Informatics

A Learning Multilevel Optimization Approach for a Large Location Allocation Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Laurenz Tomandl, B.Sc.
Matrikelnummer 01326545

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Dipl.-Ing. Dr.techn. Thomas Jatschka

Wien, 14. Mai 2023

Laurenz Tomandl Günther Raidl

Informatics

A Learning Multilevel Optimization Approach for a Large Location Allocation Problem

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Laurenz Tomandl, B.Sc.
Registration Number 01326545

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Dipl.-Ing. Dr.techn. Thomas Jatschka

Vienna, 14th May, 2023

Laurenz Tomandl   Günther Raidl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Laurenz Tomandl, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 14. Mai 2023

Laurenz Tomandl

Acknowledgements

Ich möchte mich ganz besonders bei meinen Betreuern Günther Raidl und Thomas Jatschka bedanken, die mich in unzähligen Meetings unter fachkundiger Anleitung bei dieser Arbeit unterstützt haben.

Weiters möchte ich mich bei meiner Eltern Barbara und Erich bedanken, die mich nicht nur finanziell, sondern auch emotional während meines Studiums unterstützten.

Außerdem bedanke ich mich noch bei meiner Freundin Miriam, die alle Höhen und Tiefen, die ich beim Entwickeln und Schreiben der Arbeit durchlebte, miterleben musste und immer ein offenes Ohr für meine Probleme hatte.

Kurzfassung

Ein Problem für die breite Akzeptanz von Elektrofahrzeugen sind die langen Ladezeiten der Batterien. Um den Kunden die Wartezeit zu ersparen, sind Fahrzeuge mit austauschbaren Batterien und ein Netz von Batterietauschstationen eine vielversprechende Lösung für kleinere Fahrzeuge wie Elektroroller. Der Kunde kann zu einer Station fahren und seine leere Batterie gegen eine bereits geladene austauschen und so die Wartezeit vermeiden. Um die Einführung eines solchen Systems zu ermöglichen, ist eine gute Infrastruktur erforderlich.

Wir betrachten daher das Demand Maximizing Battery Swapping Station Location Problem (DMBSSLP). Das DMBSSLP modelliert die Herausforderung, optimale Standorte und Konfigurationen für Batterietauschstationen in einer städtischen Umgebung zu finden.

In dieser Arbeit entwickeln wir eine neuartige, durch maschinelles Lernen unterstützte Metaheuristik, die darauf abzielt, dieses Problem für eine große Anzahl von potenziellen Gebieten und Kunden zu lösen. Wir nennen diesen Algorithmus den Learning Multi Level Optimization (LMLO) Ansatz. Dieser verwendet Elemente der konventionellen Multi Level Optimization (MLO) Metaheuristik. Vergröberung einer Probleminstanz, bis sie klein ist, genaue Lösung der kleinen Instanz und Projektion der Lösung der kleinen Probleminstanz zurück auf die ursprüngliche Probleminstanz. Ein wichtiger Teil dieses Ansatzes ist zu bestimmen, wie die Elemente der Probleminstanz vergröbert werden sollen. Um dieses Problem zu lösen, benutzen wir zwei neuronale Netzwerke, die den Vergröberungsprozess steuern.

Der Algorithmus wird an Probleminstanzen mit bis zu 10.000 Gebieten und Kunden getestet und mit konventionellen MLO-Ansätzen verglichen, die das Jaccard-Ähnlichkeitsmaß [Jac02] verwenden. Unser LMLO Ansatz ist in der Lage, die Anzahl der Kunden, die erfolgreich ihre Batterien tauschen können, auf Instanzen dieser Größenordnung signifikant zu verbessern.

Abstract

A problem in the wide-scale adoption of electric vehicles is the long charging time. To avoid the waiting time for the customer, vehicles with exchangeable batteries and a network of battery swapping stations are a promising solution for smaller-scale vehicles like electric scooters. A customer can drive to a station and exchange their depleted batteries with an already charged battery and thus avoid the waiting that would be necessary otherwise. A good infrastructure is needed to make the implementation of such a system viable.

We, therefore, consider the Demand Maximizing Battery Swapping Station Location Problem (DMBSSLP). The DMBSSLP models the challenge of finding optimal locations and configurations for battery swapping stations in an urban environment.

In this thesis, we develop a novel machine learning supported metaheuristic which aims to solve this problem for a large number of potential areas and customers. We call this algorithm the Learning Multilevel Optimization (LMLO) approach. It uses elements of the conventional Multilevel Optimization (MLO) metaheuristic. Coarsening a problem instance until it is small, solving the small instance accurately, and projecting the solution of the small problem instance back to the original problem instance. A challenging part of this algorithm is to determine how elements of the problem instance should be coarsened. To solve this problem we consider a machine learning approach using two Neural Network (NN) models to guide the coarsening process.

The algorithm is tested on problem instances with up to 10.000 areas and customers and is compared to conventional MLO approaches which use the Jaccard similarity measures [Jac02]. Our LMLO approach can to significantly improve the number of satisfied customers on instances of sizes up to 10000 areas and customers.

Contents

CHAPTER 1
Introduction

This chapter gives an overview of the motivation, aim, and structure of this work. First of all the application scenarios and considered optimization task of the Demand Maximizing Battery Swapping Station Location Problem (DMBSSLP) is described, secondly, the algorithmic idea is sketched, lastly, the chapter concludes with an overview of the structure of the thesis.

1.1 Motivation

The recharging of batteries for electric vehicles is usually a time-consuming process. To reduce the time needed for the customer, a solution is to build vehicles with easily exchangeable batteries. Battery swapping for large vehicles, like cars, is not easily applicable due to the size and missing standardization of the batteries but, it is a promising alternative for smaller vehicles like electric scooters. Customers can swap their depleted battery for an already charged one and thus do not have to wait for their battery to charge. Charged batteries are provided at a battery swapping station in exchange for old depleted batteries. Such stations are relatively simple and can be easily attached to supermarkets or conventional gas stations. The batteries are then recharged while the user continues the journey.

An important task in designing such systems is to determine where battery swapping stations should be set up and how many battery charging slots shall be provided at each station. The construction and maintenance of such stations cost money thus such projects often only have a set budget. The goal for our intents and purposes is to maximize the fulfilled demand among all customers while keeping the cost within the budget.

To investigate this problem we examine the DMBSSLP which is an adapted version of the Multiperiod Battery Swapping Station Location Problem (MBSSLP) by Jatschka et al. [JRR23].

1.2 Aim of the work

The DMBSSLP is an optimization problem that can be formulated in terms of a Mixed Integer Linear Programming (MILP) approach. As such it can be very difficult to find optimal or even good solutions when the problem instances get very large. The work aims to develop and implement a novel machine learning supported metaheuristic for the DMBSSLP. More specifically, our approach is based on the Multilevel Optimization (MLO) approach (also called Multilevel Refinement (MLR)) by Walshaw [Wal02] and is called Learning Multilevel Optimization (LMLO). MLO approaches scale very well in terms of running time and solution quality for large instances and are therefore suited to tackle the scalability problem.

The goal is to apply the LMLO to large instances of the DMBSSLP that resemble urban environments. For large cities with a population of millions and nearly every intersection being a possible location for a charging station, it is reasonable to assume that tens of thousands of potential areas and tens of thousands of potential customers are to be considered. To be able to solve such large instances of the problem we have to resort to metaheuristics, which allow us to find solutions to large problems in reasonable time.

In the LMLO approach, the original problem instance is repeatedly coarsened to a smaller instance until the problem is small enough to be quickly and accurately solved. The found solution is then iteratively projected back through the coarsened levels and refined until a solution for the original problem instance is found.

The difference between traditional MLO and our novel LMLO approach lies in the coarsening process. Conventional methods use handcrafted similarity measures to decide how to coarsen a problem instance. In the LMLO approach the decision of how to coarsen an instance is guided by two Neural Networks that aim to predict which merging steps lead to the smallest error in the final outcome.

The work aims to show that the LMLO approach can improve conventional MLO approaches in terms of solution quality and still perform well in terms of scalability. This is shown by applying MLO and LMLO to instances of the DMBSSLP.

1.3 Structure of the work

Chapter 2 starts by defining the DMBSSLP as a problem on a graph. This representation on a graph is then transformed into a representation in terms of a Mixed Integer Linear Program (MILP).

Chapter 3 concerns the general methodology we apply in this thesis. We give an exact and detailed explanation of the MLO algorithm for different graph structures and discuss MILP techniques and solvers.

Chapter 4 discusses work related to the DMBSSLP. In the beginning, we present previous work that has been done as a foundation for this work. We continue with discussion of location problems representing generalized versions of the problem which in turn leads problems that are very similar to the DMBSSLP.

In addition, we also discuss the MLO algorithm and what problems it already has been applied to.

Chapter 5 describes the development and implementation process of the LMLO. This includes similarity calculation, instance, and data generation processes.

In **Chapter 6** we investigate the experimental results. We start by describing the used computing environment, continue with a data analysis of the training data, and a description of the benchmark instances we used is given. Finally, comparisons of the models are made to other model structures and to results obtained by conventional similarity measures.

In the final **Chapter 7** we present a summary of the findings and also describe possible future work.

