the general idea of GNNs, we used a function f_w that should capture the relationship between nodes using node and edge feature information. The concrete design of this function will be the content of this section. The core principle when constructing such a function f_w is called *message passing*. Analogously to before, we take a graph $G = (V, E)$ with node and edge features x_v, x_e to create hidden states or node embedding $h_v^{(l)} \in \mathbb{R}^{d^{(l)}}$, where $h_v^{(l)}$ is the hidden state of node v after performing l iterations of the message passing having the dimension $d^{(l)}$ (Khemani et al. (2024)). If neural networks are used at any step during this process, we refer to this as *neural message passing* (Hamilton (2020a)). Stating $x_v = h_v^0$ and $l > 0$ the message-passing mechanism consists of a potential transformation step, an aggregation, and an update step. For simplicity, edge features are left out for now:

1. **Message transformation**: In the transformation step, a transformation function/matrix $W_{neigh}^{(l)}$ is applied to the feature vector of the neighboring nodes $u \in N(v)$. This optional step is done to transform the feature factors into a more suitable format for further processing

2. **Message Aggregation**: In the aggregations step, the possibly transformed feature vectors of the neighbors are aggregated using some kind of aggregation function, e.g. mean or sum to get $m_{N(u)}$ (Khemani et al. (2024)):

$$m_{N(u)}^{(l)} = agg(\{W_{neigh}^{(l)} h_u^{(l-1)} \mid \forall u \in N(v)\} \tag{13}$$

3. **Feature update**: In the feature update step, the current node embedding, $h_v^{(l-1)}$ gets updated using the current embedding, optionally transformed by $W_{neigh}^{(l)}$ and the aggregated neighboring node embeddings $m_{N(u)}^{(l)}$ The update function is usually a neural network (Khemani et al. (2024)), however at least differentiable (Hamilton (2020a)):

$$h_v^{(l)} = update(W_{self}^{(l)} h_v^{(l-1)}, m_{N(u)}^{(l)}) \tag{14}$$

The updated node embedding $h_v^{(l)}$ contains now information of the node embeddings of the node neighborhood. Refer to Figure 3.1 for a visualization of this process. A more general approach, that also leverages the power of edge features would simply adjust the aggregation step of equation 6. The goal then is, to form more robust node embeddings that incorporate also the type of relation between nodes, quantified by edge features. As pointed out by Hamilton (2020b), also respecting edge features $x_{(v,u)}$ requires at the most basic level an adjustment of

equation 6 as follows:

$$m_{N(u)}^{(l)} = agg(\{W^{(l)}(h_u^{(l-1)} \oplus x_{(v,u)}) \mid \forall u \in N(v)\} \tag{15}$$

where $(h_u^{(l-1)} \oplus x_{(v,u)}) \in \mathbb{R}^{d_H+d_E}$ denotes the concatenation of node embedding vector $h^{(l-1)}$ with edge feature $x_{(v,u)}$.

It is worth mentioning that there exist other methods to account for edge features (Schlichtkrull et al. (2017), Y. Li et al. (2017)) and J. Chen and H. Chen (2021)). However they all build up on the same foundation as equation 8 by finding ways to incorporate edge features into the message aggregation step.

At each iteration, each node gathers information from its 1-hop neighborhood. As we apply multiple GNN layers, meaning multiple message-passing iterations, each node gradually collects information from nodes that are farther away. This happens because, at iteration l, the node embeddings from iteration $l-1$ serve as the input. When $l > 1$, the neighboring node embeddings $\{h_u^{(l-1)} \mid \forall u \in N(v)\}$ already contain information from their respective neighborhoods. Using these embeddings as input for the next iteration l, we see that information from not only the immediate neighbors but also the neighbors' neighbors is aggregated. For instance, at iteration 2 $h_v^{(2)}$ includes information from its 2-hop neighborhood. This process continues, such that after k iterations, each node embedding incorporates data from its k-hop neighborhood. (Khemani et al. (2024)). For a visualization of this process refer to Figure 3.2. However, it is important to note that this process cannot continue infinitely because of the potential of oversmoothing where node embeddings of different nodes start to converge when the number of layers goes to infinity. The GNN loses then all of its expressive power. [1]

After applying k GNN layers, we obtain a new graph $G^{(k)}$ which consists of the node embeddings $\{h_v^{(k)} \mid \forall v \in V\}$ and the original edges E. The node embeddings can be concatenated into a matrix $H^{(k)} \in \mathbb{R}^{|V| \times d^{(l)}}$. This matrix can then be used for further processing. One option is to make predictions at the node level, which requires a predictor $C_{\text{node}} : \mathbb{R}^{|V| \times d^{(l)}} \mapsto \mathbb{R}^{|V|}$ (Xiao et al. (2021)). Another option is to make predictions at the graph level, which also requires a predictor. However, before applying the predictor, a global pooling operation $P : \mathbb{R}^{|V| \times d^{(l)}} \mapsto \mathbb{R}^{d^{(l)}}$ is performed to obtain a global graph embedding. This embedding can then be used by a graph-level predictor $C_{\text{graph}} : \mathbb{R}^{d^{(l)}} \to \mathbb{R}$ to make graph-level predictions (Reiser et al. (2022)). Figure 4.3 visualizes the pooling operation.

[1] For a more detailed explanation of this concept see the *appendix B*

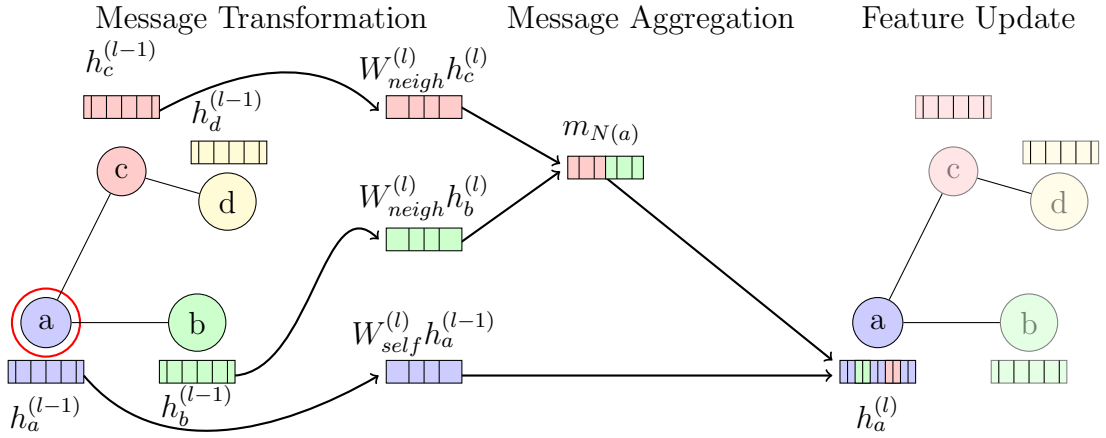


Figure 4.1.: Graph G containing four nodes a,b,c,d with corresponding node features. No edge feature for simplification. Visualization of one GNN layer for node a.

4.3. GNN models

The last part of this section gives an overview of three important aggregation methods widely used when working with GNNs.
First, the basic GNN message passing is defined as:

$$h_v^{(l)} = \sigma\left(W_{self}^{(l)} h_v^{(l-1)} + W_{neigh}^{(l)} \sum_{u \in N(v)} h_u^{(l-1)} + b^{(l)}\right) \tag{16}$$

where $W_{self}^{(l)}, W_{neigh}^{(l)} \in \mathbb{R}^{d^{(l)} \times d1(l-1)}$ are trainable parameter matrices, σ denotes an elementwise non-linearitiy (e.g. tanh or ReLU) and $b^{(l)}$ is a trainable bias term (Hamilton (2020a)). Sticking to the message passing notation, we can use equations 6 and 7 and specify them as follows:

$$\textbf{Aggregation:} \quad m_{N(u)}^{(l)} = \sum_{u \in N(v)} h_u^{(l-1)} + b^{(l)} \tag{17}$$

$$\textbf{Update:} \quad update(h_v^{(l-1)}, m_{N(u)}^{(l)}) = \sigma\left(W_{self}^{(l)} h_v^{(l-1} + W_{neigh}^{(l)} m_{N(u)}^{(l)}\right) \tag{18}$$

The aggregation involves a simple summation of the neighboring node embedding. The update step uses two different weight matrics to transform the aggregated

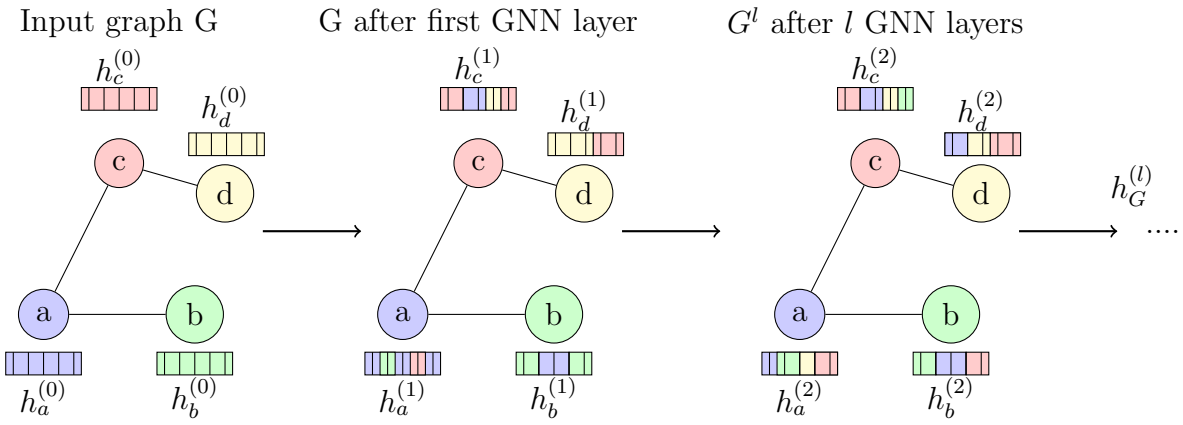


Figure 4.2.: Multiple iterations of the message passing process, i.e. multiple GNN layers. After the first iteration, each node only contains information from direct 1-hop neighborhood. After two iterations, each node has information of 2-hop neighborhood. This process continues.

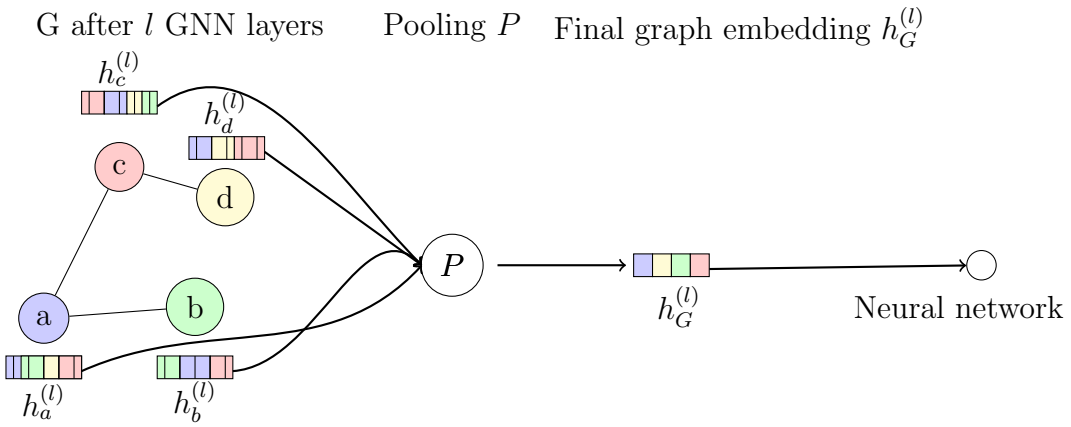


Figure 4.3.: Pooling operation P on through GNN passed graph G. Output is a global graph embedding $h_G^{(l)}$ that contains global, aggregated information about the whole graph structure and can be used for further processing such as graph level predictions by a new neural network

message and the original node embedding to form the new node embedding $h_v^{(l)}$. It is worth noting, that the notation we used, i.e. equations 9 to 11, depicts how the message passing step affects the node embedding of one specific node v in layer l. However, we can also use the concatenation matrix of all node embeddings $H^{(l)}$. This matrix lets us then formulate the message passing algorithm for all nodes simultaneously, i.e. for the whole graph as follows:

$$H^{(l)} = \sigma\left(AH^{(l-1)}W_{neigh}^{(l)} + H^{(l-1)}W_{self}^{(l)}\right) \tag{19}$$

where A is the adjacency matrix of the graph. This lets us get rid of the sum over the neighboring nodes from equation 9, because non neighboring nodes have a value of 0 in A and are therefore ignored. This graph level formulation will be used sometimes when it is convenient.

Second, another fundamental aggregation method used in GNNs is the one of Graph Convolutional Networks (GCN) (Kipf and Welling (2017)). The aggregation process, known as symmetric-normalized aggregation (Hamilton (2020a)) is given as follows:

$$h_v^{(l)} = \sigma\left(W^{(l)} \sum_{u \in N(v) \cup v} \frac{h_u^{(l-1)}}{\sqrt{|N(u)||N(v)|}}\right) \tag{19}$$

where $W^{(l)}$ is a learnable parameter matrix of layer l.
Unlike Equation 9, this formulation includes a normalization of the node embeddings $h_u^{(l-1)}$ by $\sqrt{|N(u)||N(v)|}$. This symmetric normalization helps prevent numerical instabilities (Kipf and Welling (2017)). Additionally, the GCN employs a shared transformation matrix $W^{(l)}$ for both the neighboring nodes and the node itself, rather than using two distinct weight matrices. This strategy helps avoid overfitting and reduces the model's complexity and the number of parameters. (Kipf and Welling (2017)). The graph-level formulation is represented as:

$$H^{(l)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l-1)}W^{(l)}\right) \tag{20}$$

Here $\tilde{A} = A + I$ is the adjacency matrix A with added self-loops at the diagonal, ensuring that the node embeddings $h_v^{(l-1)}$ of the current node are also considered. Without this addition, the diagonal entries would be zero, leading to the node's features being ignored. The shared weight matrix $W^{(l)}$, accounts for both the neighboring nodes and the node itself.
$\tilde{D} = D + I$ is the degree D matrix of the graph with added self-loops, such that $\tilde{D}_{vv} = deg(v) + 1$, i.e. the degree of node v plus one. This adjustment ensures

that the node itself is considered during updates and maintains the integrity of the normalization term. The term $\tilde{D}^{-\frac{1}{2}}$ represents the inverse square root of the degree matrix, incorporating the symmetric normalization.

Finally, the Graph Attention Network (GAT) builds upon the normalization idea of GCNs. Instead of predetermining the contribution of each neighboring node embedding by setting the normalization to $\frac{1}{\sqrt{|N(u)||N(v)|}}$ the GAT introduces a learnable weight $\alpha_{(v,u)}$ that represents the importance of node u when updating node v relative to all nodes $u \in N(v)$.
To determine $\alpha_{(u,v)}$, Veličković et al. (2018) suggest applying an attention mechanism $a : \mathbb{R}^{d^{(l)}} \times \mathbb{R}^{d^{(l)}} \mapsto \mathbb{R}$ on $h_v^{(l-1)}$ and $h_u^{(l-1)}$. Formally, this is:

$$e_{(u,v)} = a(h_v^{(l-1)}, h_u^{(l-1)}) \tag{21}$$

The attention mechanism a follows Vaswani et al. (2023), usually used in the field of natural language processing for modern models like GPT. [2]. Normalization is then applied to all $e_{(v,u)} \forall u \in N(v)$ to derive $\alpha_{(v,u)}$:

$$\alpha_{(v,u)} = \frac{\exp\big(e_{(v,u)}\big)}{\sum_{w \in N(v)} \exp\big(e_{(v,w)}\big)} \tag{22}$$

such that $\sum_{u \in N(v)} \alpha_{(u,v)} = 1$. If we set $\alpha_{(u,v)} = \frac{1}{\sqrt{|N(u)||N(v)|}}$, we get the GCN again. This shows that $\alpha_{(u,v)}$ represents the normalized importance of neighboring node embeddings for node v.
The update formula on the node level is:

$$h_v^{(l)} = \sigma\left(\sum_{u \in N(v) \cup v} \alpha_{(u,v)} W^{(l)} h_u^{(l-1)}\right) \tag{23}$$

and on the graph level:

$$H^{(l)} = \sigma\left(\mathrm{A}_\alpha H^{(l-1)}\right) \tag{24}$$

where A_α contains the attention factors from equation 22.
Each approach enhances message passing by improving the model's ability to dynamically assess and adjust the significance of neighboring nodes. While GCNs operate under the assumption that each neighbor contributes equally to the node's

[2]The exact description of the attention mechanism is out of scope for this thesis, but can be read in the paper Vaswani et al. (2023).

representation, GATs refine this by learning to assign different attention weights, thereby allowing the model to focus on more relevant nodes.
This nuanced understanding of GNNs forms the backbone of this thesis. GNNs are not merely tools used within the study but are integral to the proposed solution approach. By rigorously exploring and leveraging their capabilities, this work aims to push the boundaries of what can be achieved in graph-based learning tasks. The insights gained from understanding these models' underlying mechanisms are essential, as they directly inform the design and implementation of the solutions discussed in the subsequent chapters. Thus, a thorough examination and application of GNNs are neccesary and critical for addressing the challenges tackled in this thesis.