CHAPTER 2

The Demand Maximizing Battery Swapping Station Location Problem

In this section, we formalize the DMBSSLP. First, we explain how the problem can be understood on a graph. Secondly, we transform this graph model into the MILP formulation.

2.1 Graph Model

The DMBSSLP is modeled on a graph $P(V, E)$ with its vertex set V and edge set E. There are $|V| = n$ different nodes. Each node in the graph represents a possible area for battery swapping stations. At each station battery charging slots can be built. The expected customer demand is described in terms of origin-destination-pairs (OD-pairs). These pairs consist of a starting node, an end node, and the number of expected customers, called the demand, on this trip. Each edge has a distance property describing the distance between the nodes it connects. The graph is connected which means for every node pair there exists a path within the graph connecting the pair.

An example of a potential graph $P(V, E)$ can be seen in Figure 2.1. This example graph consists of $n = 10$ possible areas and $m = 10$ OD-pairs.

The graph formulation $P(V, E)$ is very close to the real-world problem but does not allow to grasp certain characteristics of the problem, e.g. which area can satisfy the demand of an OD-pair. We, therefore, reformulate the problem and model it on a bipartite graph $G(Q, L, E)$. This graph consists of two distinct node sets. There are no edges between nodes that belong to the same set, only edges between nodes of different sets. In our problem, the Q set represents all OD-pairs of the instance and the L set represents all

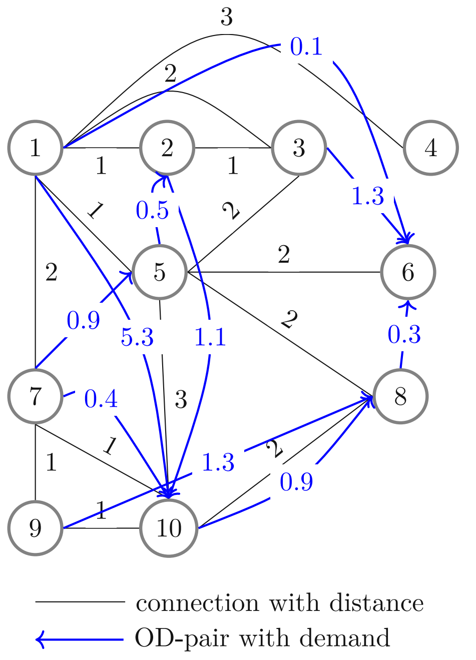


Figure 2.1: Graph model (own figure)

possible areas where stations can be built. A station built in area l can meet the demand of an OD-pair if the detour from the shortest path from the origin to the destination does not exceed a certain allowed detour range. A node q of the Q set has a connection to a node l of the L set if the area l is within the detour range and thus a station built at this area could potentially satisfy the demand of q. This representation allows us to quickly check if stations in an area can satisfy the demand of an OD-pair, as all eligible OD-pairs are connected to the area via an edge.

An example of such a graph can be seen in Figure 2.2. This represents the corresponding bipartite graph to the previously introduced graph in Figure 2.1 with a maximum detour length of 0. We can see that all OD-pairs just consider areas that are along their shortest path. For example the shortest path for OD-pair $1-6$ travels through area 1, 5 and 6, therefore the Q node $1-6$ is connected to the L nodes 1, 5 and 6. It might also happen that an area is not considered by any OD-pair, as in this example area 4. This area will later be trimmed when searching for solutions.

2.2 MILP Formulation

The MILP formulation for the DMBSSLP is based on the MILP formulation of the MBSSLP by Jatschka et al. [JRR23] where additional time and customer loss constraints are considered.

Until now we described the problem on a high level without defining any properties of

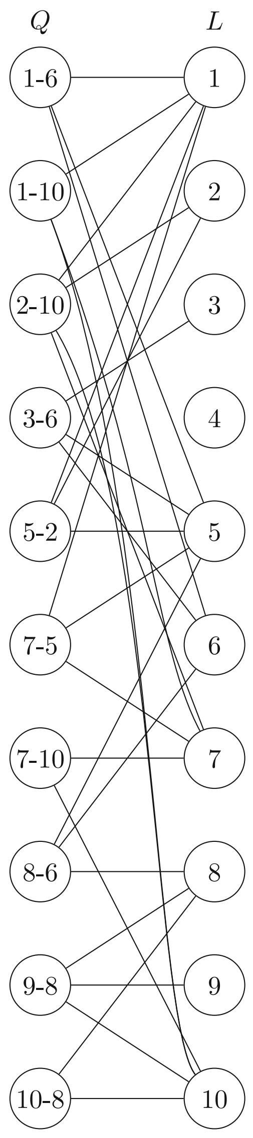


Figure 2.2: Bipartite graph (own figure)

the DMBSSLP outside of the set of potential areas L and the set of OD-pairs Q. To describe the problem in terms of MILP we need to define some variables.

The variables defining our solution are:

- x_l: The x_l variable represents the number of station being built at area l. At each station, a certain number of battery charging slots can be built
- y_l: The y_l variable represents the number of slots that are built at area l. Each slot represents one battery that can be charged there.

- a_{ql}: This represents the actual demand that is assigned from OD-pair q to an area l.

The following properties are constant for a single instance.

- s: The number of slots that can maximally be built at a station.
- c: The cost of a station.
- b: The cost of a slot.
- B: The project budget for building stations and slots.

Properties that are variable for each area or OD-pair are denoted with a subscript representing the node, e.g. the number of stations that are allowed to be built at an area l is r_l.

The relevant properties for a node l of the area set L are:

- r_l: The number of stations that can be built in an area.
- $N(l)$: The set of OD-pairs that can fulfill their demand at a station built in area l.
- $\overline{d_l}$: The maximum demand that can be fulfilled by an area.

$$\overline{d_l} = \min(sr_l, \sum_{q \in N(l)} d_q) \tag{2.1}$$

The relevant properties for a node q of the OD-pair set Q are:

- d_q: The demand a OD-pair has.
- $N(q)$: The set of locations that can potentially fulfill the demand of OD-pair q.

The edges connecting nodes of the L and Q set also have properties. These are described with a subscript of firstly the q node and secondly the l node.

- e_{ql}: The maximal demand that can be assigned from an OD-pair q to an area l.

$$e_{ql} = \min(\overline{d_l}, d_q) \tag{2.2}$$

With these definitions, we can create a MILP formulation for the problem.

$$\max \sum_{q\in Q} \sum_{l\in N(q)} a_{ql} \tag{2.3}$$

$$sx_l \geq y_l \qquad l \in L \tag{2.4}$$

$$\sum_{l\in N(q)} a_{ql} \leq d_q \qquad q \in Q \tag{2.5}$$

$$\sum_{q\in N(l)} a_{ql} \leq y_l \qquad l \in L \tag{2.6}$$

$$\sum_{l\in L} (c_l x_l + b_l y_l) \leq B \tag{2.7}$$

$$x_l \in \{0, \ldots, r_l\} \qquad l \in L \tag{2.8}$$

$$y_l \in \{0, \ldots, \lceil \bar{d}_l \rceil\} \qquad l \in L \tag{2.9}$$

$$0 \leq a_{ql} \leq e_{ql} \qquad q \in Q,\ l \in N(q) \tag{2.10}$$

The goal of the objective (2.3) is to find the solutions that maximize the fulfilled demand. With the inequality (2.4) we link the x_l and y_l variables and ensure that slots are only opened if a sufficient number of stations are opened in the area. Constraint (2.5) enforces that the total amount of demand assigned from an OD-pair to an area does not exceed the given demand of the OD-pair. Inequality (2.6) ensures that the total amount of demand assigned to an area is smaller than the number of slots at the location. With the constraint (2.7) we assert that the total cost of all stations and slots stays within the bounds of the available budget. Lastly, the domain of the x_l, y_l, and a_{ql} variables are given in (2.8)-(2.10), where x_l and y_l are defined as integer variables and a_{ql} is defined as a continuous variable.

CHAPTER 3

Methodology

In this chapter, we are going to explain the methodological approaches used in this thesis on a high level. Firstly we will explain the Multilevel Optimization algorithm, secondly, we give a short overview of MILP.

3.1 Multilevel Optimization

Algorithm 3.1: high level MLO based on [Wal02] and [VFF$^+$20]

```
   Input : a problem instance $G_0$
   Output: a solution $S_0(G_0)$
 1: for $i = 1 \dots N$ do
 2:     $M_i \leftarrow$ partitioning($G_{i-1}$);
 3:     $G_i \leftarrow$ coarsen($G_{i-1}$, $M_i$);
 4: end for
 5: $S_N(G_N) \leftarrow$ solve problem $G_N$;
 6: for $i = N - 1 \dots 0$ do
 7:     $S_i^0 \leftarrow$ project($S_{i+1}(G_{i+1}), G_i$);
 8:     $S_i \leftarrow$ refine($S_i^0(G_i)$);
 9: end for
10: return $S_0(G_0)$;
```

The Multilevel Optimization (MLO), also called Multilevel Refinement (MLR), approach was first introduced in the 90s by Barnard et al. [BS94] and later formalized by Hendrickson et al. [HL$^+$95] and Walshaw [Wal02]. The MLO approach is often well suited to address very large instances of a problem.

A high-level illustration of the MLO can be seen in Figure 3.1 taken from Valejo et al.[VFF$^+$20].

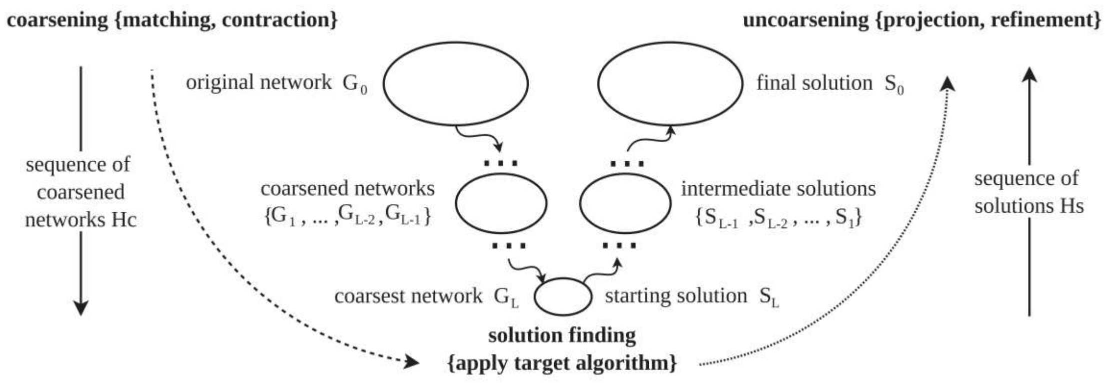

coarsening {matching, contraction}
uncoarsening {projection, refinement}
original network G0
final solution S0
sequence of coarsened networks Hc
coarsened networks {G1 , ... ,GL-2 ,GL-1 }
intermediate solutions {SL-1 ,SL-2 , ... , S1}
sequence of solutions Hs
coarsest network GL
starting solution SL
solution finding {apply target algorithm}

Figure 3.1: Phases of the MLO process [VFF+20]

The algorithm consists of three major steps. The first step is partitioning and coarsening in which the original problem is iteratively simplified. Firstly a partition is created. The partition decides which parts of the problem will be merged to create a smaller problem instance. The next step is coarsening. One coarsening step is usually done by combining partitioned items of a problem instance. e.g. nodes for a problem on a graph to create a smaller coarser instance. This process is repeated N times, where N is chosen such that the final instance is small enough to be solved efficiently.

In the second step, a solution is created for the smallest instance. This can be done in a variety of ways. Often MILP solvers or well-working problem specific heuristics are applied to create an initial solution.

In the third and final step, the found solution is repeatedly projected back and refined to the less coarse instances until a solution for the original problem instance is found. A high-level version of this Algorithm can be seen in 3.1 as described in [Wal02] and [VFF+20].

3.1.1 Partitioning and Coarsening

During the coarsening process, vertices are merged according to the partitioning. This leads to an inevitable loss of information about the structure and behavior of the problem. It is therefore crucial that during the partitioning process, suitable merging partners are selected to keep this loss of information minimal. Over the years, several partitioning algorithms have been developed, and a select few are presented here:

- **Heave Edge Matching (HEM):** Iteratively a random item of the problem instance is chosen, if it is still unmatched, it is matched with the neighbor it is most similar to [KAKS97]. This method runs in $O(m)$ where m is the number of neighboring pairs.

- **Greedy Heave Edge Matching (GHEM):** For every item, the similarity between its neighbors is calculated. The item pairs are sorted according to their similarity.

The most similar item pair is chosen and if both items are still unmatched they are matched together. This is repeated until a large enough matching is created. This method runs in $O(m \log n)$ [VFF$^+$20] with the number of neighboring pairs being m and the number of total items being n.

- **Label Propagation:** This method is inspired by the community detection algorithm by Raghavan et al [RAK07] and was adapted by Meyerhenke et al. to a cluster selection algorithm for graphs with a runtime of $O(n + m)$ [MSS14]. This method works over multiple time steps. Initially, a unique label is assigned to each item. In the next time step, each node label is updated with the most frequent label among its neighbors. The algorithm terminates when an additional time step would not change any label. The idea is that closely connected items will converge to the same label. The items with the same label can then be partitioned together and will be collapsed into a single item in the coarsening step [VFF$^+$20]. Notably different from HEM and GHEM is that with this method multiple items are grouped together at the same time.

During the coarsening, the found partitioning is applied to the problem instance. Items that are partitioned together are collapsed into a single item that derives its information from both original items. Not partitioned items will be inherited as is. The resulting problem instance is smaller than the original.

3.1.2 Solving

Once the problem instance is small enough a problem solution is searched. A variety of strategies are applicable often MILP solvers (see Section 3.3) or meaningful problem specific heuristics can be applied.

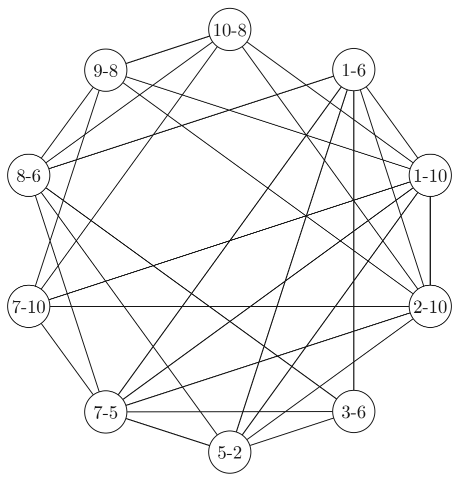
3.1.3 Projection and Refinement

During the projection phase also often called extension [VFF$^+$20] the found solution is projected to the next finer instance. This projection is problem specific and depends on the chosen problem representation. In general, the contracted items that have been coarsened are now again split apart.

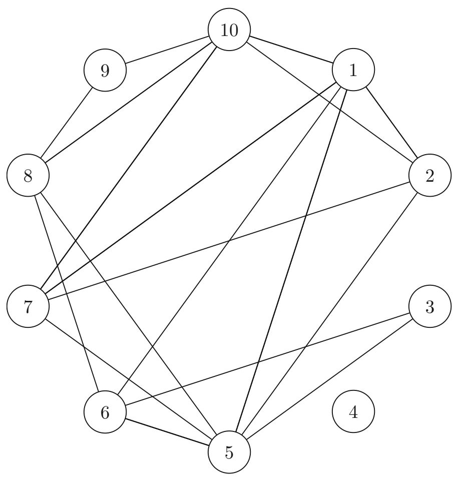
In the refinement step, the projected solution is possibly improved. This means the solution that has just been projected are reevaluated using fast problem specific heuristics and adapted if needed.

3.2 MLO for Bipartite Networks

So far we explained the MLO in a generalized way so that it can be applied to any problem. In Section 2.1 we defined our problem on a bipartite graph $G(Q, L, E)$ with two disjoint node sets Q and L. We therefore specify the definition of MLO according to Valejo et al. [VdOGFdAL18] to bipartite networks.



(a) two-hop graph of Q (own figure)



(b) two-hop graph of L (own figure)

Figure 3.2: two-hop graphs for graph Figure 2.1

The core concepts of the algorithm remain: The phases are partitioning, coarsening, solving, projecting, and refining. The only part that changes is the partitioning. We can no longer contract neighboring vertices because they must lie in different node sets and thus carry different meanings in the problem. Therefore we define the concept of the two-hop neighborhood for bipartite graphs according to Valejo et al. [VdOdSNZ21]. In the two-hop neighborhood, nodes are considered neighboring if they are connected via exactly two edges. For a bipartite graph $G(Q, L, E)$ with disjoint node sets Q and L and an Edge set E this induces two separate two-hop graphs, one for either node-set. We define the two-hop graph for the node set Q but the same definition holds for the other node set. For all $q_1, q_2 \in Q$ with $q_1 \neq q_2$ and for all $l \in L$ such that $(q_1, l), (q_2, l) \in E$ the two hop graph $Q_2(Q, E_Q)$ is defined such that $(q_1, q_2) \in E_Q$.

This definition allows us to create a partitioning on these two-hop graphs with the strategies explained in Section 3.1.1.

An example of these graphs for the Q and L set of the example graph from Figure 2.1 can be seen in Figure 3.2. These graphs allow us to infer which nodes can be considered neighbors. For example, in Figure 3.2b we can see that nodes 5 and 6 have the common neighbors 1, 3, and 8 and thus can be considered neighbors. Referencing the original graph in Figure 2.1 one can see that all paths that go to node 6 have to go through node 5 as well making it reasonable to consider them being neighbored.

3.3 Mixed Integer Linear Programming

The mathematical foundation for Linear Programming (LP) reaches far back to the 19th century when Fourier first described an algorithm to solve a system of linear inequalities in 1824 [BT97]. A milestone for LP was the development of the Simplex method by

Dantzig [DOW55]. Although this method does not guarantee sub-exponential runtime for all problems it is still used in most solvers as it often works faster than other methods in practice. An algorithm that guarantees polynomial runtime is the interior point method developed by Karmarkar [Kar84]. This method often works slower than the Simplex method in practice and is thus not generally applicable.

Nowadays these algorithms are packaged into solver libraries like Gurobi[1] and can be efficiently applied to relevant problems.