5. Practical Problem: Replicated State Machines

5.1. Introducing Replicated State Machines

A *State Machine* is defined by a set of commands C and states S. Each command is a specific action that can change the machine's state by modifying these variables or by producing some kind of output with a process f defined as either $f : C \times S \mapsto S$ or $f : C \times S \mapsto O$, depending on whether the process changes the internal state of the system or produces some output in the output space O. The execution of these commands is deterministic (Schneider (1990)), i.e., if $c_i, c_j \in C$ with $c_i = c_j$ and $s_m \in S$ is the current state of the machine, then we can state that $f(c_i, s_m) = f(c_j, s_m) = s_n$ with $n \neq m$ (analogously if f maps to O).
Clients interact with state machines by requesting the execution of a command. This involves specifying which state machine to use, the particular command to execute, and providing any necessary information the command might require to run. The state machine processes these requests one at a time, and as a result, can perform actions such as activating physical devices (actuators), interacting with peripheral devices (like disks or terminals), or responding to clients who are waiting for the outcome of their requests Schneider (1990).

Given their susceptibility to faults, state machines may fail during command execution, affecting the output. To mitigate this, *State Machine Replication* was introduced. This approach involves replicating the state machine across multiple processors in a distributed system. A fault-tolerant state machine is achieved by ensuring that each replica, operated by a non-faulty processor, starts in the identical initial state and processes the same requests in the same sequence. This guarantees that all replicas, given their deterministic behavior, produce the same output. Assuming each failure impacts only one processor or one replica, the correct output can be obtained by aggregating the outputs of the replicas. This aggregation often involves a majority vote mechanism or consensus to address discrepancies arising from failures (Schneider (1990)).
Typically, a leading replica orchestrates this process, distributing requests to followers for execution and consensus, a critical step in achieving fault tolerance and

handling operational delays effectively (Lawniczak and Distler (2021)).
Every communication step in this process is affected by some request delay, typically measured by the latency. In contrast to plain request-response schemata, where the delay is only considered between the machine and the client, the Replicated State Machine (RSM) scenario adds additional delay sources due to the needed machine-to-machine communication. The delay is accumulated for each request passing step (Köstler et al. (2023)).

The expansion of computer networks to a global scale necessitates robust, fault-tolerant systems accessible worldwide. Planetary State Replication extends traditional state machine replication to large networks, exemplified by data centers (which are the State Machines in this setup) serving globally distributed clients.
Long geographical distances between clients and data centers negatively impact the request delay of such global systems. As elaborated earlier, the additional complexity of State Machine Replication even worsens this (Köstler et al. (2023)).

Global networks often rely on a select few data centers from a larger pool to act as replicas, primarily to manage costs effectively. For instance, as proposed by Köstler et al. (2023), 4 out of 10 data centers are used as replicas. The configuration that determines which data centers are actually used as replicas (tactice data centers) is dynamic, i.e. it can be adjusted during the system's run. Further, client positions vary during different daytime, such that the majority of the requests also come from different regions.
Consequently, this creates the possibility to dynamically optimize the system's configuration, such that the overall delay, or latency, is minimized. For instance, if the current peak of client requests originate in East USA, but some or all replicas are located in East Asia, the latency is much higher than in a comparable configuration with replicas mainly located in East Asia. For a visualization refer to Figure 5.1.

5.2. Problem Formulation

Let $\mathbb{L} = \{1, 2, ..., n\}$ be the set of all possible data center locations and $C_t = \{n+1, n+2, ...\}$ the set of client locations at time step t. Further, let $d_{(u,v)^t} \in \mathbb{R}$ be the latency between location or client u and location or client v at time step t. The RSM system can then be modeled as a dynamic graph $G_t = (V_t, E_t)$, where the nodes are given by $V_t = C_t \cup \mathbb{L}$ and the edges are given by $E_t = \{(u,v)_t \mid \forall u, v \in V_t\} = \{d_{(u,v)^t} \mid \forall u, v \in V_t\}$. A dynamic graph is a graph whose edges and/or nodes change over time (Harary and Gupta (1997)), undermined by the subscript t which denotes the instance of a graph at time step t. Let $D_t \subset \mathbb{L}$ be

Figure 5.1.: These maps show the active replicas (black-filled circles) and all possible locations (non-filled circles) at different daytime. At different daytime, most client requests (darkened areas) come from different locations. In this case, the data centers are placed optimally given the current peak of the client requests (Köstler et al. (2023)). This graphic displays the client movement starting at 06:00 am in the upper left, finishing at around 08:00 pm in the evening.

the set of currently active data centers. The goal is then to find a configuration D_t^* that minimizes the aggregated latency of the system computed by the aggregator $F(G_t)$ plus the reconfiguration costs and penalty costs given by $K(D_{t-1}, D_t)$. Penalty costs are additional constraints specificly used for the RSM problem[1]. For example, if a location gets selected as new active data center in step $t+1$ which was not a passive data center[2] additional penalty costs are incurred.

Considering this property, one can see that the RSM problem holds notable similarities with LA problems in general, but especially with FLPs. We can interchange facilities with data centers, demand points with clients, demand quantity with number of requests sent per client, and distances with latencies, emphasizing the similarity. As all clients send requests to all data centers there is no need to assign each client to exactly one data center (facility). This offers a more simple

[1]The exact details of the penalties for the RSM system are out of scope for this thesis but can be studied in Köstler et al. (2023)

[2]A passive data center is a data center that collects latency information around its location. Changing a passive data center to an active data center is cheaper than changing an inactive data center to an active data center. It is like an additional information source that can be used efficiently but is not controlled by the solution algorithm directly.

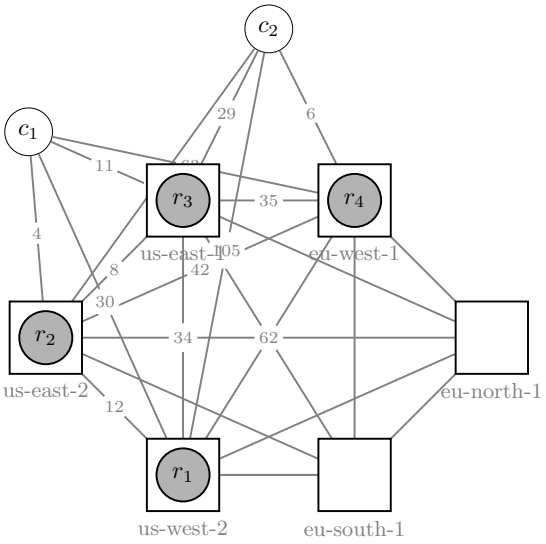


Figure 5.2.: Simplified setup for a RSM system: six potential locations, four active data centers (r_1, r_2, r_3, r_4) two clients (c_1, c_2) and edges denoting latencies

access to the problem (constraints that required these assignments can therefore be skipped). Hence, a concrete assignment about which client has to communicate with which data center is not needed. Refer to Figure 5.2 for a visual clarification. We can then assign the RSM problem to the broad scope of LA problems by using the classification schema from *section 2.2.*

1. Space: At first, data center locations are discrete, since they are predetermined and do not change over time. Client locations are in general continuous, as they can appear theoretically everywhere on the planet. In practice, however, client locations are often modeled using simulations or occur in similar locations making them also assignable to the discrete space. For this thesis, we will therefore model both clients and locations in the discrete space [3]

2. Graph and Tree: The problem can be modeled as a graph

3. Number of Facilities: For this problem setup, the number of facilities/data centers can be either fixed or variable. Mostly, however, the number of data centers will be predetermined and therefore fixed.

4. Static vs dynamic: As shown earlier, the nature of the system is dynamic.

[3] To account for the continuous nature of client location, we can set a large upper bound for possible client locations. In each episode then, only a random subset will be actually modeled.

5. Distance Metric: Latency is used as the distance metric

6. Deterministic vs Probabilistic: As the movement of clients is not predictable accurately, the system dynamic is stochastic, i.e. probabilistic. Simulations use different kinds of distributions, to model the client movement, accentuating the stochastic nature of the problem.

7. Capacitated vs Uncapacitated: As data centers can handle an unlimited amount of requests, the problem can be assigned to the uncapacitated category.

Using this classification scheme and the fact that the system of RSM is based on a dynamic, stochastic client movement lets us frame the problem as a DSFLP. Furthermore the DSFLP does not explicitly model the assignment of clients to facilities, which aligns well with the given RSM setup where each client interacts with every data center. This enables the orientation towards the optimization intuition of DSFLP which serves as a theoretical foundation for later suggested solution approach. In its most basic form, the goal of DFSLP applied to the given problem can be noted formally as:

$$\underset{d_t \in \mathcal{P}(\mathbb{L})}{\text{minimize}} \quad \mathbb{E}\left[\sum_{t=1}^{\infty} \left[F(G_{t+1}) + K(d_{t-1}, d_t)\right] B^{t-1}\right]$$

where $\mathcal{P}(\mathbb{L})$ is the power set of data center locations such that d_t is a possible configuration of active data centers. G_{t+1} is the new state, i.e. graph instance, of the system using the new configuration d_t and new client positions C_{t+1}. B is the known discount factor from DSFLP.

Comparison of problem components	
Location Allocation	RSM
Facilities	Data centers
Demand points	Clients
Demand quantity	Number of requests
Distance	Latency

Figure 5.3.: Depiction of relation of components in LA problem and RSM problem

5.3. Proposed Solution: Graph Neural Network based Reinforcement Learning

5.3.1. Graph Neural Networks as Model Foundation

GNNs have proven highly effective in understanding and working with graph-structured data. This capability will be leveraged to formulate an optimization approach for the RSM problem. Given the current state of the system at time step t, modeled as a graph $G_t = (V_t, E_t) = (C_t \cup L_t, \{d_{(u,v)^t} \mid \forall u, v \in V_t\})$ and corresponding node features $X_V \in \mathbb{R}^{|V_t| \times 3}$, where node features are the number of requests, the type of location (active, inactive, passive, client) and the current time step, we aim to construct a function f that maps the current state of the system to a relocation tuple (i, j) of nodes. Let $G = \bigcup_{t=1}^{\infty} G_t$ and $V = \bigcup_{t=1}^{\infty} V_t$ represent the global collection of all graph instances and their respective nodes. The signature of f can be described as:

$$f : G \mapsto V \times V, \quad f(G_t) = (i, j) \tag{25}$$

where the output relocation tuple (i, j) indicates that $i \in D_t \cup \varnothing$ is the location of currently active data centers to be removed, and $j \in (\mathbb{L} \setminus D_t) \cup \varnothing$ is the location of currently inactive data centers to be added. The union with the empty set allows for the possibility of no removals or additions. Using this reconfiguration tuple, we state the following transition identity for the set of active data centers:

$$D_{t+1} = (D_t \setminus i) \cup j \tag{26}$$

This approach enables the model to make gradual adjustments at each time step. Possible system configurations include:

- $i = j$: No reconfiguration is done
- $i = \varnothing$ and $j \neq \varnothing$: The amount of active data centers increases
- $i \neq \varnothing$ and $j = \varnothing$: The amount of active data centers decreases
- $i \neq \varnothing$ and $j \neq \varnothing$: The amount of active data centers stays constant

These relocation options illustrate the potential reconfiguration actions the model can undertake. By applying multiple reconfiguration steps, the model can directly set the number of active data centers, even though adjustments are made one step at a time.

The function f is implemented as a GNN, which outputs, for every location node

$v \in \mathbb{L}$, two probabilities p_v^i and p_v^j, ensuring that $\sum_{v \in \mathbb{L}} p_v^i = 1$ and $\sum_{v \in \mathbb{L}} p_v^j = 1$. Formally, we need a predictor:

$$C_{RSM} : \mathbb{R}^{|\mathbb{L}| \times d^{(l)}} \mapsto \mathbb{R}^{|\mathbb{L}| \times 2} \tag{27}$$

that maps the final node embeddings of the location nodes $H^{(l)}$ to the desired probabilities.
A critical component of this approach is the training of the GNN. Typically, this is done offline with a supervised learning objective, where a dataset is created containing pairs of input and target data. For the RSM case, the dataset would consist of graph instances G_t with corresponding optimal relocation tuples $(i,j)_t^{target}$. This data is used to train the GNN to imitate the optimal relocation pattern and generalize to unseen data. This is achieved using a loss function L that quantifies the quality of model predictions $GNN(G_t) = (i,j)_t^{pred}$, rewarding good predictions and penalizing poor ones. The objective is to minimize L. Let θ be the parameters of the GNN. Formally, this can be expressed as:

$$\underset{\theta}{\text{minimize}} \quad L((i,j)_t^{pred}, (i,j)_t^{target}) \tag{28}$$

However, as discussed in Section 2.3.5, using supervised or offline learning for combinatorial problems like the RSM problem is challenging for several reasons:

1. **Large Dataset**: A large dataset requires assigning the optimal relocation tuple for every instance, which is not scalable due to the NP-hardness of the problem.

2. **Overfitting Risk**: There is a high risk of overfitting, where the model memorizes specific mappings from graph instances to relocation tuples instead of generalizing, leading to poor performance on unseen data.

3. **Retraining**: The need for retraining arises when new system dynamics occur or when the system's setup changes.

4. **Poor performance feedback**: Traditional loss functions in offline learning do not account for the degree of suboptimality in predictions. They treat any deviation from the optimal relocation as equally incorrect, without considering that some incorrect predictions might still be near-optimal. This lack of nuanced feedback prevents the model from dynamically adjusting its learning based on the severity of errors.

The limitations of offline learning highlight the advantages of online learning methods for the given scenario. Therefore, the next section introduces an online-learning-based approach using RL

5.3.2. Reinforcement Learning as Training Approach

Unlike offline learning, online learning approaches gather training data through interaction with an environment. In essence, online learning relies on the quality and quantity of data generated from these interactions rather than on a predefined dataset. RL is a widely used methodology in online learning. It operates on the principles of an agent interacting with an environment, receiving rewards, and transitioning to new states, which aligns perfectly with online learning principles.

The application of RL to the RSM problem is particularly promising for several practical and theoretical reasons:

1. **Unsupervised Training**: The RSM problem, being a large-scale optimization challenge, cannot be effectively tackled using supervised training in an offline manner. RL, as an online learning approach, provides a suitable solution by continuously learning from interactions with the environment

2. **Quick Decision Making**: The RSM setup demands rapid decision-making at each optimization step. Traditional heuristics often struggle with large-scale instances due to long runtimes (Cavalcanti Costa, Mei, and Zhang (2021)). The generalizing capabilities of RL mitigate this issue, offering quick and satisfactorily accurate solutions.

3. **Theoretical Foundation**: The theoretical underpinnings of the DSFLP offer a robust foundation for reframing the RSM problem within the realm of RL.

To comprehend this in detail, we first need to explain the solution approach for the DSFLP, using the notation provided in the corresponding *section 2.1.5*.
The authors employ a policy K that specifies a location $k(i,j) \in N$ to which the facility is moved whenever the decision-maker's observation of the state is (i,j). This incurs a cost of $f(i,k(i,j)) + g(k(i,j),l)$. Refer to Figure 5.4 for a simplified visualization of this process. We can define the expected service cost matrix $W = PG$, where P is the Markov transition matrix. The expected service costs depend on the stochastic client positions, which are accounted for by the transition matrix P. If a system operates indefinitely under policy K, then the total expected costs $v_K(i,j)$ are characterized by state-value equations (Rosenthal, J. A. White, and Young (1978)):

$$v_K(i,j) = f(i,k(i,j) + w(j,k(i,j)) + \beta \sum_{l=1}^{n} p(j,l) v_K(k(i,j),l), \quad \forall (i,j) \in N \times N \tag{29}$$

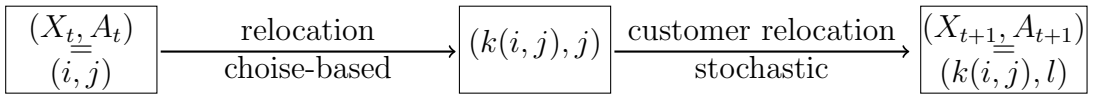


Figure 5.4.: Visualization of the DSFLP process where facilities X_t are relocated by choice using a policy k and customer A_t move stochastically using the Markov transition matrix

Examining equation 23 closely, the first term represents the relocation cost $f(i, k(i, j))$ plus the expected service costs $w(j, k(i, j))$ if we choose policy k in the current state. The second term sums over all possible client locations $[n]$ and adds the state-value of being in that state $v_K(k(i, j), l)$ multiplied by the probability $p(j, l)$ of being in that state, determined by the transition matrix P. This essentially computes the expected value of the next state-value using the stochastic client positions[4]. This equation employs recursive principles by basing the value of the current state on the value of all possible next states.
The optimal policy K^* is then the one that minimizes the expected costs of the system (Rosenthal, J. A. White, and Young (1978)), formally depicted as:

$$\underset{k \in N}{\text{minimize}} \quad \{f(i, k) + w(j, k)) + \beta \sum_{l=1}^{n} p(j, l) v_K(k, l)\}, \quad \forall (i, j) \in N \times N \tag{30}$$

Thus, we aim to find policy k that minimizes the expected costs of the current state based on a system of $n^{(2)}$ equations and $n^{(2)}$ unknowns. As n grows, this approach becomes infeasible. Hence, new procedures are necessary to overcome this dimensionality problem (Rosenthal, J. A. White, and Young (1978)). This is where RL, and later DRL, comes into place.