Mixed Integer Linear Programming (MILP) is a generalized version of LP in which the domain of the solution variables can assume real and integer values. The standardized form for MILP problems can be seen in formulas (3.1)-(3.6) [Obs22]. MILPs are typically solved by a branch-and-bound framework in which the MILPs LP relaxation is solved for obtaining dual bounds. The LP relaxation of the MILP (3.1)-(3.6) is obtained by replacing the integrality constraints of (3.6) with $\mathbf{x_2} \in \mathbb{R}$. MILPs are $\mathcal{NP}$hard [GJ79] and thus MILP solvers are not useful to solve very large problem instances.

$$\max\ \mathbf{c_1'x_1} + \mathbf{c_2'x_2} \tag{3.1}$$
$$\mathbf{A_1x_1} + \mathbf{A_2x_2} \leq \mathbf{b} \tag{3.2}$$
$$\mathbf{x_1} \geq 0 \tag{3.3}$$
$$\mathbf{x_2} \geq 0 \tag{3.4}$$
$$\mathbf{x_1} \in \mathbb{R} \tag{3.5}$$
$$\mathbf{x_2} \in \mathbb{N} \tag{3.6}$$

$\mathbf{c_1'} \ldots n$-dimensional vector

$\mathbf{x_1} \ldots n$-dimensional vector

$\mathbf{c_2} \ldots p$-dimensional vector

$\mathbf{x_2} \ldots p$-dimensional vector

$\mathbf{A_1} \ldots m \times n$-dimensional Matrix

$\mathbf{A_2} \ldots m \times p$-dimensional Matrix

$\mathbf{b} \ldots m$-dimensional vector

When describing a problem in terms of a MILP, the constraint in formula (3.2) are often separated row-wise and depicted as their own inequalities to increase readability and interpretability. For further information on modeling problems as MILPs and respective solution techniques see the book by Bertsimas [BT97].

[1]www.gurobi.com

CHAPTER 4

Related Work

In this section, we are first going to discuss a previous project that has been carried out at TU Wien on which a large part of our implementation and theoretical work builds upon. We are then describing related location problems from literature and finally give an overview of how MLO has developed and what problems it has already been applied to.

4.1 Previous Work

Originally the Multi-Period Battery Swapping Station Location Problem (MBSSLP) has been formulated and addressed by Jatschka et al. [JRR23, RKJ$^+$23]. They investigate how to distribute battery swapping stations in an urban environment. Similar to our problem they try to find the optimal locations for battery swapping stations with a variable number of charging slots. Instead of working with a limited budget, they try to minimize the used budget while fulfilling a minimum amount of demand. As hinted in the name of the work they also consider multiple time steps in the charging process. As a consequence, not all batteries are always available as they need a certain amount of time to charge. Furthermore, the expected workload differs over time. For example in an urban environment, one can expect a high workload during the morning and evening hours but a lower workload around midday and night. Instead of considering a maximum detour length for customer trips, they apply a penalty to the satisfied demand meaning customers assigned to stations that would include a far detour are less likely to use the assigned station.

Using a Large Neighborhood Search [MG19] the authors were able to solve instances with up to 2000 locations and 8000 OD-pairs [RKJ$^+$23]. They later improved this approach by switching the solution method from LNS to MLO with which they solved instances with up to 10000 areas and 100000 OD-pairs [JRR23]. Their MLO approach is similar to our approach described in Chapter 5 but applies a rather simple fixed similarity measure, the Jaccard similarity, to which we will compare our results.

4.2 Related Location Problems

One of the first papers on location problems was by Cooper [Coo63] who introduced the p-median problem where the goal is to cover a set of nodes in a network with a limited amount of locations. Since then the field has developed massively and influenced among others on economics, geography, regional science, and logistics [LNS15].

Because of the variety of different types of location problems that exist we are going to focus here on only a select few that appear to be most related to our problem. The DMBSSLP falls under the category of Maximum Coverage Location Problems (MCLP). Therefore we are going to describe the work that has been done in the category of Covering Problems (CP) and then continue this description to MCLP which in itself is a subcategory of CP.

4.2.1 Covering Problems

Covering Problems (CP) deal with the problem of finding the best possible areas for stations such that a given demand is covered.

The following introduction is adapted from Laporte et al. [LNS15].

The idea for CP is that a node can only be considered covered if a station is within a certain reach. For example, when considering emergency services like medical services it is crucially important that every location can be reached within a limited time. An area can be considered as covered if it is within the reach of an emergency station hence comes the name covering problem.

Berge [Ber57] was the first to describe the problem of finding a minimum cover on a graph. Hakimi [Hak65] developed an algorithm to find the minimum number of police patrols required to protect a highway network. The first mathematical formulation of the problem in a location context was done by Toregas et al. [TSRB71]. Out of a Location context, it had already been formulated by Roth [Rot69].

There are two types of covering problems, set covering and maximal covering problems. While the goal of set covering problems is to cover all nodes, the goal of maximal covering problems is to cover a maximal amount of nodes. Because the DMBSSLP tries to maximize the satisfied demand it falls under the latter category.

4.2.2 Maximum Coverage Location Problem

MCLP have been applied to a broad field of different research topics among others are banking [XYD$^+$09, AMVC15], restaurants [EA16, YK16], battery swapping [Wan08], natural gas refueling stations [KBP14, BLM15] and urban parks [YCW$^+$14]

The original MCLP was formulation by ReVelle et al. [CR74] and is denoted in Equations (4.1)-(4.5). The objective (4.1) maximizes the fulfilled demand according to the population at node q if that node is covered, which is described by the binary variable a_q. Instead of using a_{ql} as we do in Section 2.2 to describe which area covers which demand, they use a_q to describe if the demand is covered at all. Constraint (4.2) ensures that a node is only covered if at least one neighboring station is built. This is achieved by only iterating over the neighboring l nodes for each q node. Constraint (4.3) is the equivalent of our budget constraint and ensures that not more than B stations are built. Equations (4.4) and (4.5) define the x_l and a_q variables as binary.

$$\max \sum_{q \in Q} d_q a_q \tag{4.1}$$

$$a_q \leq \sum_{l \in N(q)} x_l \qquad q \in Q \tag{4.2}$$

$$\sum_{l \in L} x_l \leq B \tag{4.3}$$

$$x_l \in \{0,1\} \qquad l \in L \tag{4.4}$$

$$a_q \in \{0,1\} \qquad q \in Q \tag{4.5}$$

When comparing this formulation to the DMBSSLP described in Section 2.2 we see a lot of similarities. Firstly the demand is being maximized as seen in formula (2.3) in the DMBSSLP formulation. The difference is that the satisfied demand is intrinsic to the a_{ql} variable itself in our formulation while in (4.1) a_q is just binary. Secondly in (2.6) the demand satisfied at an area a_{ql} is capacitated while in (4.2) it is just necessary for a station to be built anywhere in reach of q. Lastly, the budget constraint is different. In (4.3) a set number of stations can be built while in (2.7) each slot and each area have a cost and their sum must comply with the overall defined budget.

ReVelle et al. [CR74] use different heuristics to solve the MCLP. First, the Greedy Adding Algorithm iteratively adds stations that satisfy the most demand. The second heuristic which builds on the first heuristic is called the Greedy Adding with Substitution Algorithm. Additionally to adding the most promising station in each step, they apply a simple local search that tries to replace each already existing station with a better one. Lastly, they also solved the MCLP using a Linear Programming approach to obtain exact solutions.

They applied these methods to a network with 55 nodes. Where the Greedy Adding with Substitution Algorithm found solutions within a range of 90% of the optimum.

4.2.3 Capacitated Maximum Coverage Location Problem

The original MCLP does not consider the stations capacitated. This means that a station can serve any amount of demand. For real-world problems, it is often better to assume a station to be capacitated because after it exceeds its capacity it might not be able to adequately service the demand. Therefore the Capacitated Maximum Coverage Location Problem (CMCLP) is introduced. This problem was first described by Current and Storbeck [CS88] as the capacitated plant location problem. The formulation (4.6)-(4.12) is adapted from the formulation by Pirkul and Schilling [PS91].

$$\max \sum_{q \in Q} \sum_{l \in N(q)} d_q a_{ql} \tag{4.6}$$

$$a_{ql} \leq \sum_{l' \in N(q)} x_{l'} \qquad q \in Q, l \in N(q) \tag{4.7}$$

$$\sum_{l \in L} x_l \leq B \tag{4.8}$$

$$\sum_{l \in N(q)} a_{ql} d_q \leq y \qquad q \in Q \tag{4.9}$$

$$\sum_{q \in N(l)} a_{ql} \leq 1 \qquad l \in L \tag{4.10}$$

$$x_l \in \{0,1\} \qquad l \in L \tag{4.11}$$

$$a_{ql} \in \{0,1\} \qquad l \in L, q \in Q \tag{4.12}$$

The objective (4.6) maximizes the served population. The first constraint (4.7) defines that the demand of a q node can only be satisfied if a location is built on a neighboring l node. The next constraint (4.8) limits the number of stations that can be built. The capacity is introduced in constraint (4.9), this ensures that the demand satisfied at each station is not higher than y. The last constraint (4.10) ensures that each demand node is only served exactly once. The last two definitions (4.11) (4.12) (4.12) ensure that the x_l and a_{ql} variable are binary. The x_l is 1 if a station is to be built at location l. a_{ql} is 1 if the demand of an OD-pair q is to be satisfied at location l.

Instead of a_q describing if a node is covered at all this formulation now uses a_{ql} as we do in Section 2.2 to describe which area covers which demand. The difference between formula (4.10) and (4.7) to formula (4.2) can also be ascribed to a_{ql} now being a two-dimensional variable. The only real relevant difference is the introduced capacity y in formula (4.9). The difference to our formulation is that the capacity in (2.6) is variable while the capacity in (4.9) is constant.

Pirkul and Schilling [PS91] use a heuristic to generate solutions. The heuristic uses the lagrangian relaxation of the problem as a starting solution. They were able to solve problem instances of the CMCLP with up to 30 areas and 200 OD-pairs.

4.2.4 Modular Capacitated Maximum Coverage Location Problem

Another example from the literature that shares similarities with our problem is the Modular Capacitated Maximal Covering Location Problem [YM12] (MCMCLP). Here Yin and Mu introduce a variant of the CMCLP with modular capacities. An adapted version of their model is described in (4.13)-(4.18).

$$\max \sum_{q \in Q} \sum_{l \in N(q)} d_q a_{ql} \tag{4.13}$$

$$\sum_{l \in L} y_l = B \tag{4.14}$$

$$\sum_{q \in Q} d_q a_{ql} \leq y_l \qquad l \in L \tag{4.15}$$

$$\sum_{l \in L} a_{ql} \leq 1 \qquad q \in Q \tag{4.16}$$

$$y_l \in \{0 \dots \lceil \bar{d} \rceil\} \tag{4.17}$$

$$0 \leq a_{ql} \leq 1 \qquad q \in Q, l \in L \tag{4.18}$$

The objective (4.13) maximizes the satisfied demand. Constraint (4.14) is the budget constraint and ensures that exactly B slots are being built. The capacity constraint (4.15) enforces that the assigned demand to a location is not higher than the number of slots built at that location. The last constraint (4.16) ensures that the demand assigned from a demand node can not exceed the demand that is provided by the node. The domains of the y_l and a_{ql} variable can be seen in equations (4.17) and (4.18).

The most striking difference between the MCMCLP to the CMCLP is that instead of having stations with a fixed number of slots, the MCMCLP does not use stations at all but uses the slots of a station as a limited variable. This is already very close to the definition of slots used in the DMBSSLP.

Yin and Mu [YM12] apply the MCMCLP to Emergency Medical Service Region 10 in Georgia USA where they aim to optimally distribute ambulances. With a method called the service area spatial demand representation, they can formulate this problem in terms of an MCMCLP with 87 possible areas and 2721 OD-pairs. They create solutions for the problem by solving the MILP using CPLEX[1].

4.2.5 CMCLP with Heterogeneous Facilities, Vehicles and Setup Costs

The final model we are going to investigate here is that of CMCLP with heterogeneous facilities, vehicles, and setup costs by Gazani et.al. [HGNN21]. For the sake of similarity, we are going to call this problem the Modular Capacitated Maximum Coverage Location Problem with Setup Cost (MCMCLPSC). The problem they examine contains a superset of constraints considered in the DMBSSLP. Additionally to our constraints, they consider different vehicle and location types with different costs and capacities associated with them. They also consider an area constraint as the vehicles and facilities are bound by an area constraint.

[1]https://www.ibm.com/products/ilog-cplex-optimization-studio

By renaming, fixing certain variables, and deleting additional constraints in the problem we transform it into a variation of the DMBSSLP. The transformed problem formulation can be seen in equations (4.19)-(4.26)

$$\max \sum_{q\in Q}\sum_{l\in L} d_q a_{ql} \tag{4.19}$$

$$\sum_{l\in L} c_l x_l + b_l y_l \leq B \tag{4.20}$$

$$\sum_{l\in L} a_{ql} \leq 1 \qquad q \in Q \tag{4.21}$$

$$\sum_{q\in Q} d_q a_{ql} \leq y_l \qquad q \in Q \tag{4.22}$$

$$y_l \leq s x_l \qquad l \in L \tag{4.23}$$

$$a_{ql} \in \{0,1\} \qquad q \in Q, l \in L \tag{4.24}$$

$$x_l \in \{0,1\} \qquad l \in L \tag{4.25}$$

$$y_l \geq 0 \qquad l \in L \tag{4.26}$$

As we can see the constraints (4.19)-(4.23) are the same as in the DMBSSLP. The differences lie within the domains of the variables (4.24)-(4.26). The a_{ql} variable in the MCMCLPSC is binary meaning the demand of an OD-pair can be assigned to an area in total or not at all, whereas in the DMBSSLP the a_{ql} is continuous, meaning multiple different areas can satisfy the demand of a single OD-pair. In the MCMCLPSC the x_l variable is also binary. The x_l variable in the DMBSSLP is an integer but can be in a certain range of values. The capacities represented in the y_l variables are also different. In the MCMCLPSC it is continuous while in the DMBSSLP y_l is an integer and can only take a certain range of values.

Gazani et.al. [HGNN21] used a handcrafted heuristic as well as a Genetic Algorithm [MG19] to create solutions for their problem. They were able to produce good solutions for instances with up to 50 areas and 200 OD-pairs. These results are not directly comparable to our problem as they considered more constraints and their problem was more difficult to solve.

4.3 Related Work to MLO

The MLO method was first introduced in the 90s by Barnard et al. [BS94] and applied to improve the Recursive Spectral Bisection method. The MLO method was later adapted by Hendrickson et al. [HL$^+$95] and applied to the graph partitioning problem. Their adapted version looked similar to the MLO known today and already contained the steps of coarsening, solving, and extending.

Walshaw solidified this approach and showed that the MLO approach is applicable to a variety of optimization problems [Wal08]. Among those are the graph partitioning problem [Wal08], the traveling salesman problem [Wal02] where Walshaw presents a framework to apply the MLO method to other optimization problems, and the vehicle routing problem [RSW07] where Rodney et al. apply the MLO method to the vehicle routing problem. Other application fields are Network drawing and visualization, Image segmentation, Dimensionality reduction, Graph contraction, Classification, and Graph coloring [VFF$^+$20].

4.3.1 MLO for Bipartite Networks

As we explained in Section 3.2 the MLO approach needs to be extended to be applicable to bipartite networks. The extension of MLO to bipartite networks is relatively new and thus not so well established. A pioneer in this specific field is Alan Valejo as most papers considering explicitly MLO for bipartite networks list him as the main author.

Some relevant contributions that he made are the extension of the MLO algorithm to bipartite networks [VdOGFdAL18] and the development and continuation of the label propagation coarsening method for bipartite networks [VAdPF$^+$21, VFdOdAL20] as well as relevant review papers about the different coarsening methods in bipartite networks [VdOdSNZ21] and a review of the MLO method, in general, [VFF$^+$20].

CHAPTER 5

A Learning Multi Level Optimization Approach for the DMBSSLP

In this chapter, we are going to explain the Learning Multilevel Optimization approach that we developed for the DMBSSLP. First, we give an overview of our algorithmic approach and define a nomenclature. Secondly, we give a detailed explanation of all the major steps of MLO: partitioning, coarsening, solving, projection, and refinement. We continue with the different similarity measures that are learned and compared. Second to last we take a look at the instance generation and finally, we explain the different data generation strategies.

5.1 Overview and Nomenclature

The MLO follows the basic scheme described in Section 3.1 and 3.2. To define the concrete approach we first need to make some basic definitions.

First we define the graph sequence $\{G^0, \dots, G^K\}$ of G with $G^i = (Q^i, L^i, E^i)$, for $i = 0, \dots, K$ where K is the number of coarsened graphs. This sequence represents all intermediate graphs that are created during the coarsening. The graph $G(Q, L, E)$ which is the original graph corresponds to $G^0(Q^0, L^0, E^0) = G(Q, L, E)$. The graph $G^K(Q^K, L^K, E^K)$ corresponds to the coarsest problem instance. Each intermediate graph has the same properties r^i_l, and $\bar{d}^i_l$ for its nodes $l \in L^i$, values d^i_q for nodes $q \in Q^i$, and values e^i_{ql} for the edges $(q, l) \in E^i$. Variables marked with $(\cdot)^i$ describe properties of the respective graph of G^i.

The graphs G^{i+1} with $i \in \{0, \dots, K-1\}$ are derived from G^i by coarsening Q^i and L^i and merging all nodes in each partition. Each vertex $q \in Q^{i+1}$ and $l \in L^{i+1}$ has an associated partition represented by a non-empty subset in Q^i and L^i. These subsets are denoted as Q^i_q and L^i_l where the nodes q and l represent nodes from the coarse new sets Q^{i+1} and L^{i+1}. These subsets Q^i_q and L^i_l refer to the partition of G^i. As a node can not be partitioned twice it must hold that $Q^i_q \cap Q^i_{q'} = \emptyset$ and $L^i_l \cap L^i_{l'} = \emptyset$ for any $q, q' \in Q^i$, $q \neq q'$ and $l, l' \in L^i$, $l \neq l'$, and $i = 1, \dots, K-1$. At last, if there is at least one edge between the nodes Q^i_q and L^i_l in G^i there must also be an edge (q, l) in E^{i+1}.