As shown in *chapter 3*, a foundational concept in RL is the Bellman equation, which recursively unrolls the state-value function $v^\pi(s)$, denoted by:

$$v^\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[r + \gamma v^\pi(s')\right], \quad \text{for all } s \in S \tag{31}$$

where $v^\pi(s)$ marks the value in state s given we follow policy π, $\pi(a|s)$ represents the probability of taking action a dictated by policy π, $p(s', r|s, a)$ is the probability

[4]For the discrete case, the expected value is computed as $\mathbb{E}[X] = \sum P(x)x$. Setting $p = p(j, l)$ and $x = v_K(k(i, j), l)$ results in the described identity.

of getting reward r and being in state s' after taking action a in state s (retrieved by the Markov transition probabilities), and $r + \gamma v^{\pi}(s')$ is the immediate reward plus the discounted value of the next state with discount factor γ (Sutton and Barto (2018b)).
If we opt for deterministic action, instead of a distribution $\pi(a|s)$ and deterministic rewards r, the equation simplifies to:

$$v^{\pi}(s) = r + \gamma \sum_{s'} p(s'|s,a) v^{\pi}(s'), \quad \text{for all } s \in S \tag{32}$$

The RL objective is now to find an optimal policy π^* with the value $v^{\pi^*}(s)$ by outputting an action a that fulfills the following equation:

$$\underset{a}{\text{maximize}} \quad r + \gamma \sum_{s',r} p(s', r|s,a) v^{\pi}(s') \tag{33}$$

It is evident that equation 32 shares notable similarities with equation 29. Both denote the state value if we follow policy K or policy π, respectively. We can set $r = -(f(i,k(i,j)) + w(j,k(i,j)))$ and $\beta = \gamma$. Furthermore, we can set $s = (i,j)$ and $s' = (k(i,j),l)$, where l is the next client position realized with the probability $p(j,l)$. Finally, the probabilities can be aligned by setting $p(j,l) = p(k(i,j), l \mid i,j,k) = p(s'|s,a)$, as $p(s'|s,a)$ simply denotes the transition probability from state s to state s' after taking action a. Equation 34 offers a visual simplification for this substitution process.

$$r + \gamma \sum_{s',r} p(s', r|s,a) v^{\pi}(s') \Leftrightarrow -(f(i,k(i,j) + w(j,k(i,j)))) + \beta \sum_{l=1}^{n} p(j,l) v_K(k(i,j),l) \tag{34}$$

This allows us to reframe the state-value equation of the DSFLP as the state-value equation of RL using the Bellman equation. Thus, finding an optimal policy π^* that maximizes discounted cumulative rewards also serves as the optimal policy $K^* = \pi^*$ that minimizes the discounted cumulative costs as described in equation 30. Hence, the DSFLP can be understood as a Markov decision process that is solvable by RL algorithms as presented in *chapter 3*.

This derivation underscores the theoretical justification for applying RL algorithms to the DSFLP, besides the practical benefits. A common RL algorithm that solves the state-value equation is *Dynamic Programming*, which utilizes the known Markov transition probabilities to find an optimal policy (Sutton and Barto

(2018c)). However, these transition probabilities are often unknown in real-world applications. Therefore, other algorithms like *Monte Carlo Methods*, *Temporal Difference Learning*, and *Q-learning* are employed to interact with the environment and collect sampled transitions that approximate the real, unknown transition probabilities. This adjustment is particularly useful for the RSM problem, where transition probabilities are typically unknown. [5]

In summary, reframing the DSFLP as a classic RL problem enables the application of commonly used RL algorithms to find optimal relocation policies. Since the DSFLP is the theoretical baseline for the RSM, RL algorithms that do not rely on full knowledge of transition probabilities can be effectively applied to the RSM problem.

5.3.3. Final Solution Approach

As the RSM problem can be framed as an RL problem, we can now combine the GNN approach with RL. Traditional RL algorithms are often not scalable when the state space (the number of all possible states) grows. The approximative and generalizing power of neural networks can be used then to build RL algorithms that scale much better with increasing state spaces. This is called DRL. Several algorithms exist here, e.g. *Policy Gradient* methods (Sutton and Barto (2018a)) such as *Proximal Policy Optimization* (PPO) (Schulman, Wolski, et al. (2017)) or *Trust Region Policy Optimization* (Schulman, Levine, et al. (2017)) but also *Deep Q-Learning* (DQN) (Mnih, Kavukcuoglu, Silver, Rusu, et al. (2015)) and variants (Hasselt, Guez, and Silver (2015) and Z. Wang et al. (2016)) [6]. As GNNs are neural networks we can integrate them into DRL and use them as the core model with which the RL algorithm works with. Concretely, the suggested solution involves employing a GNN based PPO and a GNN based DQN.
To start off, the core components of every RL problem are assigned to the RSM problem:

- Environment E: The environment will be RSM system, modelled by a simulation S_{RSM} that mimics the real system dynamics
- State s_t: The state of the environment is the graph instance G_t at time step t as explained in *section 5.3.1*
- Action a_t: The actions are given by the relocation tuple (i, j) as described in *section 5.3.1*. So the action space consists of two actions.

[5] A more detailed explanation of these algorithms can be found in the *appendix D.1*
[6] For further insights and a detailed explanation of the relevant algorithms, see the *appendix D.2 and D.3*

- Reward r_t: The reward displays the negative weighted sum of aggregated latency $F(G_{t+1})$ and reconfiguration costs $K(d_{t-1}, d_t)$, such that $r_t = -(\lambda_{lat} F(G_{t+1}) + \lambda_{re} K(d_{t-1}, d_t))$. This aligns with the theoretical foundation presented in *section 5.3.2*

All of that data is retrieved by interacting with the simulation S_{RSM}[7].
As PPO and DQN require different GNN architectures, the last part of the section will give a short intuition of how the models are built. In both cases, a *GAT* type GNN with three layers is applied to the state $s_t = G_t$, such that we receive node embeddings $H_t^3 = \mathbf{GAT}(G_t)$. Only three layers are used, because the largest relevant neighborhood for this problem is the 2- and 3-hop neighborhood, as every client has an edge with every data center and every data center has an edge with every other data center. PPO requires an actor that outputs action distributions $\pi_i(a_t \mid s_t)$ and $\pi_j(a_t \mid s_t)$ for the relocation tuple (i, j) and a critic that outputs a value estimation $v^\pi(s_t)$ for the current state s_t. DQN requires an action-value output $q_i^\pi(s_t, a_t)$ and $q_j^\pi(s_t, a_t)$ that assigns action-values for each remove-location-action i and add-location-action j respectively. More precisely:

1. **GNN for PPO**: A predictor $C_{actor}^i : \mathbb{R}^{|V| \times d^{(3)}} \mapsto \mathbb{R}^{|\mathbb{L}|}$ is applied to assign each location node a score l_v[8]. These scores are then to used receive the remove-location action distribution $\pi_i(a_t \mid s_t)$. Let $u \sim \pi_i(a_t \mid s_t)$ be the selected node to be removed. Inspired by Guo, Xu, and Yaohui Jin (2023) , we opt to use the node embedding $H_{t[u]}^3 = h_u^3$ of the node u as additional information for the second action, the add-location-action j.
   Hence, the second predictor $C_{actor}^j : \mathbb{R}^{|V| \times d^{(3)}} \times \mathbb{R}^{1 \times d^{(3)}} \mapsto \mathbb{R}^{|\mathbb{L}|}$ now gets two inputs, H^3 and h_u^3, returning scores for the action j which will be used for the action distribution $\pi_j(a_t \mid s_t)$.
   Finally, the critic GNN simply uses a pooling operation $P_{critic} : \mathbb{R}^{|V| \times d^{(3)}} \mapsto \mathbb{R}^{d^{(3)}}$ combined with a predictor $C_{critic} : \mathbb{R}^{d^{(3)}} \mapsto \mathbb{R}$ to output the state-values as $v^\pi(s_t) = C_{critic}(P_{critic}(H^3))$.

2. **GNN for DQN**: Here, a predictor $C_{DQN} : \mathbb{R}^{|V| \times d^{(3)}} \mapsto \mathbb{R}^{2 \times |\mathbb{L}|}$ is used that outputs the action values $q_i^\pi(s_t, a_t)$ and $q_j^\pi(s_t, a_t)$ for each data center location.

Both algorithms make then use of the corresponding model architecture.

[7] For a detailed description about the inner workings of the simulation, see the *appendix C*

[8] Technically speaking, the output domain is $\mathbb{R}^{|D_t|}$ as the currently active data centers. However, this is a technical detail which is managed by action masking. It can be found in the corresponding model class in the *repository*

6. Experiments, Training and Results

6.1. Training Process

6.1.1. Supervised Experimental Training

To ensure that the proposed GNN model architecture effectively understood and processed the graph structure, a supervised pre-task was constructed. This pre-task involved creating a dataset of input-output pairs (X, Y), where the input data X consisted of graph instances $G_1, G_2, \ldots, G_n$, and the corresponding target data Y represented the aggregated latency $F(G_t)$ of the system.
The objective was to train the GNN using the C_{critic} predictor and P_{critic} to minimize the mean squared error (MSE) between the actual aggregated latency $y_t = F(G_t)$ and the predicted aggregated latency $\hat{y}_t = C_{\text{critic}}(P_{\text{critic}}(\mathbf{GAT}(G_t)))$. Mathematically, this is expressed as:

$$\underset{\theta}{\text{minimize}} \quad \sum_{t=1}^{n} (\hat{y}_t - y_t)^2 \tag{35}$$

where θ represents the combined parameters of C_{critic}, P_{critic}, and the **GAT** GNN. The experimental training was successful, as indicated by a decrease in the mean squared error over time, as shown in Figure 6.1. This demonstrated that the model architecture was generally capable of understanding and learning from graph data, effectively processing the graph instances G_t.

6.1.2. Imitation Experimental Training

The supervised experimental training demonstrated the model's ability to understand graph data at the graph level, predicting aggregated system latency. However, for the RSM problem, an understanding at the node level is also required, as the goal is to output relocation tuples for each node. Therefore, a second experimental training was conducted. A dataset (X, Y) was constructed, with input data X consisting of graph instances G_t and target data Y consisting of optimal relocation tuples $(i, j)_t$ for 500 graph instances. This was achieved using a brute-force

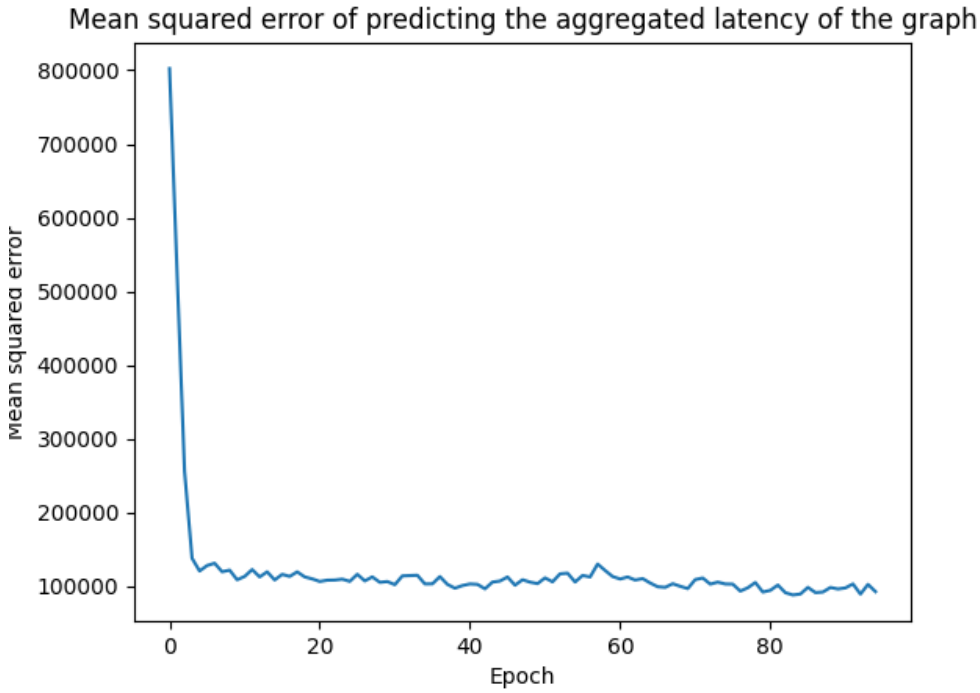


Figure 6.1.: Development of the loss, i.e. the mean squared error, of the supervised training to predict the aggregated latency of the whole graph instance, i.e. the simulated system.

approach, evaluating all possible relocation combinations for each graph instance G_t and selecting the one that minimized the sum of latency $F(G_t)$ and relocation costs $K(d_{t-1}, d_t)$. As discussed in *section 5.3.1*, this method is not scalable due to the combinatorial complexity and NP-hardness of the LA problem. Therefore, this approach was only used for experimental validation of the node-level understanding of the GAT-GNN model.
Using the DQN predictor C_{DQN}, we obtain an output $O_t = C_{DQN}(\mathbf{GAT}(G_t))$ of shape $(2 \times |\mathbb{L}|)$. By selecting the two indices (nodes) with the highest values along the first dimension, we derive the predicted optimal relocation tuple $(\hat{i}, \hat{j})_t = \text{argmax}(O_t)$. This prediction can be quantified using categorical loss functions such as cross-entropy:

$$\underset{\theta}{\text{minimize}} \quad -\sum_{t=1}^{n} \left(\sum_{l=1}^{|\mathbb{L}|} i_{tl} \log\left(\hat{i}_{tl}\right) + j_{tl} \log\left(\hat{j}_{tl}\right) \right) \tag{36}$$

where i and j are the respective elements of the target and predicted relocation tuples. θ represents the combined parameters of C_{critic} and the **GAT** GNN. The training was successful, with the loss decreasing, though it exhibited volatility at times, suggesting that the model's ability to make graph-level predictions is somewhat better. The behavior of the loss is depicted in Figure 6.2. This method is referred to as *Imitation Learning* (Zare et al. (2023)).

Both experimental validation approaches demonstrated that the GAT-GNN is capable of processing graph-structured data. The next section describes the actual RL training process.

6.1.3. RL Training: PPO and DQN

The general setting of the system involved reward r_t with weights $\lambda_{lat} = 1$ and $\lambda_{re} = 0.5$. The general setting of the system involved 15 possible data center locations, i.e. $|\mathbb{L}| = 15$ and 50 client locations, i.e. $|C_t| = 50$ for each step t. Latencies between clients $d_{(u,v)t}$ were retrieved through the simulation S_{RSM} or the environment E. Three active data centers should be maintained, i.e. $|D_t| = 3$. The initial phase involved training a PPO agent. The training was accelerated using a NVIDIA Titan V GPU. The methodology comprised an initial training phase of approximately 50,000 simulation steps, using a specific set of PPO parameters, to observe detectable learning behavior. If successful, this parameter set was retained for further training of an additional 250,000 steps. This incremental approach was adopted due to the eight-hour time requirement to simulate 50,000 steps, making it impractical to train with every parameter set for the full 300,000 steps. 300,000 steps took around two days and were therefore a reasonable amount

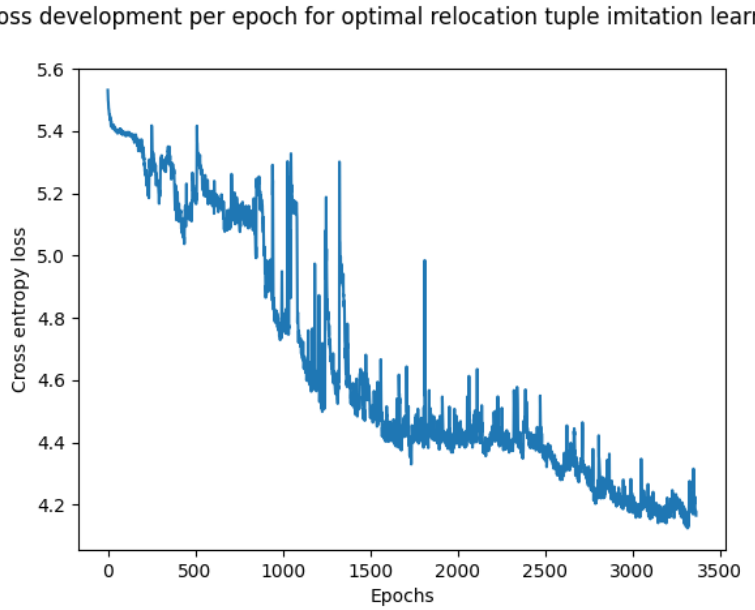


Figure 6.2.: Development of the loss, i.e. the cross-entropy loss, of the imitation training to predict the optimal relocation tuples. It is steadily decreasing emphasizing the general understanding of the model

of steps to observe the learning process while being able to also try new parameter settings. This strategy represents a compromise between identifying the optimal long-term parameter set and ensuring detectable learning behavior, even though it may result in some parameter settings being overlooked. Key adjustable parameters for PPO and DQN can be found in the *appendix D.3.*

Both algorithms were trained using multiple parameter sets to determine the most effective configurations. After completing the training of various settings, the setting that provided the best average reward during training was chosen. [1]

6.2. Training Results

6.2.1. PPO Results

Several parameter settings were tested for both algorithms. The first key finding was that the DQN algorithm consistently outperformed the PPO algorithm. The core principle of policy gradient methods like PPO is to increase the probability

[1] A detailed overview of the algorithms and training process can be found in the *appendix D.*.

of favorable actions towards 1 and decrease the probability of unfavorable actions towards 0. Although the probabilities did change during training, their development did not align with expectations. The probabilities were not significantly shifted towards obviously favorable actions, which is typically observed in other PPO applications.
For instance, in a system configuration with three active data centers, an equally distributed remove-location-action distribution $\pi_i(a_t \mid s_t)$ would have a probability of $\frac{1}{3}$ for each location. After approximately 300,000 steps, the probabilities for more favorable actions were around $\frac{2}{5}$, indicating a slow learning process. However, the expected clear distinction in probabilities was not observed. As a result, the sampling of actions during evaluation phases often returned unfavorable actions, leading to poor evaluation results and suboptimal rewards.
Consequently, the PPO training was not deepened and DQN was carried out.

6.2.2. DQN Results

In general, it can be stated that the overall results of both algorithms, especially the PPO, were not as good as expected. The theoretical foundations and practical reasons for applying RL to this problem combined with GNNs appeared more promising than the actual experimental outcomes. Nevertheless, an overall learning pattern could be observed, as depicted in Figure 6.1 using DQN. The average evaluation reward increased consistently over the first 200,000 steps. After this period, a saturation effect occurred, where the average reward initially decreased slightly and then remained constant for the remainder of the training (approximately 300,000 steps).
Analyzing the training development and delving deeper into what the model learned, it becomes evident that the rules of the RSM system were learned relatively quickly. Since $K(d_{t-1}, d_t)$ also captures penalties for undesirable actions that define soft game rules (e.g., selecting a passive data center as active if it was not previously passive), the model was able to avoid these highly negative reward-yielding actions. This demonstrates the model's general capability to learn beneficial behavior within the environment, as supported by the reward development shown in Figure 6.1. Some parameter settings did not learn the system's soft rules within the first 50,000 steps, resulting in decreased or highly volatile rewards. It is also worth noting that the learned reconfiguration outperformed random search heuristics with the intuition to select a new data center randomly. This should be no surprise since the randomized searches do not pay attention to any high-penalty rule breaks. Also, they do not discover any kind of relocation strategy which the currently best model does. Thus, although not being good enough out of absolute perspective, the model showed competence in being superior to classic, randomized heuristics.

The learning of optimal relocation patterns also took place, but the suggested relocations did not improve overall latency to the extent necessary to benefit the system in a way that it could be applied to the real-world existing system. However, given the detectable general learning pattern, achieving better results may be feasible with different parameter settings and more training time.

6.2.3. Interpretation of Results

One of the main goals of this thesis was demonstrated: the GNN-based RL approach is generally working. The model can learn and adjust its policy to improve the reward over time. While the magnitude of latency improvement was not fully convincing, the general learning tendency highlights the potential of this approach with more optimized parameter settings.

Several factors may have contributed to the saturation of the learning process and the smaller-than-expected reward improvement.
First, the core of the learning process is the simulation S_{RSM}. This simulation should accurately reflect the dynamics of the real-world RSM system. It is possible that the simulation is not realistic enough and fails to capture the complex dynamics of the real system accurately. This can lead to distorted and non-unique states and rewards, hindering the model's efficient learning process. Specifically, the reward r_t, which serves as the only feedback signal for the model, may be too smooth. This means the rewards for good and bad actions do not differ sufficiently to force policy adjustments. A possible solution would be to rebuild S_{RSM} with more nuanced latency and cost feedback.
Second, the node input features X_{Vt} might be too few or lack sufficient expressive power. Currently, the input features include the number of requests sent per client, the type of location, and the current time step. It could be beneficial to store an encoded history of the node's feature development over the past n time steps. This would inform the node not only about the spatial data in its neighborhood but also about the temporal-spatial data of the past time steps. This can be achieved using *Spatio-Temporal GNNs* (Verdone, Scardapane, and Panella 2024) that model the time dimension over instances of dynamic graphs.
Third, the training time might have been too short. As noted earlier, time constraints were significant as the goal was to experiment with different parameter settings to find one that showed some reward improvement. The training time of around two days for 300,000 steps limited the number of parameter combinations tested over a large number of training steps. Consequently, no further training beyond 300,000 steps was conducted, possibly missing out on beneficial relocation strategies that might have been discovered later. This limitation could be overcome in the future with more powerful hardware resources to speed up training.

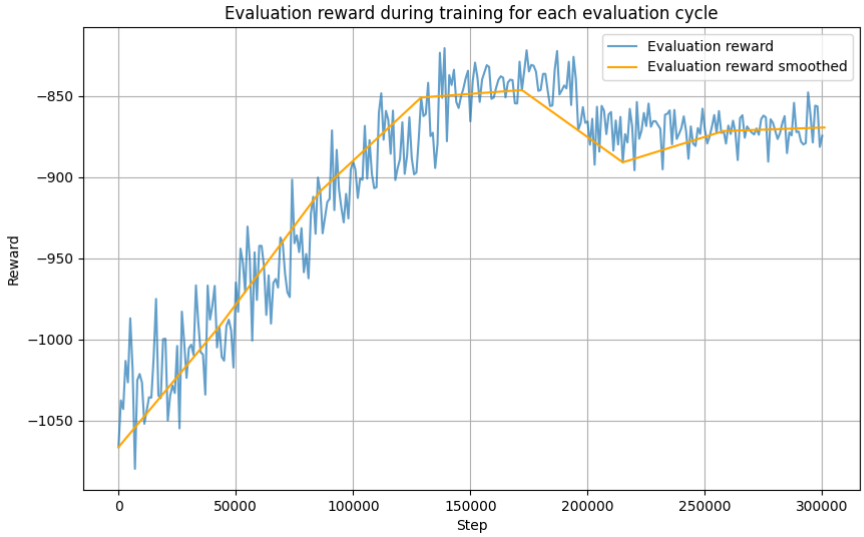


Figure 6.3.: Average evaluation reward of the training of the best model

Lastly, the update steps may have been too infrequent. In the current setup, one update step corresponds to one full hour of the day. Given the periodic trend in client movement over a day, it is possible that favorable relocations were not proposed by the model because client locations changed too quickly. For example, performing an update step every 30 minutes could allow clients to remain longer in the current region, potentially emphasizing the benefit of relocating data centers to that region. The low frequency of hourly update steps sometimes pushed the model toward a local optimum, where it merely rotated active data centers within one region. With five possible locations in Asia for the three active data centers, the model focused on a strategy of switching locations within Asia. Although it is generally better to shift active data centers with client movement, keeping them in one region until clients move back (which occurs more frequently with hourly updates) was a sufficiently good policy to avoid opting for other strategies if they were not discovered. Consequently, this behavior occurred more frequently with rarer relocation updates. This suboptimal strategy selection likely contributed to the saturation effect. More frequent relocations and the injection of expert knowledge could mitigate this. Expert knowledge involves a dataset of human-crafted optimal actions in certain states that help the model escape local optima (Sonabend-W et al. (2020)).

7. Conclusion

This thesis aimed to address the optimization of LA problems through a hybrid approach combining GNNs and DRL. The focus was on solving DSFLP within the context of RSM in global networks. By reframing the RSM problem as an RL problem, we leveraged the capabilities of GNNs to handle graph-structured data and the adaptability of RL to learn optimal policies dynamically. The key contribution and findings were:

1. **Theoretical Foundation**: The RSM problem was suggested as an instance of the broad field of LA problems, especially DSFLPs. Furthermore, a theoretical framework by linking DSFLPs with RL was established, demonstrating how the state-value equations of DSFLP can be reframed as Bellman equations in RL. This connection provided a solid foundation for applying RL algorithms to the RSM problem. This also demonstrates that dynamic LA problems, if formulated in a similar way to the DSFLP, can be framed as a RL problem and therefore making optimal policies of the RL approach also optimal policies for dynamic LA problems.

2. **Graph Neural Network Architecture**: A GNN architecture based on Graph Attention Networks to process the dynamic graph structures inherent in RSM systems was proposed. The GNN was designed to output relocation tuples, capturing the nuanced dependencies between nodes (data centers and clients) in the graph.

3. **Reinforcement Learning Integration**: Two RL algorithms, PPO and DQN, were integrated with the GNN model. These algorithms were adapted to handle the specific requirements of the RSM problem, such as quick decision-making and handling large state spaces.

4. **Simulation Environment**: A comprehensive simulation environment was developed to mimic the real-world dynamics of an RSM system. This simulation facilitated the training and evaluation of the proposed RL algorithms, providing a platform to test various configurations and scenarios.

5. **Experimental Validation**: Experimental training validated the model's ability to understand and process graph data at both the graph and node

levels. The model demonstrated effective prediction of system latency and optimal relocation decisions, showcasing its potential to optimize RSM configurations dynamically.

6. **Validation of GNN-based RL**: The core idea of using a GNN-based RL approach to solve the RSM problem, framed as LA problem, was validated. While the model showed promise, the learning process did not fully meet expectations. Nonetheless, the solution approach demonstrated that the RL model could learn soft rules of the RSM system that are punished with high negative rewards as well as some local optimum strategies. This undermines the general and promising capability to use GNN-based RL approaches for the LA problem.

These findings pave the way for future research. The GNN-based RL approach to optimize LA problems shows considerable promise. Future research could focus on improving the learning process and overcoming local optima in relocation strategies. This can be achieved through new model architectures, enhanced simulations, improved RL algorithms, or more powerful hardware resources. A different concrete problem setting, instead of the RSM, could be also a promising future research option. The main takeaway is that the proposed approach has the potential to challenge existing heuristics, despite not being fully mature yet.

A. Literature Review Table

Table A.1.: Literature Review Table

Author(s)	**Year**	**Title**	**Journal/Book[1]**	**Key Findings**
Zeinab Azarmand and Ensiyeh Neishabouri	2009	Location Allocation Problem	Facility Location: Concepts, Models, Algorithms and Case Studies (Book)	This paper addresses the location-allocation problem, which aims to minimize transportation costs by optimally placing facilities. It reviews numerous algorithms developed since 1963, including branch-and-bound, simulated annealing, and variable neighborhood search, particularly for large-scale problems.
Derya Celik Turkoglu and Mujde Erol Genevois	2020	A comparative survey of service facility location problems	Annals of Operations Research (Journal)	This paper surveys 90 papers on service facility location problems since 2000, presenting a taxonomy based on 19 characteristics to aid researchers and practitioners, categorizing problems by application fields and highlighting future research directions

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Mark S. Daskin and Kayse Lee Maass	2015	The p-Median Problem	Location Science (Book)	The p-median problem, central to discrete location modeling, is NP-hard on general graphs but solvable in polynomial time on a tree, with various algorithms and metaheuristics, including tabu search and genetic algorithms, applied for solutions, and multi-objective extensions discussed.
Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama	2015	Introduction to Location Science	Location Science (Book)	The book introduces modern Location Science, tracing its roots, identifying linked disciplines, providing successful application examples, detailing the volume's purpose and structure, and offering suggestions for organizing location courses for various audiences.
C.S. ReVelle, H.A. Eiselt, and M.S. Daskin	2008	A bibliography for some fundamental problem categories in discrete location science	European Journal of Operational Research (Journal)	This paper presents a taxonomy of facility location modeling, providing an annotated bibliography of recent papers on median and plant location models, as well as center and covering models, and concludes with a summary and outlook

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Charles S. ReVelle and Ralph W. Swain	1970	Central Facilities Location	Geographical Analysis (Journal)	This paper addresses the problem of central facilities location, focusing on minimizing the average distance or time traveled per person by designating a specified number of communities as centers, and discusses various constraints, optimization methods, and practical applications such as clinics and warehouses
Lev Kazakovtsev	2013	Random Search Algorithm for the p-Median Problem	Informatica (Journal)	This article introduces the p-Median problem referring to the original problem Weber solves in the early 20th century
Michele Barbato et al.	2023	Node based compact formulations for the Hamiltonian p-median problem	Networks (Journal)	Provides a more specified formulation of the p-Median problem
Pasquale Avella, Antonio Sassano, and Igor Vasil'ev	2007	Computational study of large-scale p-Median problems	Math. Program. (Journal)	Provides a more specified formulation of the p-Median problem
Alfred Weber	1922	Ueber den standort der industrien	Ueber den Standort der Industrien (Book)	Weber introduces in this book the foundational single warehouse problem which serves as a baseline for further development in the field of LA problems

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Frank Plastria	2001	Static competitive facility location: An overview of optimisation approaches	European Journal of Operational Research (Journal)	This paper provides an overview of research, models, and literature on optimization approaches for the problem of optimally locating one or more new facilities in environments where competing facilities already exist. It serves as reference literature for LA problems in continuous spaces.
Reza Zanjirani Farahani, Maryam Abedian, and Sara Sharahi	2009	Dynamic Facility Location Problem	Facility Location: Concepts, Models, Algorithms and Case Studies (Book)	This book discusses facility location as a strategic management decision, typically based on current parameters such as population, infrastructure, and service requirements. It reviews key models in location theory, including single/multi-facility location, covering, P-median, and P-center problems, noting the challenges in solving these problems and the focus on static and deterministic formulations that often fail to capture the complexities of real-world location issues.

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Andreas Klose and Andreas Drexl	2005	Facility location models for distribution system design	European Journal of Operational Research (Journal)	This article addresses the strategic importance of distribution system design for companies, focusing on the core topics of facility location and customer allocation. It reviews various model formulations and solution algorithms, highlighting their fundamental assumptions, mathematical complexity, and computational performance, with a summary of continuous location models, network location models, mixed-integer programming models, and their applications.
Susan Hesse Owen and Mark S. Daskin	1998	Strategic facility location: A review	European Journal of Operational Research (Journal)	This paper reviews literature on the strategic nature of facility location decisions, focusing on stochastic and dynamic aspects. It discusses how model formulations and solution approaches address uncertainties in future events, timing issues, and scenario planning, with applications across various industries. The review highlights the importance of selecting robust facility sites that remain profitable over time despite changes in environmental factors, population shifts, and market trends.

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Elena Fernandez and Mercedes Landete	2015	Fixed-Charge Facility Location Problems	Location Science (Book)	This chapter addresses Fixed-Charge Facility Location Problems, a fundamental topic in Location Science, involving decisions on where to establish facilities and how to allocate user demand to these facilities. It provides an overview of key elements in modeling and solving these problems, including modeling hypotheses, formulation characteristics and their interrelations, domain properties, and suitable solution techniques, highlighting potential applications in various contexts.
O. Kariv and S. L. Hakimi	1979	An Algorithmic Approach to Network Location Problems. II: The p-Medians	SIAM Journal on Applied Mathematics (Journal)	This chapter demonstrates that the problem of finding a p-median in a network is NP-hard, even for simple structures such as planar graphs with a maximum vertex degree of 3
Nenad Mladenovic et al.	2007	The p-median problem: A survey of metaheuristic approaches	European Journal of Operational Research (Journal)	This paper surveys the p-median problem, a fundamental model in discrete location theory classified as NP-hard, and examines advances in solving it using recent procedures based on metaheuristic rules

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
W. Michiels, E. H. L. Aarts, and J. Korst	2018	Theory of Local Search	Handbook of Heuristics (Book)	This chapter discusses local search as a widely used heuristic method for solving combinatorial optimization problems, particularly NP-hard problems, by trading off solution quality against computation time, with well-known approaches including iterative improvement, simulated annealing, and tabu search.
Michael O. Ball	2011	Heuristics based on mathematical programming	Surveys in Operations Research and Management Science (Journal)	This paper surveys heuristics using mathematical programming, covering methods that decompose problems into subproblems, improvement algorithms for better feasible solutions, branch-and-bound for approximations, and solving relaxations to find good solutions
R.A. Whitaker	1983	A Fast Algorithm For The Greedy Interchange For Large-Scale Clustering And Median Location Problems	INFOR: Information Systems and Operational Research (Journal)	This paper serves as reference for greedy search algorithms
S. Salhi and R.A. Atkinson	1995	Subdrop: A modified drop heuristic for location problems	Location Science (Journal)	This paper serves as reference for stingy search algorithms
S. Salhi	1997	A Perturbation Heuristic for a Class of Location Problems	The Journal of the Operational Research Society (Journal)	This paper serves as reference for constructive heuristics

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
F. E. Maranzana	1964	On the Location of Supply Points to Minimize Transport Costs	OR (Journal)	This paper serves as reference for alternate heuristics
Michelle Hribar and Mark S. Daskin	1997	A dynamic programming heuristic for the P-median problem	European Journal of Operational Research (Journal)	This paper introduces dynamic programming as a possible solution approach for the p-median problem
J.E. Beasley	1993	Lagrangean heuristics for location problems	European Journal of Operational Research (Journal)	This paper introduces a demand partitioning method that reduces aggregation errors in p-median problems, tested with Costa Rica data, and outperforms existing methods while requiring less computational effort than using unaggregated data.
Stefan Voß	2008	Metaheuristics	Encyclopedia of Optimization (Book)	This chapter serves as orientation for metaheuristics. It introduces the concepts and shows how it can applied to existing problems
Osman Alp, Erhan Erkut, and Zvi Drezner	2003	An Efficient Genetic Algorithm for the p-Median Problem	Ann. Oper. Res. (Journal)	This paper demonstrates how a genetic search algorithms can be used to solve facility location problems
Enrique Dominguez Merino and José Muñoz Perez	2002	An Efficient Neural Network Algorithm for the p-Median Problem	Advances in Artificial Intelligence — IBERAMIA 2002 (Book)	This paper presents a neural network model and new formulation for the p-median problem, demonstrating its effectiveness for small-scale problems (less than 100 points) and showing that it can produce good solutions for large-scale problems in a few seconds, outperforming conventional heuristic methods

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Kailash Lachhwani	2020	Application of neural network models for mathematical programming problems: A state of art review	Arch. Comput. Methods Eng. (Journal)	This article introduces the concept of neural networks for mathematical programming problems. It serves as a theoretical foundation and orientation for concepts related to neural networks.
Laura I. Burke and James P. Ignizio	1992	Neural networks and operations research: An overview	Computers and Operations Research (Journal)	This paper introduces the use of neural networks as an alternative method for solving common OR problems such as resource allocation, classification, prediction/estimation, and clustering, highlighting its potential advantages, including parallel processing capabilities, and emphasizing its relevance for OR analysts
Kurt Hornik, Maxwell Stinchcombe, and Halbert White	1989	Multilayer feedforward networks are universal approximators	Neural Networks (Journal)	This paper rigorously establishes that standard multilayer feedforward networks with a single hidden layer using arbitrary squashing functions are universal function approximators, capable of approximating any Borel measurable function between finite dimensional spaces to any desired accuracy.