We are dealing with a problem defined on a bipartite network, therefore, we coarsen and project the nodes in L and Q separately. The processes are split into two steps. For the coarsening from G^i to G^{i+1} we initially only coarsen the nodes in L^i. This yields an intermediate graph $\tilde{G}^{i+1}$. Next, we coarsen only the nodes $\tilde{Q}^{i+1}$ of the $\tilde{G}^{i+1}$ graph which are the same as Q^i, which yields the graph G^{i+1}. With this two-step process, we obtain a full coarsening from G^i to G^{i+1}.

The same follows for the projection. First, we project the solution regarding the Q^i nodes in G^i. This yields a solution for $\tilde{G}^i$. We now project the solution regarding the nodes of $\tilde{L}^i$ of the $\tilde{G}^i$ graph which are the same as L^i to obtain the solution for G^{i-1}. Variables marked with $\tilde{(\cdot)}$ are associated with the intermediate graph $\tilde{G}$. For example, the description $\tilde{N}^i(q)$ refers to the neighbors of node $q \in \tilde{Q}^i$ in the graph $\tilde{G}^i$. Similar we denote the intermediate solutions x^i, y^i, a^i for the problem G^i.

A small difference to the earlier MLO versions we described is that we use a while loop and a shrinkage factor ξ. ξ defines the percentage of the original problem size to which the smallest graph should be coarsened. The LMLO algorithm is denoted in Algorithm 5.1. A detailed explanation of all steps is described in the following sections.

5.2 LMLO Details

In this section, we are going to explain the details of the LMLO implementation for the DMBSSLP. This involves explanations for all major steps: partitioning, coarsening, solving, projection, and refinement as they are listed in Algorithm 5.1.

5.2.1 Partitioning

To create a partitioning we use Greedy Heavy Edge Matching [VFF+20] (see Section 3.1.1). The DMBSSLP is defined on a bipartite graph we therefore calculate the similarity for all neighboring nodes in the two-hop graph (see Section 3.2). An explanation of how the similarity calculation works is described in Section 5.3.

The first partitioning step on Line 4 in Algorithm 5.1 creates a partition for the nodes in L^i. Every node $l \in L^i$ is paired with all its neighbors in the two-hop graph and their similarity is calculated. All pairs are then sorted according to their similarity. A node pair (u, v) is put into the partition if both nodes are not partitioned and it has a higher similarity than other pairs (u, x) or (v, x) where x is also an unpartitioned node. To realize this sorting we use a max-heap which allows us to efficiently access the best

Algorithm 5.1: LMLO

```
    Input : a DMBSSLP instance $G(Q, L, E)$, a value $0 \le \xi \le 1$
    Output: a solution $(x^0, y^0, a^0)$
 1: $i \leftarrow 0$;
 2: $G^i \leftarrow G$;
 3: while $|Q^i| > \xi \cdot |Q| \wedge |L^i| > \xi \cdot |L|$ do
 4:     $L^i_l \leftarrow$ derive partitioning of $L^i$ in $G^i$;
 5:     $\tilde{G}^{i+1} \leftarrow$ Coarsen($G^i$) w.r.t. $L^i_l$;
 6:     $Q^i_q \leftarrow$ derive partitioning of $\tilde{Q}^{i+1}$ in $\tilde{G}^{i+1}$;
 7:     $G^{i+1} \leftarrow$ Coarsen($\tilde{G}^{i+1}$) w.r.t. $Q^i_q$;
 8:     $i \leftarrow i + 1$;
 9: end while
10: $(x^i, y^i, a^i) \leftarrow$ solve problem w.r.t. $G^i$;    // coarsest problem
11: while $i > 0$ do
12:     $(\tilde{x}^i, \tilde{y}^i, \tilde{a}^i) \leftarrow$ project solution $(x^i, y^i, a^i)$ for $G^i$ to $\tilde{G}^i$;
13:     $(x^{i-1}, y^{i-1}, a^{i-1}) \leftarrow$ project solution $(\tilde{x}^i, \tilde{y}^i, \tilde{a}^i)$ for $\tilde{G}^i$ to $G^{i-1}$;
14:     $(x^{i-1}, y^{i-1}, a^{i-1}) \leftarrow$ refine solution $(x^{i-1}, y^{i-1}, a^{i-1})$
15:     $i \leftarrow i - 1$;
16: end while
17: return $(x^0, y^0, a^0)$;
```

element. In Line 6 the same partitioning process is applied to the $\tilde{Q}$ nodes. Nodes that are not partitioned after the procedure are left as is. They will not be coarsened but will remain as they are in the following step.

The pseudo code representation of this Algorithm can be seen in 5.2.

5.2.2 Coarsening

We now explain the procedure for obtaining a coarsened graph G^i from a given graph G^{i-1}. Recall that we denote a partition of nodes of G^{i-1} as Q^{i-1}_q and L^{i-1}_l, respectively, with $q \in Q^i, l \in L^i$, for $i = 1, \ldots, K$.

When contracting nodes, one has to also aggregate the associated node and edge properties appropriately. We first coarsen the set L^{i-1} of the graph G^{i-1} to obtain an intermediate graph $\tilde{G}^i$ and then in a second step coarsen the set $\tilde{Q}^i$ of the graph $\tilde{G}^i$ to obtain the coarse graph G^i.

Coarsening for L

In the inital step we merge all nodes $l_1, l_2 \in L^{i-1}$ that are partitioned in L^{i-1}_l. The relevant properties that need handling are all properties of the areas: r^i_l, $N^i(l)$, $\overline{d^i_l}$ as well as properties associated with edges: e^i_{ql}.

Algorithm 5.2: Partitioning

```
Input : a set of nodes $U$
Output: a partitioning $M(U)$
1: $S$ ←empty max heap;
2: for $u \in U$ do
3:     for $v \in neighborhood(u)$ s.t. $(u, v) \notin S$ do
4:         $S(u, v) = similarity(u, v))$;
5:     end for
6: end for
7: $M \leftarrow \emptyset$;
8: $unpartitioned \leftarrow U$;
9: while $S$ is not empty do
10:    $(u, v) \leftarrow$ pop pair with highest similarity from $S$;
11:    if $u, v \in unpartitioned$ then
12:        $M \leftarrow M \cup (u, v)$;
13:        remove $u, v$ from $unpartitioned$;
14:    end if
15: end while
16: add all nodes in $unpartitioned$ to $M$ partitioned just to themselves ;
17: return $M$;
```

$$\tilde{e}^i_{ql} = \min\left(\sum_{l' \in L^{i-1}_l} e^{i-1}_{ql'},\ d^{i-1}_q\right), \tag{5.1}$$

Equation (5.1) describes $\tilde{e}^i_{ql}$ which represents the maximal demand that can be assigned from an OD-pair q to a new area l. The new $\tilde{e}^i_{ql}$ is composed of the sum of old $e^{i-1}_{ql'}$ or if that sum is now higher than the provided demand of OD-pair, the total demand d^{i-1}_q of an OD-pair. If two areas are merged they now should be able to serve the same demand along the same connections that the old areas were able to serve and only if this demand can not be delivered from the OD-pair should the new demand be capped.

When two nodes $l_1, l_2 \in L^{i-1}$ are merged to a node $l \in \tilde{L}^i$ the new node l is now neighbored to all nodes $\tilde{N}^i(l) = (N^{i-1}(l_1) \cup N^{i-1}(l_2))$ meaning the new node l is neighbored to all nodes either of the original nodes l_1 or l_2 were neighbored to. Note that $\tilde{Q}^i = Q^{i-1}$ because only the L^{i-1} set was coarsened yet.

The new maximum demand that can be fulfilled by the stations in an area $l \in \tilde{L}^i$ is determined by Formula (5.2).

$$\tilde{\overline{d}}^i_l = \min\left(\sum_{l' \in L^{i-1}_l} \overline{d}^{i-1}_{l'},\ \sum_{q \in \tilde{N}^i(l)} \tilde{e}^{i-1}_{ql}\right). \tag{5.2}$$

The new area can serve as much demand as both old areas with the caveat that if the neighboring OD-pairs can not deliver enough demand for the new area the maximum satisfiable demand is bound by the demand that the OD-pairs can deliver.

Finally, maximum station numbers are derived by Formula (5.3).

$$\tilde{r}_l = \left\lceil \frac{\tilde{\tilde{d}}_l}{s} \right\rceil . \tag{5.3}$$

This ensures that at an area l only so many slots and stations can be built that all possible demand is satisfiable.

Coarsening for Q

After a partitioning Q_q^{i-1} for the nodes of $\tilde{Q}^i$ of the graph $\tilde{G}^i$ has been derived we can coarsen the nodes $q_1, q_2 \in \tilde{Q}^i$ to the new node $q \in Q^i$. The relevant properties that need handling are the features of the OD-pairs: d_q, $N(q)$ as well as properties associated with edges: e_{ql}.