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Chenguang Wang et al.	2023	Solving uncapacitated P-Median problem with reinforcement learning assisted by graph attention networks	Appl. Intell. (Journal)	This paper introduces the first RL-based method using Multi-Talking-Heads Graph Attention Networks to solve the uncapacitated P-Median Problem, demonstrating superior performance in solution quality and time efficiency compared to traditional methods, and highlighting the impact of data distribution differences on final performance.
Shiqing Liu, Xueming Yan, and Yaochu Jin	2023	End-to-End Pareto Set Prediction withÂ Graph Neural Networks forÂ Multi-objective Facility Location	Evolutionary Multi-Criterion Optimization (Book)	This paper addresses the multi-objective facility location problem (MO-FLP), aiming to minimize cost and maximize system reliability. It introduces a learning-based approach using graph neural networks to predict the distribution probability of the Pareto set, allowing for efficient sampling of non-dominated solutions. Experimental results demonstrate that this method matches the performance of traditional multi-objective evolutionary algorithms while significantly reducing computational costs.

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Enrique Dominguez and Jose Munoz	2008	Computers and Operations Research (Journal)	A neural model for the p-median problem	This paper proposes a competitive recurrent neural network for solving the p-median problem, incorporating constraints into the neural architecture to ensure feasible solutions without tuning parameters, and demonstrating its efficiency and effectiveness compared to traditional heuristics, with potential integration into GIS software for real-time spatial query support.
Ashish Kumar Shakya, Gopinatha Pillai, and Sohom Chakrabarty	2023	Reinforcement learning algorithms: A brief survey	Expert Systems with Applications (Journal)	This review provides an overview of Reinforcement Learning (RL), a machine learning technique for sequential decision-making in complex problems, highlighting its trial-and-error inspiration and capability to autonomously learn optimal policies. It covers fundamental model-free RL algorithms, DRL algorithms for complex tasks, and briefly discusses model-based and multi-agent RL approaches, concluding with promising research directions in the field.
Manuel Schneckenreither and Stefan Haeussler	2019	Reinforcement Learning Methods for Operations Research Applications: The Order Release Problem	Machine Learning, Optimization, and Data Science (Book)	This paper proposes a DRL-based adaptive order release mechanism for manufacturing, optimizing release times to improve flow times and performance. A two-stage flow-shop simulation shows this approach outperforms traditional methods.

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Ching Pui Wan, Tung Li, and Jason Min Wang	2023	RLOR: A Flexible Framework of Deep Reinforcement Learning for Operation Research	Preprint (Journal)	This paper introduces RLOR, a flexible framework for DRL in operations research. By re-implementing and training the Attention Model with Proximal Policy Optimization (PPO), it achieves an 8x speedup in solving vehicle routing problems, integrating recent advances in RL
Lina Yu et al.	2021	Reinforcement learning approach for resource allocation in humanitarian logistics	Expert Systems with Applications (Journal)	This paper proposes a Q-learning algorithm for disaster relief resource allocation, optimizing efficiency, effectiveness, and equity. The algorithm outperforms dynamic programming in efficiency and heuristic methods in accuracy, providing near-optimal solutions by adjusting training episode values.
Matthias Klar, Moritz Glatt, and Jan C. Aurich	2021	An implementation of a reinforcement learning based algorithm for factory layout planning	Manufacturing Letters (Journal)	This paper presents a RL approach for automated factory layout planning using Double Deep Q Learning. The algorithm optimizes layouts based on transportation time, solving an allocation problem with four functional units in 8,000 training episodes and demonstrating promising potential for broader applications.

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Yimo Yan et al.	2022	Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities	Transportation Research Part E: Logistics and Transportation Review (Journal)	This paper reviews RL applications in logistics and supply chain management, highlighting Q-learning as the most popular method and increased focus on urban logistics due to E-commerce. It also discusses current challenges and potential future research directions.
Qi Wang and Chunlei Tang	2021	Deep reinforcement learning for transportation network combinatorial optimization: A survey	Knowledge-Based Systems (Journal)	This paper reviews DRL for NP-hard combinatorial optimization problems like the traveling salesman and vehicle routing problems. It explores state-of-the-art techniques, compares algorithm performance, and identifies challenges and future research directions in this field.
Kai Arulkumaran et al.	2017	Deep Reinforcement Learning: A Brief Survey	IEEE Signal Processing Magazine (Journal)	This survey explores DRL , covering key algorithms like DQN, TRPO, and asynchronous advantage actor critic, and its applications in video games and robotics. It highlights DRL's potential to advance AI and concludes with current research areas.
Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson	2017	Dynamic Facility Location via Exponential Clocks	ACM Trans. Algorithms (Journal)	This article introduces the dynamic deterministic facility location problem. It serves as an orientation for the mathematical formulation of the problem.

Continued on next page

Table A.1 – continued from previous page

Author(s)	Year	Title	Journal/Book	Key Findings
Richard E. Rosenthal, John A. White, and Donovan Young	1978	Stochastic Dynamic Location Analysis	Management Science (Journal)	This research introduces stochastic decision processes into location analysis, using an infinite-horizon Markov decision chain to minimize relocation and interaction costs for dynamic facility relocation. It also reports methods for solving large problems. It serves as the foundation for dynamic location-allocation problems.
Reza Zanjirani Farahani, Maryam Abedian, and Sara Sharahi	2009	Dynamic Facility Location Problem	Facility Location: Concepts, Models, Algorithms and Case Studies (Book)	This chapter discusses facility location as a strategic management decision, often based on current parameters like population and infrastructure. It reviews research on various models, including single/multi-facility location, covering, P-median, and P-center problems. While much work has focused on static and deterministic formulations, these do not fully capture the complexities and dynamic nature of real-world location problems

Category	Heuristic	Literature found
Traditional	Classic Heuristics	Nenad Mladenovic et al. (2007), Salhi (1997), Whitaker (1983), Salhi and Atkinson (1995), GalvÃ£o (1980), Salhi (1997), Michiels, Aarts, and Korst (2018), Maranzana (1964), Hansen and N. Mladenovic (1997), Ball (2011), Hribar and M. S. Daskin (1997), Beasley (1993), Bowerman, Calamai, and Brent Hall (1999)
	Metaheuristics	Voß (2008), Nenad Mladenovic et al. (2007), Salhi (2002)), Alp, Erkut, and Drezner (2003), Rosing et al. (1998)
Modern	NN heuristics	Burke and Ignizio (1992), Dominguez Merino and Muñoz Perez (2002), Dominguez and Munoz (2008), Matis and Tarabek (2023), C. Wang et al. (2023), Liu, X. Yan, and Yaochu Jin (2023)
	RL heuristics	Schneckenreither and Haeussler (2019), C. Wang et al. (2023), Guo, Xu, and Yaohui Jin (2023), Yu et al. (2021), Klar, Glatt, and Aurich (2021)

Table A.2.: Literature Review Table of literature found for each heuristic. Modern heuristics are much more recent than traditional heuristics. Often in the last four years (2020-2024). Especially RL heuristics are often also still preprints and not reviewed

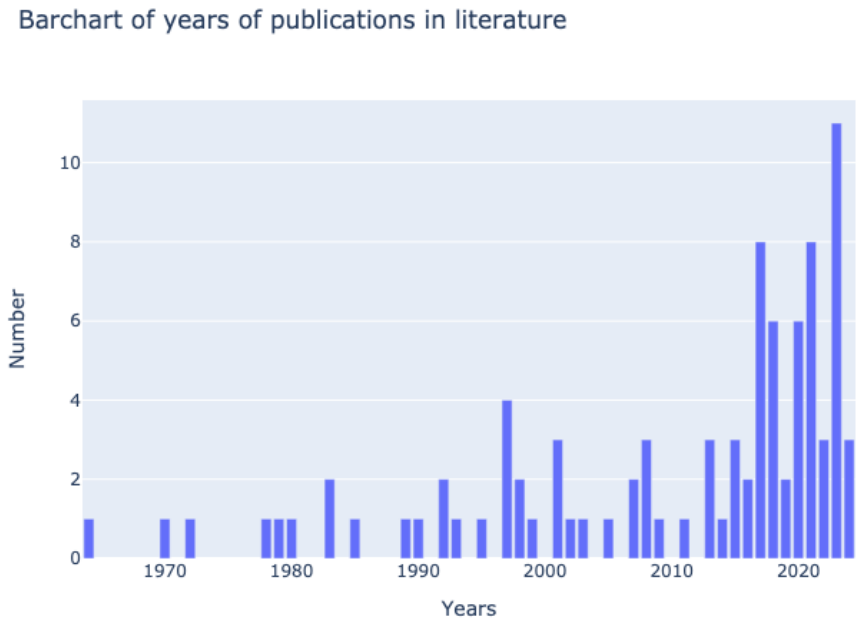


Figure A.1.: Distribution of publication years for the literature used in this thesis. The majority of the literature is very recent, reflecting the emerging nature of research on RL and GNN for LA problems. Since these topics are relatively unexplored, particularly in the context of GNN-based RL for LA problems, most relevant studies have been published in the past few years. This distribution highlights the identified research gap, indicating that literature in this area is either sparse, not thoroughly reviewed, or newly emerging

B. Oversmoothing in Graph Neural Networks

Oversmoothing in GNNs refers to the phenomenon where, as the depth of the network increases, i.e. more layers and message-passing steps are added, the node embeddings across the graph become increasingly similar (Keriven 2024).
In each layer of a GNN, the node embeddings $h_v^{(l)}$ are updated by aggregating information from their neighboring nodes. As more layers are added, this aggregation process incorporates information from nodes that are progressively further away. Eventually, a point is reached where every node's embedding contains information from almost every other node in the graph.
Beyond this point, further layers do not introduce new information but merely aggregate what has already been combined in previous layers. As a result, the embeddings of all nodes start to converge, becoming nearly identical. If the number of layers continues to increase indefinitely, the embeddings of all nodes will eventually become the same. This can be formally expressed as:

$$\lim_{l \to \infty} h_v^{(l)} = h_u^{(l)} \quad \forall u, v \in V$$

This convergence causes the GNN to lose its ability to distinguish between different nodes, which is detrimental to its performance on tasks such as node- and graph-level predictions. Hence, the GNN loses its expressive power (Sun (2022)). *Figure B.1* undermines this concept. Oversmoothing prevents GNNs from being too deep which is contrary to other neural network architectures where it is usually the deeper the network the better the performance. Therefore, the number of layers of a GNN should be chosen carefully and with consideration of the size of the graph or more specifically with consideration of the largest k-hop neighborhood for each node.

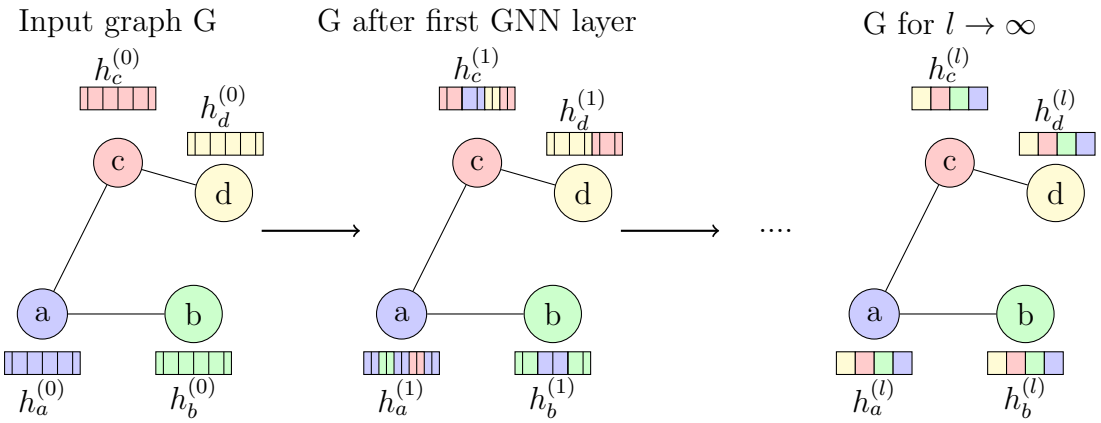


Figure B.1.: Oversmoothing of node embeddings. As $l \to \infty$, the node embeddings converge, displayed by the same color structure at each node embedding.

C. Replicated State Machine Simulation Overview

One of the contributions of this thesis is the simulation of the RSM, implemented using the standard OpenAI Gym interface (Brockman et al. (2016)). This simulation is built around a base class, *NetworkEnvironment*, which manages the system dynamics. The class simulates stochastic client movements based on time-of-day distributions and models client request quantities using the Dirichlet distribution, with weights adjusted according to the time of day (e.g., if the peak time in Asia is around 06:00 am, and the current time is 08:00 pm, Asia will have a lower weight). Each region has a peak time window from hour x_1 to hour x_2. To calculate the weights w for these probabilistic properties, the following value is computed:

$$d = \min\left(\left|\frac{x_1 + x_2}{2} - t\right|, \left|24 + \frac{x_1 + x_2}{2} - t\right|\right)$$
$$w = \frac{1}{d}$$

where t is the current hour. Given a set of clients $C = \{c_1, c_2, \ldots, c_n\}$ and regions $R = \{r_1, r_2, \ldots, r_k\}$, each with distinct peak times (representing possible regions where clients can move dynamically), we compute the weights $W = \{w_1, w_2, \ldots, w_k\}$ for each region. A new region for each client $c_i \in C$ is then sampled based on these weights:

$$p_i = \frac{w_i}{\sum_{j=1}^{k} w_j}$$
$$r_{c_i} \sim \text{Categorical}(p_1, p_2, \ldots, p_k)$$

This process determines a new region for each client at each timestep t. The number of requests per client q_{c_i} is then sampled using a Dirichlet distribution with the weights W. Formally:

$$Q = \{q_{c_1}, q_{c_2}, \ldots, q_{c_n}\} \sim \text{Dirichlet}(W)$$

This approach generates stochastic client dynamics, with varying requests throughout the day. Additionally, the *NetworkEnvironment* class has access to an internal map of latency ranges between data centers and clients, corresponding to the new regions. These latencies are also sampled to construct a dynamic graph at time t, reflecting the evolving client positions. By tracking changes in the data center configuration, the class can compute the reward at each step t.

The *NetworkEnvironment* class thus simulates client movements, computes rewards, and returns the current graph instance at each timestep t.

The *NetworkEnvGym* class implements the gym interface, taking a *NetworkEnvironment* instance as an attribute. This provides access to all public methods of the class, which are used to implement the *step(action a)*, *reset()*, and *render()* methods. These methods follow the OpenAI Gym guidelines, ensuring a seamless and modular interaction with the environment class, which simplifies applying various algorithms to the problem using a consistent interface.

The *TorchGraphObservationWrapper* class converts the graph instance state at each timestep t into a torch geometric data type, easily processed by PyTorch and torch geometric neural networks. This wrapper class takes the *NetworkEnvGym* as input and transforms the state output each time *step(action a)* is called, enabling seamless integration with applications without the need for additional data type transformations.

The connection between these three classes is illustrated in the following, highly simplified, UML diagram. Figure B.1 displays the visualized environment at the time of the day $t = 1$. This is done via the *visualize()* or the *render()* method by the respective class. This contribution makes it possible to evaluate single actions visually, improving understanding of the problem and consequences of actions. For a more detailed overview of the class structure and the visualization process, refer to the *repository* for this thesis.

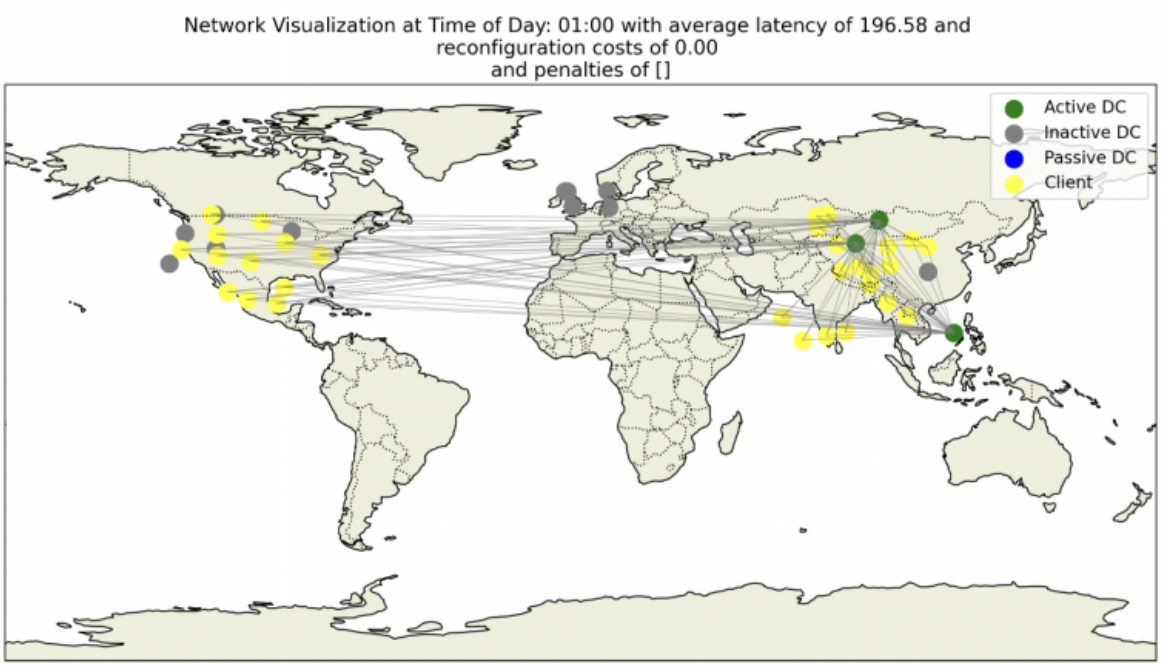
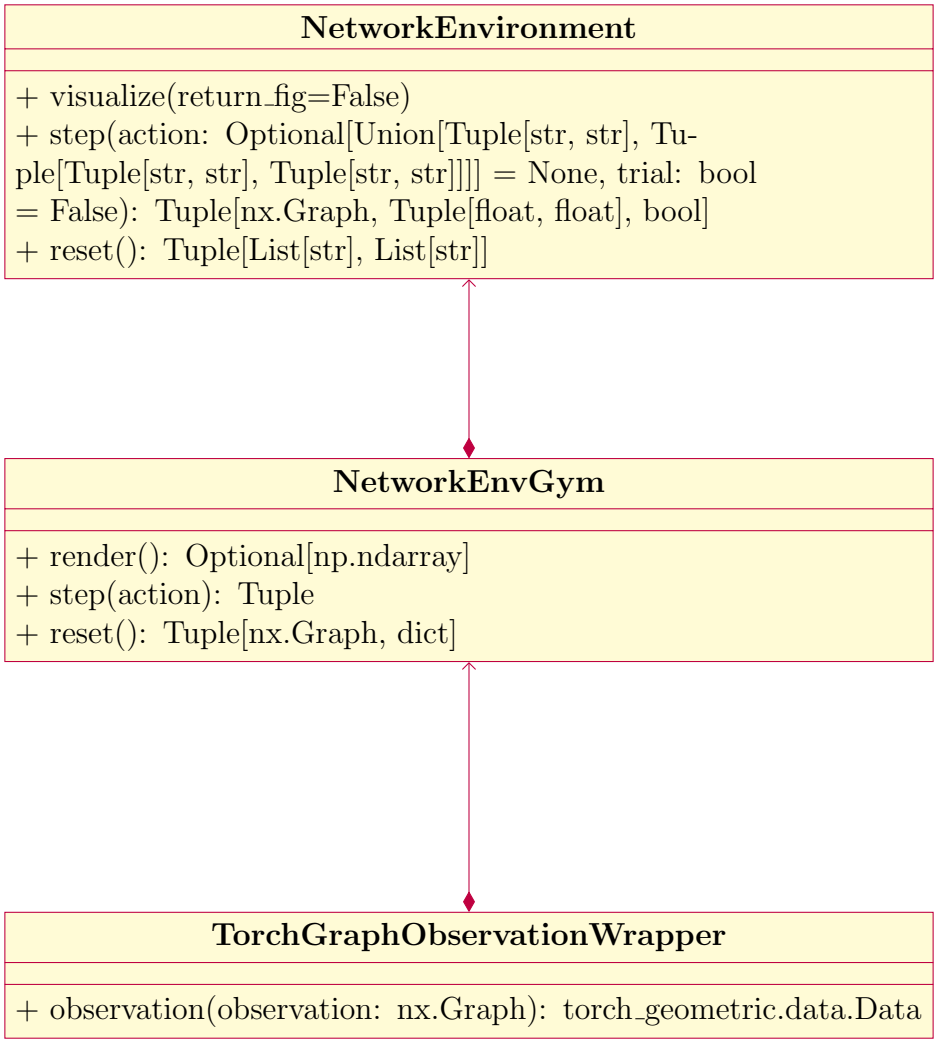


Figure C.1.: Simulation visualization of the environment. Yellow client locations, grey non-active locations, green active locations, and blue passive locations. Edges are latencies between data centers and clients, but also data centers and other data centers

Figure C.2.: Simplified UML diagram for the RSM simulation workflow



D. Foundations of Deep Reinforcement Learning

D.1. Reinforcement Learning Foundation

To find optimal policies, the state transition probabilities are used as shown by equations 10 and 11. This approach is then called *Dynamic Programming* (DP). DP is an algorithm that computes the agent's optimal policy when the complete environment dynamics are known in the MDP framework. One important method of DP is, that it makes use of the current state-value estimations for the next state s' to update the state-value estimation for the current state s. However, DP is impractical for large RL problems with continuous state and action spaces due to its high computational demands and the need for complete environment knowledge. Despite this, understanding DP is crucial for grasping other RL methods. DP relies on Bellman equations and involves two main processes: policy evaluation, which estimates state values for a given policy, and policy improvement (Shakya, Pillai, and Chakrabarty (2023a)).

In practical situations, the state transition probabilities may not be accessible for all possible states. Model-free RL methods bypass the need for a complete environment model by using agent experience to directly learn the best possible value functions or policies, often through a trial-and-error process. Model-free approaches typically require more data samples from agent-environment interactions compared to model-based algorithms. However, even model-based algorithms often rely on model-free methods to help build the environment model. Model-free RL methods are particularly advantageous for tackling complex problems where constructing an accurate environment model is difficult. (Shakya, Pillai, and Chakrabarty (2023a)). Common model-free approaches are:

- **Monte Carlo Methods**: Monte Carlo (MC) is a model-free approach where an agent learns from experience data of episodic tasks, either in simulated or real-world environments. It averages the returns from agent-environment interactions to solve RL problems. A model may be used to simulate the environment, but the policy and actions are not optimized based on this

model. In MC, value and policy estimates are updated after each episode is completed.

- **Temporal Difference Learning**: Temporal Difference (TD) learning is a foundational concept in many key RL (RL) algorithms. It blends the strengths of both DP and MC methods. Like MC, TD learning is a model-free approach, meaning it doesn't require a model of the environment. However, similar to DP, TD learning updates value estimates based on previous estimates rather than relying solely on complete episodes. In a straightforward MC update, a whole episode with all the rewards is sampled to get a full return trajectory. This return over all sampled steps is used as the target for the update. Whereas, TD learning, samples only a few steps, often just one step such that it updates the target with value estimates to refine predictions. Common TD algorithms are *SARSA* and *Q-Learning*

- **Value Function Approximation**: In methods like MC and TD, value functions for all states are stored in memory. Since a state is defined by a specific arrangement of observation features, even small changes in these features create new states. For problems with large state and action spaces, learning value functions for every possible state becomes impractical due to the extensive memory and computational resources required.
  To address this, Value Function Approximation (VFA) is used, introducing generalization to RL. VFA assigns similar value estimates to states with similar features, allowing for an approximation of optimal policies and state values. This approach significantly speeds up computation. Common methods for VFA include linear function approximators, neural networks, nearest neighbor, and decision trees, with gradient-based optimization often used to improve results. Linear function approximators and neural networks are particularly effective choices for parameterized VFA, where value functions are represented by parameterized functions with a weight vector.

Linear VFA requires carefully designed input features, which can be difficult to select and often doesn't perform well without the right features. It also faces challenges like correlations between training samples and non-stationary targets. A better alternative is using deep neural networks (DNNs) for function approximation, which can directly use states without needing explicit feature specification and can represent complex nonlinear functions.
The success of TD-Gammon in 1992, which combined ANNs and RL, marked a significant advance. However, early RL methods were limited by computational constraints. The rise of deep learning and DNNs has overcome these limitations, enabling the automatic extraction of features from high-dimensional data and solving complex tasks like video game playing, making DRL a powerful tool for ad-

vanced problems (Shakya, Pillai, and Chakrabarty (2023a)).

D.2. Value Optimization Methods

In general, there are two DRL approaches. Value optimization and policy optimization methods. In value optimization-based methods, an initial value function is randomly assigned to each state. The value function is then iteratively updated using learning data until the optimal value function is determined. The optimal policy, which aligns with the optimal value function, is implicitly updated during this process. This subsection explores various value optimization-based DRL approaches.
Very popular is the algorithm of Deep Q-Learning (DQN). Usually, Q-Learning creates a state-action table for estimating the optimal policy, but this approach struggles with large state and action spaces. DQN, introduced by Google DeepMind in 2013 and improved in 2015, address this by using a DNN to approximate Q-value functions. DQN gained significant attention in AI because it successfully learned control policies directly from high-dimensional sensory inputs, like images, using RL. In DQN, states are for example input as images, and the network outputs estimated Q-values for all possible actions, enabling easy selection of the optimal action. Refer to Figure C.1 for a visual depiction. Algorithm 1 shows how DQN works. It makes use of experience replay memory, which is a storage mechanism used in RL algorithms like DQN. It stores past experiences, which include the state, action, reward, and next state tuples from the agent's interactions with the environment. The algorithm randomly samples these stored experiences during training to break the correlation between consecutive data points, leading to more stable and efficient learning (Mnih, Kavukcuoglu, Silver, Graves, et al. (2013)).

- **Initialization**: The algorithm starts by initializing the experience replay memory $\mathcal{D}$ and the Q-value function approximator (a neural network) with random weights θ.
- **Episode Loop**: The algorithm runs for multiple episodes, where each episode consists of multiple steps or time steps.
- **State Initialization**: At the beginning of each episode, the initial state $\{s_1\}$ is observed, and it is preprocessed into a feature representation ϕ_1.
- **Action Selection**: At each time step t, an action a_t is selected based on an ϵ-greedy policy. This means that with probability ϵ, a random action is

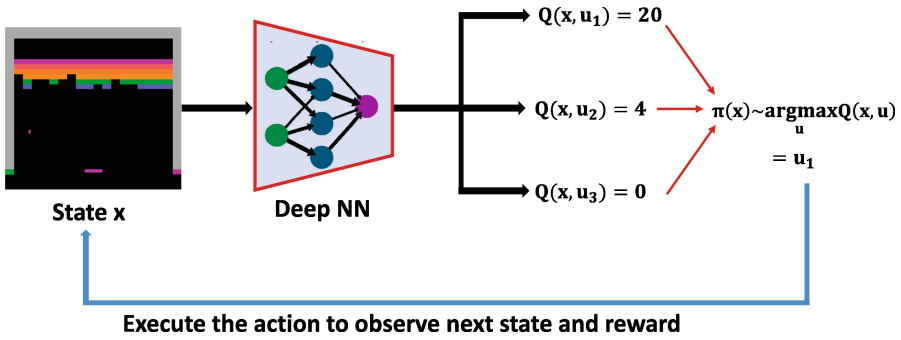


Figure D.1.: DQN depiction. State x is an image with an action space A of three possible actions u_1, u_2, u_3. The DQN learns estimated Q-Values $q^{\pi}(s_t, u) \quad \forall u \in A$ for each action and state pair. The optimal policy can then be easily selected by taking the maximum Q-Value (Shakya, Pillai, and Chakrabarty (2023a)).

chosen, and with probability $1 - \epsilon$, the action that maximizes the Q-value function is selected.

- **Environment Interaction**: The chosen action is executed in the environment, resulting in a reward r_t and a new state s_{t+1}.
- **State Transition Storage**: The observed transition $(\phi(s_t), a_t, r_t, \phi(s_{t+1}))$ is stored in the experience memory $\mathcal{D}$.
- **Experience Replay**: A random minibatch of stored transitions is sampled from the experience memory $\mathcal{D}$ to break correlation between consecutive experiences and stabilize training.
- **Q-Value Update**: The target Q-value y_j is calculated using the Bellman equation, and the network's weights are updated by performing a gradient descent step to minimize the difference between the predicted Q-values and the target Q-values.
- **Iteration**: Steps 4-8 are repeated for all steps in the episode, and the process is iterated over all episodes.

This approach enables the algorithm to learn control policies directly from high-dimensional input data, such as images but also graphs using GNNs, by approximating the optimal Q-values for each state-action pair using a deep neural network.

Algorithm 1 DQN Algorithm

Initialize: Experience memory $\mathcal{D}$ and Q-values with randomly selected weights θ
for all episodes $e = 1, 2, \ldots$ **do**
    Initialize state sequence $\{s_1\}$ and preprocessed state sequence $\phi_1 = \phi(s_1)$
    for all steps t in each episode **do**
        With probability ϵ select a random action a_t
        Otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
        Take action a_t in the simulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t$, preprocess s_{t+1} as ϕ_{t+1}
        Store transition $(\phi(s_t), a_t, r_t, \phi(s_{t+1}))$ in memory $\mathcal{D}$
        Sample random minibatch of transitions $(\phi(s_j), a_j, r_j, \phi(s_{j+1}))$ from memory $\mathcal{D}$
        Set $y_j = r_j + \gamma \max_{a'} Q(\phi(s_{j+1}), a'; \theta^-)$
        Perform a gradient descent step on $(y_j - Q(\phi(s_j), a_j; \theta))^{(2)}$ with respect to the weights θ
    end for
end for

D.3. Policy Optimization Methods

The previously discussed optimization method uses value functions to estimate the optimal policy. In contrast, policy optimization methods directly estimate the optimal policy without relying on value functions. While value functions can still help optimize policy parameters, they aren't used for action selection. Policy-based methods are particularly useful for continuous space problems, where value function-based methods become computationally expensive. The most fundamental approach is the one of *Policy Gradient* (PG). PG methods optimize a policy by parameterizing it as a function $\pi_\theta(a|x_k)$ that determines the probability of selecting an action a at time step k given a state x_k, with parameters θ. The policy parameters are adjusted using a performance measure $J(\theta)$, such as the value of the start state for episodic tasks or the average reward rate for continuous tasks. Unlike value-based methods, where action selection probabilities can change drastically with small updates, PG methods offer smoother and more stable convergence by adjusting policy parameters directly.

In PG methods, the policy must be differentiable with respect to its parameters θ, and typically employs stochastic policies to ensure adequate exploration, represented as $\pi_\theta(a|x) > 0$ for all a and x. The challenge in PG methods is that the performance measure $J(\theta)$ depends on both the action selection probabilities and the state distributions, which are influenced by the policy parameters. Since state

distributions are often unknown and environment-dependent, directly estimating the gradient $\nabla_\theta J(\theta)$ of the performance measure is difficult.
The policy gradient theorem provides an analytical expression for the gradient of the performance measure with respect to the policy parameters, represented as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \right]$$

where $\mathbb{E}_{\pi_\theta}$ denotes the expectation under the policy π_θ, and $Q^{\pi_\theta}(s,a)$ is the action-value function.
This expression does not require the gradient of state distributions, which makes it feasible to compute. The REINFORCE algorithm, based on this theorem, updates the policy parameters using the stochastic gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

where α is the learning rate. By using this approach, the policy is iteratively improved, leading to an optimal policy. The REINFORCE algorithm is shown in Algorithm 2. It can be explained as follows:

- **Initialization:** The algorithm begins by initializing the policy parameters θ. These parameters define the policy $\pi_\theta(a|s)$ that will be optimized.
- **Episode Generation:** For each episode, a sequence of states $\{s_0, s_1, \ldots, s_K\}$, actions $\{a_0, a_1, \ldots, a_{K-1}\}$, and rewards $\{r_1, r_2, \ldots, r_K\}$ is generated by following the current policy π_θ.
- **Return Calculation:** For each step k in the episode, the algorithm calculates the return G, which is the cumulative discounted reward starting from that step:

  $$G \leftarrow \sum_{t=k+1}^{K} \gamma^{t-k-1} r_t$$

  where γ is the discount factor.
- **Policy Update:** The policy parameters θ are updated using the policy gradient:

  $$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_k|s_k) G$$

  Here, α is the learning rate, and $\nabla_\theta \log \pi_\theta(a_k|s_k)$ is the gradient of the log-probability of the action taken, with respect to the policy parameters. This update rule shifts the policy in the direction that increases the expected return.

- **Iteration:** The process is repeated for all steps in each episode and over multiple episodes to gradually improve the policy.

(Shakya, Pillai, and Chakrabarty (2023a), Sutton and Barto (2018d)).