For each edge $(q, l) \in \tilde{E}^i$ the maximum assignable demand is calculated by Equation (5.4).

$$e_{ql}^i = \min \left(\sum_{q' \in Q_q^{i-1}} \tilde{e}_{q'l}^i , \tilde{\tilde{d}_l^i} \right) \tag{5.4}$$

If only one of the old nodes q_1, q_2 served an area l the maximum assignable demand along that edge should not increase. If both OD-pairs were able to serve the area the demand should now be the sum of them unless the area can not handle that much demand in which case the second condition takes over and the maximum assignable demand is capped at $\tilde{\tilde{d}_l^i}$

When two nodes $q_1, q_2 \in \tilde{Q}^i$ are merged to a node $q \in Q$ the new node q is now neighbored to all nodes $N^i(q) = (\tilde{N}^i(q_1) \cup \tilde{N}^i(q_2))$. Note that $\tilde{L}^i = L^i$ because only the $\tilde{Q}^i$ set is coarsened in this step. The new q node is now neighbored to all nodes l that either l_1 or l_2 were neighbored to before.

The final property for which a coarsening needs to be defined is the demand of an OD-pair. The demands of the merged OD-pairs are aggregated by taking the respective sums for all $q \in Q_q^{i-1}$ while also considering the maximal amount of demand that can be assigned to stations at all adjacent areas $l \in \tilde{N}^i(q)$. A Formula for this can be seen in (5.5).

$$d_q^i = \min \left(\sum_{q' \in Q_q^i} \tilde{d}_{q'}^i , \sum_{l \in \tilde{N}^i(q)} \tilde{e}_{ql}^i \right) . \tag{5.5}$$

When two OD-pairs $q_1, q_2 \in \tilde{Q}^i$ are coarsened their demands are added up, unless this would cause an overflow in the assigned demands along the adjacent edges in which case the e_{ql} are the limiting factor.

5.2.3 Solving

With the partitioning and coarsening being repeatedly applied we obtain the smallest problem instance G^K. We can now apply a solver using the MILP-formulation from Section 2.2. This will produce an inital solution (x^i, y^i, a^i) for the problem G^K. In the following sections, we will explain how to project this solution to the less coarse instance G^{i-1}.

5.2.4 Projection

In this section we describe how a solution (x^i, y^i, a^i) to a graph G^i is projected to graph G^{i-1}.

The projection of a solution is done in two steps: First projecting from G^i to $\tilde{G}^i$ and then further projecting to G^{i-1}. Both of these projections are done by solving multiple smaller subproblems, one subproblem for each node that was obtained during the partitioning and coarsening.

A solution for the problem instance G^i, is $x^i = (x^i_l)_{l \in L^i}$ with $x^i_l \in \{0, \dots, r^i_l\}$, $y^i = (y^i_l)_{l \in L^i}$ with $y^i_l \in \{0, \dots, \lceil \bar{d}^i_l \rceil\}$ and $a^i = (a^i_{ql})_{q \in Q^i, l \in L^i}$ with $0 \leq a^i \leq e^i_{ql}$. We describe the intermediate solution on the graph $\tilde{G}^i$ with $\tilde{x}^i, \tilde{y}^i$, and $\tilde{a}^i$.

Projecting from G^i to $\tilde{G}^i$

To project a solution (x^i, y^i, a^i) given on graph G^i to obtain a solution $(\tilde{x}^i, \tilde{y}^i, \tilde{a}^i)$ on graph $\tilde{G}^i$ we need to undo the coarsening of the Q^i nodes. This is done by solving the following independent LP for each $q \in Q^i$:

$$\max \sum_{q' \in Q^{i-1}_q} \sum_{l \in \tilde{N}^i(q')} \tilde{a}^i_{q'l} \tag{5.6}$$

$$\sum_{l \in \tilde{N}^i(q')} \tilde{a}^i_{q'l} \leq \tilde{d}^i_{q'} \qquad q' \in Q^{i-1}_q \tag{5.7}$$

$$\sum_{q' \in \tilde{N}^i(l) \cap Q^{i-1}_q} \tilde{a}^i_{q'l} \leq a^i_{ql} \qquad l \in N^i(q) \tag{5.8}$$

$$0 \leq \tilde{a}^i_{q'l} \leq \tilde{e}_{q'l} \qquad q' \in Q^{i-1}_q,\ l \in \tilde{N}^i(q) \tag{5.9}$$

The objective function (5.6) of the LP maximizes the satisfied demand. Constraints (5.7) ensure that assigned demand does not exceed an OD-pair's available demand. Formula (5.8) ensure that the total demand assigned to each area $l \in N^i(q)$ does not exceed the amount of demand assigned to l in solution (x^i, y^i, a^i). The final constraint (5.9) gives the domain of the $\tilde{a}^i_{ql}$ variable. Because of constraint (5.8) the total number of battery slots required in an area does not increase when projecting the solution to $\tilde{G}^i$. Therefore, all these subproblems can be solved independently of each other.

Projecting from $\tilde{G}^i$ to G^{i-1}

When the solution is projected from $\tilde{G}^i$ to G^{i-1}, we again have one subproblem for each $l \in \tilde{L}^i$. When projecting the demand assignment of an area $l \in \tilde{L}^i$ to the areas $l' \in L_l^{i-1}$ not all of the demand assigned to l can necessarily also be assigned to the areas l'. But there may also be unused demand that could potentially be satisfied in the areas l'. This interaction results in dependence between the subproblems. One subproblem might result in some demand being left unsatisfied and another subproblem can now use this unsatisfied demand to increase the overall demand it satisfies. The order in which the problems will be solved now does affect the results. Some preliminary tests showed that solving the areas first that have the highest amount of fulfilled demand per cost yields the best results.

The subproblem for $l \in \tilde{L}^i$ consider the areas L_l^{i-1} and the OD-pairs $q \in \tilde{N}^{i-1}(l)$. We define a helper variable δ_q which represents the unassigned demand of an OD-pair q. For details on how this value is calculated, we refer to Algorithm 5.3. A solution for each subproblem can be obtained by solving the MILP (5.10)-(5.17).

$$\max \sum_{l' \in L_l^{i-1}} \sum_{q \in N^{i-1}(l')} a_{ql'}^{i-1} \tag{5.10}$$

$$s x_{l'}^{i-1} \geq y_{l'}^{i-1} \qquad l' \in L_l^{i-1} \tag{5.11}$$

$$\sum_{l' \in L_l^{i-1} \cap N^{i-1}(q)} a_{ql'}^{i-1} \leq \delta_q \qquad q \in N^{i-1}(l) \tag{5.12}$$

$$\sum_{q \in N^{i-1}(l')} a_{ql'}^{i-1} \leq y_{l'}^{i-1} \qquad l' \in L_l^{i-1} \tag{5.13}$$

$$\sum_{l' \in L_l^{i-1}} (c x_{l'}^{i-1} + b y_{l'}^{i-1}) \leq c \tilde{x}_l^i + b \tilde{y}_l^i \tag{5.14}$$

$$x_{l'}^{i-1} \in \{0, \dots, r_{l'}^{i-1}\} \qquad l' \in L_l^{i-1} \tag{5.15}$$

$$y_{l'}^{i-1} \in \{0, \dots, \lceil \bar{d}_{l'}^{i-1} \rceil\} \qquad l' \in L_l^{i-1} \tag{5.16}$$

$$0 \leq a_{ql'}^{i-1} \leq e_{ql'}^{i-1} \qquad l' \in L_l^{i-1},\ q \in N^{i-1}(l') \tag{5.17}$$

This model is derived from the original MILP (2.3)-(2.10). Slight differences are in Equation (5.12) and (5.14). Inequality (5.14) restricts what is spent in $(x^{i-1}, y^{i-1}, a^{i-1})$ in areas L_l^{i-1} to the corresponding costs in the solution to $\tilde{G}^i$, i.e., $c\tilde{x}_l^i + b\tilde{y}_l^i$. Inequality (5.12) restricts the demand of each OD-pair $q \in N^{i-1}(l)$ that may at most be covered by adjacent areas in L_l^{i-1} to δ_l. The demand δ_q, for $q \in N^{i-1}(l)$, consists of the demand of $\tilde{a}_{ql}$ fulfilled in l in $\tilde{G}^i$ and additionally the so far uncovered demand of q, $\tilde{d}_q - \sum_{l \in \tilde{N}^i(q)} \tilde{a}_{ql}$ w.r.t. any area in $\tilde{L}^i$.

Algorithm 5.3 shows how the areas in $\tilde{L}$ are sequentially projected. The core part of this method is how δ_q is updated in each step. The other parts consist of solving the corresponding MILP and bookkeeping the different variables.