Algorithm 2 REINFORCE: MC Policy Gradient Control for Episodic Task

Require: A differentiable parameterized policy $\pi_\theta(a|s)$ and learning rate α

Initialize: Policy parameter θ
for all episodes **do**
  Generate an episode $\{s_0, a_0, r_1, \ldots, s_{K-1}, a_{K-1}, r_K\}$ following policy π_θ
  for all steps in the episode $k = 0, 1, \ldots, K-1$ **do**
    $G \leftarrow \sum_{t=k+1}^{K} \gamma^{t-k-1} r_t$ ▷ Compute the return from step k
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_k|s_k) G$ ▷ Update policy parameters
  end for
end for

Unlike the REINFORCE algorithm, which can be slow due to its Monte Carlo approach, actor-critic methods utilize TD learning, allowing for online and incremental updates. The critic's bootstrapped value estimates provide more stable updates to the actor's policy, making the method effective for continuous space problems.
The actor-critic method improves over traditional policy gradient methods by using the critic's value function as a baseline, which stabilizes and speeds up the learning process. The updates follow:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s)(r + \gamma V^\pi(s') - V^\pi(s))$$

where $r + \gamma V^\pi(s') - V^\pi(s)$ is the TD error used to update both the actor and critic (Shakya, Pillai, and Chakrabarty (2023a)). Algorithm 3 demonstrates the actor-critic approach:

- **Initialization:** Initialize the policy parameters θ and value function parameters ω randomly. These parameters define the policy $\pi_\theta(a|s)$ and the state value function $V_\omega(s)$, respectively. Set learning rates α_θ and α_ω for updating the policy and value function.

- **Episode Loop:** The algorithm runs over multiple episodes, each starting with an initial state s_0. For each step within an episode, the agent interacts with the environment until a terminal state is reached.

- **Action Selection:** At each time step t, the agent selects an action a_t based on the current policy $\pi_\theta(a|s_t)$.
- **Environment Interaction:** The agent takes action a_t and observes the reward r_{t+1} and the next state s_{t+1}.
- **TD Error Calculation:** Compute the Temporal Difference (TD) error δ_t using:
$$\delta_t \leftarrow r_{t+1} + \gamma V_\omega(s_{t+1}) - V_\omega(s_t)$$
where γ is the discount factor.
- **Critic Update:** Update the value function parameters ω using the TD error:
$$\omega \leftarrow \omega + \alpha_\omega \delta_t \nabla_\omega V_\omega(s_t)$$
This step adjusts the critic to better estimate the value function based on new information.
- **Actor Update:** Update the policy parameters θ using the TD error:
$$\theta \leftarrow \theta + \alpha_\theta \delta_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$
This step adjusts the actor to improve the policy, making actions that lead to higher rewards more likely.
- **Iteration:** Steps 3-7 are repeated for all steps within the episode, and the process is iterated over multiple episodes to gradually improve the policy.

The one-step Actor-Critic algorithm effectively balances exploration (through the actor) and exploitation (through the critic) by leveraging the critic's value estimates to guide policy updates. This approach results in more stable and efficient learning in RL tasks. Numerous variations of the actor-critic framework have been developed to enhance performance and efficiency. For instance, Proximal Policy Optimization (Schulman, Wolski, et al. (2017)) and Trust Region Policy Optimization (Schulman, Levine, et al. (2017)) are advanced actor-critic methods designed to limit the divergence between successive policies, ensuring that updates do not lead to drastically different policies, which could potentially derail learning. This conservative update strategy helps maintain the policy on a favorable trajectory.

In summary, DRL is a powerful approach for handling large state and action spaces by utilizing the expressive capabilities of deep neural networks. DRL has achieved significant success in recent years, with many researchers proposing optimized algorithms based on foundational RL principles but differing in implementation details.

Algorithm 3 One-step Actor-Critic for Optimal Policy Estimation

Require: A differentiable parameterized policy $\pi_\theta(a|s)$ and state value function $V_\omega(s)$, learning rates α_θ and α_ω

Initialize: Policy parameters θ and value function parameters ω randomly
for all episodes **do**
    Initialize the starting state s_0 of the episode
    for all steps in the episode until s_t is terminal **do**
        Select action $a_t \sim \pi_\theta(a|s_t)$
        Take action a_t, observe reward r_{t+1} and next state s_{t+1}
        $\delta_t \leftarrow r_{t+1} + \gamma V_\omega(s_{t+1}) - V_\omega(s_t)$ ▷ Compute the TD error
        $\omega \leftarrow \omega + \alpha_\omega \delta_t \nabla_\omega V_\omega(s_t)$ ▷ Update critic (value function)
        $\theta \leftarrow \theta + \alpha_\theta \delta_t \nabla_\theta \log \pi_\theta(a_t|s_t)$ ▷ Update actor (policy)
    end for
end for

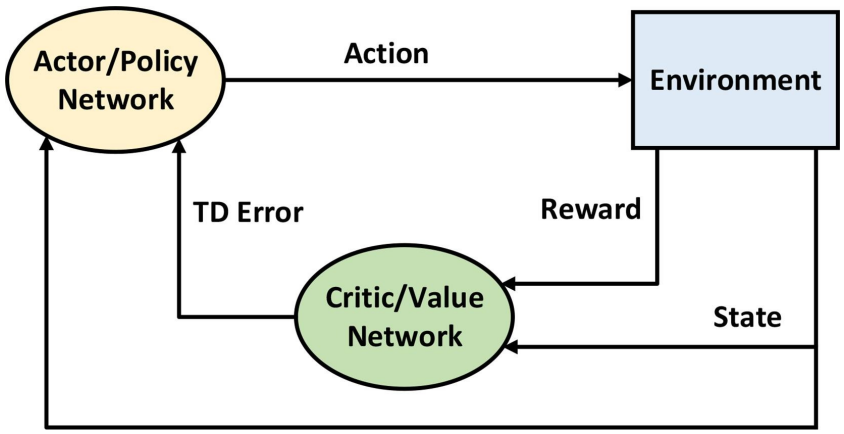


Figure D.2.: Actor-Critic approach workflow. The state and reward resulting from the action are used to compute the TD error, which is then used to update both the actor and critic networks (Shakya, Pillai, and Chakrabarty (2023a)).

Both algorithms require specialized hyperparameter settings. Hyperparameters are parameters that are not optimized by the model training but rather have to be selected via trial and error or intuition. To sum the chapter up, an overview of the important hyperparameters for both algorithms are given:

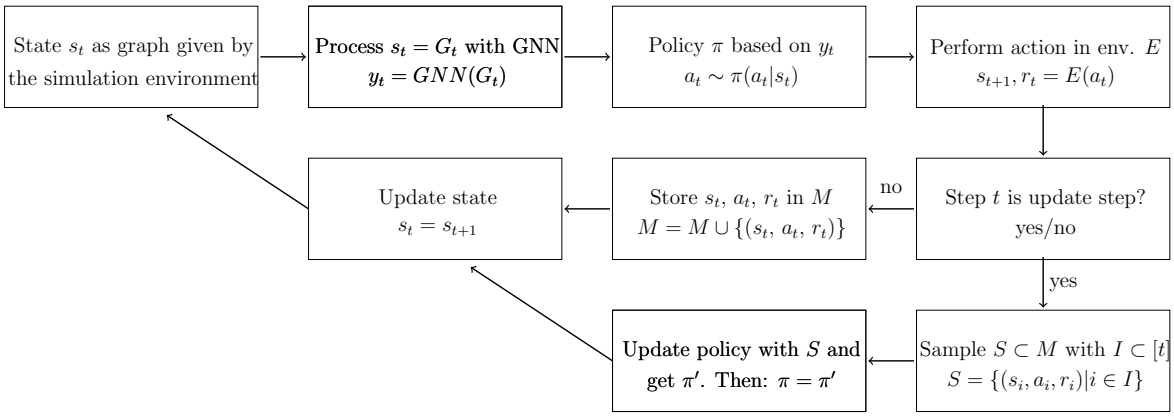
- Value optimization methods:
  - ϵ: Probability of selecting random actions rather than the maximum value action
  - ϵ_{decay}: Decay rate of ϵ for each episode that with more episodes trained the probability of selecting random actions decreases
  - ϵ_{min}: Minimum value of epsilon
  - γ: Discount factor gamma
  - α or η: Learning rate of gradient ascent/descent update
  - n_{buffer}: Size of experience replay memory
  - n_{epochs}: The number of iterations to train the model at each update step
  - b: The batch size. It is the amount of samples included in each gradient update step sampled from the experience replay memory
- Policy optimization methods:
  - ϵ_{reg}: This is an additional parameter that controls how much of entropy regularization is included in the objective function on which the actor gets updated as introduced by Mnih, Badia, et al. (2016)
  - α or η: Learning rate of gradient ascent/descent update
  - n_{epochs}: The number of iterations to train the model at each update step
  - b: The batch size. It is the amount of samples included in each gradient update step sampled from the trajectory

These are the basic parameter settings of both algorithms. More advanced versions of these algorithms (e.g. PPO, TRPO) introduce more parameters. Nevertheless, the listing here provides a sufficient overview of which foundational parameters are required. In general, it can be said that value optimization methods require more parameters and a more sophisticated approach to find the best parameter setting than policy optimization methods.

D.4. Problem Application

In this thesis, we apply the RL framework to the RSM problem, providing a theoretical justification for using RL, and by extension, DRL methods. DRL is particularly useful when dealing with large state and action spaces. Given that the underlying RSM problem is based on a dynamic graph, where each graph instance differs, the generalization capabilities of neural networks, which can assign similar values to similar states (i.e., graph instances), are highly valuable. Since GNNs are differentiable functions, they can be seamlessly integrated into any DRL algorithm. In this work, DQN and Proximal Policy Optimization were combined with GNNs to discover optimal policies using DRL. The preceding sections provide the theoretical foundation for why DRL is appropriate and justified within the traditional RL framework. Consequently, the GNN-based RL approach to solving the RSM problem offers a novel and theoretically grounded perspective on LA problems in general, and specifically for the RSM problem. Figure C.3 gives a simplified overview of how the general approach of using DRL with the simulation environment works. This follows the principles of the DRL algorithms presented in the previous sections.

Figure D.3.: Simplified visualization of the training workflow of a DRL algorithm using the simulation environment. At the start, $M = \{\}$ such that it gets filled during the training. M can be an experience replay but also other data structures that are rather used in policy optimization methods.



Bibliography

Allahbakhsh, Mohammad et al. (Mar. 2019). "Crowdsourcing planar facility location allocation problems". en. In: *Computing* 101.3, pp. 237–261.

Aloysius, Neena and M. Geetha (2017). "A review on deep convolutional neural networks". In: *2017 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0588–0592. DOI: 10.1109/ICCSP.2017.8286426.

Alp, Osman, Erhan Erkut, and Zvi Drezner (2003). "An Efficient Genetic Algorithm for the p-Median Problem". In: *Ann. Oper. Res.* 122.1/4, pp. 21–42.

An, Hyung-Chan, Ashkan Norouzi-Fard, and Ola Svensson (Feb. 2017). "Dynamic Facility Location via Exponential Clocks". In: *ACM Trans. Algorithms* 13.2. ISSN: 1549-6325. DOI: 10.1145/2928272. URL: https://doi.org/10.1145/2928272.

Arulkumaran, Kai et al. (2017). "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6, pp. 26–38. DOI: 10.1109/MSP.2017.2743240.

Avella, Pasquale, Antonio Sassano, and Igor Vasil'ev (Jan. 2007). "Computational study of large-scale p-Median problems". en. In: *Math. Program.* 109.1, pp. 89–114.

Azarmand, Zeinab and Ensiyeh Neishabouri (2009a). "Location Allocation Problem". In: *Facility Location: Concepts, Models, Algorithms and Case Studies*. Ed. by Reza Zanjirani Farahani and Masoud Hekmatfar. Heidelberg: Physica-Verlag HD, pp. 93–109. ISBN: 978-3-7908-2151-2. DOI: 10.1007/978-3-7908-2151-2_5. URL: https://doi.org/10.1007/978-3-7908-2151-2_5.

— (2009b). "Location Allocation Problem". In: *Facility Location: Concepts, Models, Algorithms and Case Studies*. Ed. by Reza Zanjirani Farahani and Masoud Hekmatfar. Heidelberg: Physica-Verlag HD, pp. 93–109. ISBN: 978-3-7908-2151-2. DOI: 10.1007/978-3-7908-2151-2_5. URL: https://doi.org/10.1007/978-3-7908-2151-2_5.

Ball, Michael O. (2011). "Heuristics based on mathematical programming". In: *Surveys in Operations Research and Management Science* 16.1, pp. 21–38. ISSN: 1876-7354. DOI: https://doi.org/10.1016/j.sorms.2010.07.001. URL: https://www.sciencedirect.com/science/article/pii/S1876735410000036.

Barbato, Michele et al. (2023). "Node based compact formulations for the Hamiltonian p-median problem". In: *Networks* 82.4, pp. 336–370. DOI: https://

doi.org/10.1002/net.22163. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.22163. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.22163.

Beasley, J.E. (1993). "Lagrangean heuristics for location problems". In: *European Journal of Operational Research* 65.3, pp. 383–399. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(93)90118-7. URL: https://www.sciencedirect.com/science/article/pii/0377221793901187.

Blanco, Víctor (June 2019). "Ordered p-median problems with neighbourhoods". en. In: *Comput. Optim. Appl.* 73.2, pp. 603–645.

Bowerman, Robert L., Paul H. Calamai, and G. Brent Hall (1999). "The demand partitioning method for reducing aggregation errors in p-median problems". In: *Computers and Operations Research* 26.10, pp. 1097–1111. ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(99)00020-9. URL: https://www.sciencedirect.com/science/article/pii/S0305054899000209.

Brockman, Greg et al. (2016). *OpenAI Gym*. arXiv: 1606.01540 [cs.LG]. URL: https://arxiv.org/abs/1606.01540.

Burke, Laura I. and James P. Ignizio (1992). "Neural networks and operations research: An overview". In: *Computers and Operations Research* 19.3, pp. 179–189. ISSN: 0305-0548. DOI: https://doi.org/10.1016/0305-0548(92)90043-5. URL: https://www.sciencedirect.com/science/article/pii/0305054892900435.

Caterini, Anthony L. and Dong Eui Chang (2018). "Recurrent Neural Networks". In: *Deep Neural Networks in a Mathematical Framework*. Cham: Springer International Publishing, pp. 59–79. ISBN: 978-3-319-75304-1. DOI: 10.1007/978-3-319-75304-1_5. URL: https://doi.org/10.1007/978-3-319-75304-1_5.

Cavalcanti Costa, Joao Guilherme, Yi Mei, and Mengjie Zhang (2021). "An Evolutionary Hyper-Heuristic Approach to the Large Scale Vehicle Routing Problem". In: *2021 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2109–2116. DOI: 10.1109/CEC45853.2021.9504818.

Chen, Jun and Haopeng Chen (2021). *Edge-Featured Graph Attention Network*. arXiv: 2101.07671 [cs.LG]. URL: https://arxiv.org/abs/2101.07671.

Corso, Gabriele et al. (Mar. 2024). "Graph neural networks". en. In: *Nat. Rev. Methods Primers* 4.1.

Daskin, Mark S. and Kayse Lee Maass (2015). "The p-Median Problem". In: *Location Science*. Ed. by Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama. Cham: Springer International Publishing, pp. 21–45. ISBN: 978-3-319-13111-5. DOI: 10.1007/978-3-319-13111-5_2. URL: https://doi.org/10.1007/978-3-319-13111-5_2.

Dominguez, Enrique and Jose Munoz (2008). "A neural model for the p-median problem". In: *Computers and Operations Research* 35.2. Part Special Issue: Location Modeling Dedicated to the memory of Charles S. ReVelle, pp. 404–416. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2006.03.005. URL: https://www.sciencedirect.com/science/article/pii/S0305054806000943.

Dominguez Merino, Enrique and José Muñoz Perez (2002). "An Efficient Neural Network Algorithm for the p-Median Problem". In: *Advances in Artificial Intelligence — IBERAMIA 2002*. Ed. by Francisco J. Garijo, José C. Riquelme, and Miguel Toro. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 460–469. ISBN: 978-3-540-36131-2.

Eisenstat, David, Claire Mathieu, and Nicolas Schabanel (Mar. 2014). "Facility Location in Evolving Metrics". In: vol. 8573. ISBN: 978-3-662-43950-0. DOI: 10.1007/978-3-662-43951-7_39.

Farahani, Reza Zanjirani, Maryam Abedian, and Sara Sharahi (2009a). "Dynamic Facility Location Problem". In: *Facility Location: Concepts, Models, Algorithms and Case Studies*. Ed. by Reza Zanjirani Farahani and Masoud Hekmatfar. Heidelberg: Physica-Verlag HD, pp. 347–372. ISBN: 978-3-7908-2151-2. DOI: 10.1007/978-3-7908-2151-2_15. URL: https://doi.org/10.1007/978-3-7908-2151-2_15.

— (2009b). "Dynamic Facility Location Problem". In: *Facility Location: Concepts, Models, Algorithms and Case Studies*. Ed. by Reza Zanjirani Farahani and Masoud Hekmatfar. Heidelberg: Physica-Verlag HD, pp. 347–372. ISBN: 978-3-7908-2151-2. DOI: 10.1007/978-3-7908-2151-2_15. URL: https://doi.org/10.1007/978-3-7908-2151-2_15.

Fernandez, Elena and Mercedes Landete (Sept. 2015). "Fixed-Charge Facility Location Problems". In: *Location Science*. Ed. by Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama. Springer Books. Springer. Chap. 0, pp. 47–77. DOI: 10.1007/978-3-319-13111-5. URL: https://ideas.repec.org/h/spr/sprchp/978-3-319-13111-5_3.html.

GalvÃ£o, Roberto D. (Oct. 1980). "A Dual-Bounded Algorithm for the p -Median Problem". In: *Operations Research* 28.5, pp. 1112–1121. DOI: 10.1287/opre.28.5.1112. URL: https://ideas.repec.org/a/inm/oropre/v28y1980i5p1112-1121.html.

Guo, Wenxuan, Yanyan Xu, and Yaohui Jin (2023). *Swap-based Deep Reinforcement Learning for Facility Location Problems in Networks*. arXiv: 2312.15658 [cs.LG].

Hamilton, William L. (2020a). "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3, pp. 49–54.

Hamilton, William L. (2020b). "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3, pp. 62–64.

Hansen, P. and N. Mladenovic (1997). "Variable neighborhood search for the p-median". In: *Location Science* 5.4, pp. 207–226. ISSN: 0966-8349. DOI: https://doi.org/10.1016/S0966-8349(98)00030-8. URL: https://www.sciencedirect.com/science/article/pii/S0966834998000308.

Harary, F and G Gupta (Apr. 1997). "Dynamic graph models". en. In: *Math. Comput. Model.* 25.7, pp. 79–87.

Hasselt, Hado van, Arthur Guez, and David Silver (2015). *Deep Reinforcement Learning with Double Q-learning*. arXiv: 1509.06461 [cs.LG]. URL: https://arxiv.org/abs/1509.06461.

Hoi, Steven C.H. et al. (2021). "Online learning: A comprehensive survey". In: *Neurocomputing* 459, pp. 249–289. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.04.112. URL: https://www.sciencedirect.com/science/article/pii/S0925231221006706.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5, pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

Hribar, Michelle and Mark S. Daskin (Sept. 1997). "A dynamic programming heuristic for the P-median problem". In: *European Journal of Operational Research* 101.3, pp. 499–508. URL: https://ideas.repec.org/a/eee/ejores/v101y1997i3p499-508.html.

"Introduction to Location Theory and Models" (2013). In: *Network and Discrete Location: Models, Algorithms, and Applications, Second Edition*. John Wiley and Sons, Ltd. Chap. 1, pp. 1–28. ISBN: 9781118537015. DOI: https://doi.org/10.1002/9781118537015.ch01. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118537015.ch01. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118537015.ch01.

Kariv, O. and S. L. Hakimi (1979). "An Algorithmic Approach to Network Location Problems. II: The p-Medians". In: *SIAM Journal on Applied Mathematics* 37.3, pp. 539–560. ISSN: 00361399. URL: http://www.jstor.org/stable/2100911 (visited on 05/15/2024).

Kazakovtsev, Lev (Oct. 2013). "Random Search Algorithm for the p-Median Problem". In: *Informatica* 37, pp. 267–278.

Keriven, Nicolas (2024). "Not too little, not too much: a theoretical analysis of graph (over)smoothing". In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS '22. New Orleans, LA, USA: Curran Associates Inc. ISBN: 9781713871088.

Khemani, Bharti et al. (Jan. 2024). "A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions". In: *Journal of Big Data* 11.1, p. 18. ISSN: 2196-1115. DOI: 10.1186/s40537-023-00876-4. URL: https://doi.org/10.1186/s40537-023-00876-4.

Kipf, Thomas N. and Max Welling (2017). *Semi-Supervised Classification with Graph Convolutional Networks*. arXiv: 1609.02907 [cs.LG]. URL: https://arxiv.org/abs/1609.02907.

Klar, Matthias, Moritz Glatt, and Jan C. Aurich (2021). "An implementation of a reinforcement learning based algorithm for factory layout planning". In: *Manufacturing Letters* 30, pp. 1–4. ISSN: 2213-8463. DOI: https://doi.org/10.1016/j.mfglet.2021.08.003. URL: https://www.sciencedirect.com/science/article/pii/S2213846321000651.

Klose, Andreas and Andreas Drexl (2005). "Facility location models for distribution system design". In: *European Journal of Operational Research* 162.1. Logistics: From Theory to Application, pp. 4–29. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2003.10.031. URL: https://www.sciencedirect.com/science/article/pii/S0377221703008191.

Köstler, Johannes et al. (2023). "Fluidity: Location-Awareness in Replicated State Machines". In: *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. SAC '23. Tallinn, Estonia: Association for Computing Machinery, 192â201. ISBN: 9781450395175. DOI: 10.1145/3555776.3577763. URL: https://doi.org/10.1145/3555776.3577763.

Lachhwani, Kailash (Jan. 2020). "Application of neural network models for mathematical programming problems: A state of art review". en. In: *Arch. Comput. Methods Eng.* 27.1, pp. 171–182.

Laporte, Gilbert, Stefan Nickel, and Francisco Saldanha da Gama (2015). "Introduction to Location Science". In: *Location Science*. Ed. by Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama. Cham: Springer International Publishing, pp. 1–18. ISBN: 978-3-319-13111-5. URL: https://doi.org/10.1007/978-3-319-13111-5_1.

Lawniczak, Laura and Tobias Distler (2021). "Stream-based State-Machine Replication". In: *2021 17th European Dependable Computing Conference (EDCC)*, pp. 119–126. DOI: 10.1109/EDCC53658.2021.00024.

Li, Yujia et al. (2017). *Gated Graph Sequence Neural Networks*. arXiv: 1511.05493 [cs.LG]. URL: https://arxiv.org/abs/1511.05493.

Liu, Shiqing, Xueming Yan, and Yaochu Jin (2023). "End-to-End Pareto Set Prediction withÂ Graph Neural Networks forÂ Multi-objective Facility Location". In: *Evolutionary Multi-Criterion Optimization*. Ed. by Michael Emmerich et al. Cham: Springer Nature Switzerland, pp. 147–161. ISBN: 978-3-031-27250-9.

Maranzana, F. E. (1964). "On the Location of Supply Points to Minimize Transport Costs". In: *OR* 15.3, pp. 261–270. ISSN: 14732858. URL: http://www.jstor.org/stable/3007214 (visited on 05/15/2024).

Matis, David and Peter Tarabek (2023). "Reinforcement Learning for Weighted p-median Problem". In: *2023 International Conference on Information and Digital Technologies (IDT)*, pp. 293–298. DOI: 10.1109/IDT59031.2023.10194404.

Michiels, W., E. H. L. Aarts, and J. Korst (2018). "Theory of Local Search". In: *Handbook of Heuristics*. Ed. by Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende. Cham: Springer International Publishing, pp. 299–339. ISBN: 978-3-319-07124-4. DOI: 10.1007/978-3-319-07124-4_6. URL: https://doi.org/10.1007/978-3-319-07124-4_6.

Mladenovic, Nenad et al. (2007). "The p-median problem: A survey of metaheuristic approaches". In: *European Journal of Operational Research* 179.3, pp. 927–939. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2005.05.034. URL: https://www.sciencedirect.com/science/article/pii/S0377221706000750.

Mnih, Volodymyr, AdriÃ PuigdomÃ¨nech Badia, et al. (2016). *Asynchronous Methods for Deep Reinforcement Learning*. arXiv: 1602.01783 [cs.LG]. URL: https://arxiv.org/abs/1602.01783.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, et al. (2013). *Playing Atari with Deep Reinforcement Learning*. arXiv: 1312.5602 [cs.LG]. URL: https://arxiv.org/abs/1312.5602.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, et al. (Feb. 2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, 529â533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: http://dx.doi.org/10.1038/nature14236.

Owen, Susan Hesse and Mark S. Daskin (1998). "Strategic facility location: A review". In: *European Journal of Operational Research* 111.3, pp. 423–447. ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(98)00186-6. URL: https://www.sciencedirect.com/science/article/pii/S0377221798001866.

Plastria, Frank (2001). "Static competitive facility location: An overview of optimisation approaches". In: *European Journal of Operational Research* 129.3, pp. 461–470. ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(00)00169-7. URL: https://www.sciencedirect.com/science/article/pii/S0377221700001697.

Reiser, Patrick et al. (Nov. 2022). "Graph neural networks for materials science and chemistry". In: *Communications Materials* 3.1. ISSN: 2662-4443. DOI: 10.1038/s43246-022-00315-6. URL: http://dx.doi.org/10.1038/s43246-022-00315-6.

ReVelle, C.S., H.A. Eiselt, and M.S. Daskin (2008). "A bibliography for some fundamental problem categories in discrete location science". In: *European Journal of Operational Research* 184.3, pp. 817–848. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2006.12.044. URL: https://www.sciencedirect.com/science/article/pii/S037722170700080X.

ReVelle, Charles S. and Ralph W. Swain (1970). "Central Facilities Location". In: *Geographical Analysis* 2.1, pp. 30–42. DOI: https://doi.org/10.1111/j.1538-4632.1970.tb00142.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1538-4632.1970.tb00142.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.1970.tb00142.x.

Rohlfs, Chris (2023). "A descriptive analysis of olfactory sensation and memory in Drosophila and its relation to artificial neural networks". In: *Neurocomputing* 518, p. 26. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2022.10.068. URL: https://www.sciencedirect.com/science/article/pii/S0925231222013492.

Rosenthal, Richard E., John A. White, and Donovan Young (1978). "Stochastic Dynamic Location Analysis". In: *Management Science* 24.6, pp. 645–653. ISSN: 00251909, 15265501. URL: http://www.jstor.org/stable/2630839 (visited on 07/13/2024).

Rosing, K.E. et al. (1998). "Heuristic concentration and Tabu search: A head to head comparison". In: *European Journal of Operational Research* 104.1, pp. 93–99. ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(97)00310-X. URL: https://www.sciencedirect.com/science/article/pii/S037722179700310X.

Salhi, S. (1997). "A Perturbation Heuristic for a Class of Location Problems". In: *The Journal of the Operational Research Society* 48.12, pp. 1233–1240. ISSN: 01605682, 14769360. URL: http://www.jstor.org/stable/3010753 (visited on 05/15/2024).

— (2002). "Defining tabu list size and aspiration criterion within tabu search methods". In: *Computers and Operations Research* 29.1, pp. 67–86. ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(00)00062-9. URL: https://www.sciencedirect.com/science/article/pii/S0305054800000629.

Salhi, S. and R.A. Atkinson (1995). "Subdrop: A modified drop heuristic for location problems". In: *Location Science* 3.4, pp. 267–273. ISSN: 0966-8349. DOI: https://doi.org/10.1016/0966-8349(96)00003-4. URL: https://www.sciencedirect.com/science/article/pii/0966834996000034.

Scarselli, Franco et al. (2009). "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

Schlichtkrull, Michael et al. (2017). *Modeling Relational Data with Graph Convolutional Networks*. arXiv: 1703.06103 [stat.ML]. URL: https://arxiv.org/abs/1703.06103.

Schneckenreither, Manuel and Stefan Haeussler (2019). "Reinforcement Learning Methods for Operations Research Applications: The Order Release Problem". In: *Machine Learning, Optimization, and Data Science*. Ed. by Giuseppe Nicosia et al. Cham: Springer International Publishing, pp. 545–559. ISBN: 978-3-030-13709-0.

Schneider, Fred B. (Dec. 1990). "Implementing fault-tolerant services using the state machine approach: a tutorial". In: *ACM Comput. Surv.* 22.4, 299â319. ISSN: 0360-0300. DOI: 10.1145/98163.98167. URL: https://doi.org/10.1145/98163.98167.

Schulman, John, Sergey Levine, et al. (2017). *Trust Region Policy Optimization*. arXiv: 1502.05477 [cs.LG]. URL: https://arxiv.org/abs/1502.05477.

Schulman, John, Filip Wolski, et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

Shakya, Ashish Kumar, Gopinatha Pillai, and Sohom Chakrabarty (2023a). "Reinforcement learning algorithms: A brief survey". In: *Expert Systems with Applications* 231, p. 120495. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.120495. URL: https://www.sciencedirect.com/science/article/pii/S0957417423009971.

— (2023b). "Reinforcement learning algorithms: A brief survey". In: *Expert Systems with Applications* 231, p. 120495. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.120495. URL: https://www.sciencedirect.com/science/article/pii/S0957417423009971.

Shehab, Mohammad, Ahamad Tajudin Khader, and Mohammad Alia (Apr. 2019). "Enhancing Cuckoo Search Algorithm by using Reinforcement Learning for Constrained Engineering optimization Problems". In: pp. 812–816. DOI: 10.1109/JEEIT.2019.8717366.

Simpson, Mike, Andrea Genovese, and Ahmad Rais Mohamad Mokhtar (Oct. 2022). "Gaining access for data collection". In: *Handbook of Research Methods for Supply Chain Management*. Edward Elgar Publishing, pp. 86–104.

Sonabend-W, Aaron et al. (2020). *Expert-Supervised Reinforcement Learning for Offline Policy Learning and Evaluation*. arXiv: 2006.13189 [cs.LG]. URL: https://arxiv.org/abs/2006.13189.

Sun, Fang (2022). *Over-smoothing Effect of Graph Convolutional Networks*. arXiv: 2201.12830 [cs.LG]. URL: https://arxiv.org/abs/2201.12830.

Sutton, Richard S. and Andrew G. Barto (2018a). *Reinforcement Learning: An Introduction*. Second. The MIT Press, pp. 321–335. URL: http://incompleteideas.net/book/the-book-2nd.html.

— (2018b). *Reinforcement Learning: An Introduction*. Second. The MIT Press, pp. 58–60. URL: http://incompleteideas.net/book/the-book-2nd.html.

— (2018c). *Reinforcement Learning: An Introduction*. Second. The MIT Press, pp. 73–78. URL: http://incompleteideas.net/book/the-book-2nd.html.

— (2018d). *Reinforcement Learning: An Introduction*. Second. The MIT Press, pp. 321–337. URL: http://incompleteideas.net/book/the-book-2nd.html.

Tellier, Luc-Normand (1972). "The Weber Problem: Solution and Interpretation*". In: *Geographical Analysis* 4.3, pp. 215–233. DOI: https://doi.org/10.1111/j.1538-4632.1972.tb00472.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1538-4632.1972.tb00472.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.1972.tb00472.x.

Turkoglu, Derya Celik and Mujde Erol Genevois (Sept. 2020). "A comparative survey of service facility location problems". In: *Annals of Operations Research* 292.1, pp. 399–468. DOI: 10.1007/s10479-019-03385-. URL: https://ideas.repec.org/a/spr/annopr/v292y2020i1d10.1007_s10479-019-03385-x.html.

Vaswani, Ashish et al. (2023). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

Veličković, Petar et al. (2018). "Graph Attention Networks". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rJXMpikCZ.

Verdone, Alessio, Simone Scardapane, and Massimo Panella (2024). "Explainable Spatio-Temporal Graph Neural Networks for multi-site photovoltaic energy production". In: *Applied Energy* 353, p. 122151. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2023.122151. URL: https://www.sciencedirect.com/science/article/pii/S0306261923015155.

Voß, Stefan (2008). "Metaheuristics". In: *Encyclopedia of Optimization*. Boston, MA: Springer US, pp. 2061–2075.

Wan, Ching Pui, Tung Li, and Jason Min Wang (2023). *RLOR: A Flexible Framework of Deep Reinforcement Learning for Operation Research*. arXiv: 2303.13117 [math.OC].

Wang, Chenguang et al. (Jan. 2023). "Solving uncapacitated P-Median problem with reinforcement learning assisted by graph attention networks". en. In: *Appl. Intell.* 53.2, pp. 2010–2025.

Wang, Qi and Chunlei Tang (2021). "Deep reinforcement learning for transportation network combinatorial optimization: A survey". In: *Knowledge-Based Systems* 233, p. 107526. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.107526. URL: https://www.sciencedirect.com/science/article/pii/S0950705121007887.

Wang, Ziyu et al. (2016). *Dueling Network Architectures for Deep Reinforcement Learning*. arXiv: 1511.06581 [cs.LG]. URL: https://arxiv.org/abs/1511.06581.

Weber, Alfred (1922). *Ueber den standort der industrien*. Vol. 2. JCB Mohr (Paul Siebeck).

Whitaker, R.A. (1983). "A Fast Algorithm For The Greedy Interchange For Large-Scale Clustering And Median Location Problems". In: *INFOR: Information Systems and Operational Research* 21.2, pp. 95–108. DOI: 10.1080/03155986.1983.11731889. eprint: https://doi.org/10.1080/03155986.1983.11731889. URL: https://doi.org/10.1080/03155986.1983.11731889.

Xiao, Shunxin et al. (Nov. 2021). "Graph neural networks in node classification: survey and evaluation". In: *Machine Vision and Applications* 33.1. ISSN: 1432-1769. DOI: 10.1007/s00138-021-01251-0. URL: http://dx.doi.org/10.1007/s00138-021-01251-0.

Yan, Yimo et al. (2022). "Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities". In: *Transportation Research Part E: Logistics and Transportation Review* 162, p. 102712. ISSN: 1366-5545. DOI: https://doi.org/10.1016/j.tre.2022.102712. URL: https://www.sciencedirect.com/science/article/pii/S136655452200103X.

Yu, Lina et al. (2021). "Reinforcement learning approach for resource allocation in humanitarian logistics". In: *Expert Systems with Applications* 173, p. 114663. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.114663. URL: https://www.sciencedirect.com/science/article/pii/S0957417421001044.

Zare, Maryam et al. (2023). *A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges*. arXiv: 2309.02473 [cs.LG]. URL: https://arxiv.org/abs/2309.02473.