Algorithm 5.3: Project Solution from $\tilde{G}^i$ to G^{i-1}

```
   Input : a solution $(\tilde{x}^i, \tilde{y}^i, \tilde{a}^i)$ to $\tilde{G}^i$
   Output: a $(x^{i-1}, y^{i-1}, a^{i-1})$ solution to $G^{i-1}$
1: $(x^{i-1}, y^{i-1}, a^{i-1}) \leftarrow$ empty solution w.r.t. $G^{i-1}$;
2: $\delta_q \leftarrow \left(\tilde{d}_q - \sum_{l \in \tilde{N}^i(q)} \tilde{a}_{ql}\right); \quad \forall q \in \tilde{Q}^i$   // unassigned demand w.r.t. $\tilde{G}^i$
3: for $l \in \tilde{L}^i$ according to decreasing $\frac{\sum_{q \in N^i(l)} \tilde{a}_{ql}}{\tilde{x}_l c + \tilde{y}_l b}$ do
4:     $\delta_q \leftarrow \delta_q + \tilde{a}_{ql} \quad \forall q \in \tilde{N}^i(l)$;   // demand available for areas in $L_l^{i-1}$
5:     $(x', y', a') \leftarrow$ solve$(L_l^{i-1}, \tilde{N}^i(l), \delta)$;  // apply MILP (5.10)–(5.17)
6:     $x_{l'}^{i-1} \leftarrow x'_{l'}$, $y_{l'}^{i-1} \leftarrow y'_{l'} \quad \forall l' \in L_l^{i-1}$;
7:     $a_{q'l'}^{i-1} \leftarrow a'_{q'l'}$, $\delta_{q'} \leftarrow \delta_{q'} - a_{q'l'} \quad \forall l' \in L_l^{i-1} \quad \forall q' \in N^{i-1}(l')$;
8: end for
9: return $(x^{i-1}, y^{i-1}, a^{i-1})$;
```

During the procedure δ_q is repeatedly updated on the go for each area. At first in Line 2 the variable δ_q is initialized with the demand of each OD-pair $q \in \tilde{Q}^i$. From these values the sum of the already allocated demands of the previous solution $\tilde{a}^i$ is subtracted. This leaves us with the so far not allocated demand. In each iteration of the for-loop in Line 4 δ_q is updated to include the demand that was assigned to the area l. This allows the MILP (5.10)-(5.17) to allocate the same amount of demand that was already allocated in the previous solution $\tilde{a}_{ql}^{i-1}$ plus the so far unaccounted demand with which δ_q was initialized with. Lastly in Line 7 the newly allocated demand is again subtracted from δ_q. If not all demand is assigned again in the current solution it will therefore remain in δ_q and can thus be allocated when projecting another area l.

The order in which the areas l are solved is according to non-decreasing efficiencies $\frac{\sum_{q \in N^i(l)} \tilde{a}_{ql}}{\tilde{x}_l c + \tilde{y}_l b}$ reflecting the potentially fulfillable demand per cost of the stations.

5.2.5 Refinement

The final step of the MLO Algorithm 5.1 is the refinement. In this step, a greedy Construction Heuristic (CH), which can also be used as a stand-alone to find solutions, is applied to improve the so-far found solution. During the projection, it happens that the optimal solution uses less budget than available. In this case, the budget will be left over in the end. Applying the CH improves the solution by using this leftover budget to build additional stations and slots. In Algorithm 5.4 we describe how the CH for the refinement works.

At first, we reallocate the a_{ql}^i values in Line 1 according to the LP in (5.18) - (5.21). After the different subproblems for l and q are solved the demands might be distributed unfavorable. By fixing the x and y values and solving the LP just for the a values it can

Algorithm 5.4: Construction Heuristic

```
    Input : an unrefined solution $(x^i, y^i, a^i)$ to $G^i$
    Output: a refined solution $(x^i, y^i, a^i)$ to $G^i$
 1: Reallocate $a^i$ by solving LP (5.18) - (5.21) ;
 2: $B_{left} \leftarrow B - \sum_{l \in L^i}(x_l^i c + y_l^i b)$
 3: $\Delta d_l \leftarrow \bar{d}_l - \sum_{q \in N^i(l)} a_{ql} \quad \forall l \in L^i$
 4: $\Delta B_l \leftarrow (r_l - x_l)c + (\lceil \bar{d}_l \rceil - y_l)b \quad \forall l \in L^i$
 5: for $l \in L^i$ according to decreasing $\frac{\Delta d_l}{\Delta B_l}$ do
 6:     $y_{new} \leftarrow \min(sx_l^i - y_l^i, \lceil \bar{d}_l \rceil - y_l, \lfloor \frac{B_{left}}{b} \rfloor)$
 7:     $B_{left} \leftarrow B_{left} - y_{new} \cdot b$
 8:     $y_l^i \leftarrow y_l^i + y_{new}$
 9:     while $x_l^i < r_l^i \wedge (c + b) > B_{left}$ do
10:         $x_l^i \leftarrow x_l^i + 1$
11:         $B_{left} \leftarrow B_{left} - c$
12:         $y_{new} \leftarrow \min(sx_l^i - y_l^i, \lceil \bar{d}_l \rceil - y_l, \lfloor \frac{B_{left}}{b} \rfloor)$
13:         $B_{left} \leftarrow B_{left} - y_{new} \cdot b$
14:         $y_l^i \leftarrow y_l^i + y_{new}$
15:     end while
16:     allocate $a^i$
17: end for
18: Reallocate $a^i$ by solving LP (5.18) - (5.21) ;
19: return $(x, y, a)$;
```

be possible to improve the solution slightly. Then the leftover Budget is calculated in Line 2. The For loop in Line 5 iterates over the l values in L^i such that the locations which offer the most demand for the cheapest price are considered first. Then as many battery charging slots as possible are built. In the while loop in Line 9 additional stations and slots are built until no more can be built. In Line 16 the demand of the neighboring OD-pairs is assigned to a^i. This process is repeated until all areas have been considered. Finally in the last step in Line 18 the a^i values are once again reallocated according to LP (5.18) - (5.21). This reallocation is necessary as it is not checked during the CH if some demand has been assigned twice.

$$\max \sum_{q \in Q^i} \sum_{l \in N^i(q)} a^i_{ql} \tag{5.18}$$

$$\sum_{l \in N^i(q)} a^i_{ql} \leq d^i_q \qquad q \in Q^i \tag{5.19}$$

$$\sum_{q \in N^i(l)} a^i_{ql} \leq y^i_l \qquad l \in L^i \tag{5.20}$$

$$0 \leq a^i_{ql} \leq e^i_{ql} \qquad q \in Q^i,\ l \in N^i(q) \tag{5.21}$$

5.3 Similarity

Similarity measures are used to quantify how similar two nodes of the network are. As we are creating a partitioning on a bipartite graph we always consider the similarity for two nodes that are neighbored considering the two-hop neighborhood. Recall that in contrast to the two-hop neighborhood where we consider nodes $q_1, q_2 \in Q$ neighbored if they are neighbored in the respective two-hop graph, the neighbors considering the variable $N(q)$ for $q \in Q$ are in the L set.

In this section, we give an overview of different similarity measures that can be used to create the partitioning. At first, we explain the Jaccard similarity which is a conventional similarity measure that has been used by Jatscka et al.[JRR23] in a previous MLO approach to a similar problem. Secondly, we are going to define a machine learning based similarity calculation that is based on two neural networks that predict the similarity of two nodes based on handcrafted input features.

5.3.1 Jaccard Similarity

The Jaccard similarity was used by Jatschka et al. [JRR23] in an MLO approach for the Multi Period Battery Swapping Station Problem the equation for the Jaccard similarity can be seen in Equation (5.22). The Jaccard similarity also called Intersection over Union similarity compares the number of common neighbors to the total number of neighbors.

$$Jaccard(q_1, q_2) = \frac{|N(q_1) \cap N(q_2)|}{|N(q_1) \cup N(q_2)|} \tag{5.22}$$

An interpretation of this measure might be that if nodes $q_1, q_2 \in Q$ have a high Jaccard similarity they have a lot of common neighboring areas in the L set. Meaning a station built at area $l \in L$ that is neighbored to q_1 will likely also be neighbored to q_2. Meaning both OD-pairs q_1 and q_2 can potentially satisfy their demand at that station. The other way around if two nodes $l_1, l_2 \in L$ have a high similarity, they have a lot of similar OD-pairs as neighbors. A station built at either l_1 or l_2 can thus fulfill most of the demand that the other station would also be able to fulfill.

5.3.2 Learned Similarity

The goal is to train two neural networks (NN), one for the Q similarity and one for the L similarity. These NNs will use handcrafted input features reflecting the properties of a problem instance to calculate a similarity measure between the two input nodes. Those features contain properties of the finer problem instance as well as properties of the potential coarsened problem instance. The models will be trained using the Mean Squared Error (MSE) as a loss function and the ADAM optimizer [KB14]. The MSE is not applied to the output directly as this would increase the influence of the far outliers existing in the distributions of $z_{l_1 l_2}$, $z_{q_1 q_2}$ and z_j. Instead, we apply the symlog transformation [HPBL23] as seen in Equation (5.23) to both the predicted output of the network and ground truth and calculate the MSE on the difference between the symlog values.

$$\operatorname{symlog}(x) := \operatorname{sign}(x) \log(x+1) \tag{5.23}$$

Details on architecture, as well as training data generation, follow in Section 6.4.

Neural Network Features for L

We describe the neural network features in terms of a problem instance $G^i(Q^i, L^i, E^i)$ and the respective coarsened problem instance $\tilde{G}^{i+1}(\tilde{Q}^{i+1}, \tilde{L}^{i+1}, \tilde{E}^{i+1})$ where the L^i nodes have been coarsened.

The nodes $l, l' \in L^i$ are neighbored considering the two-hop neighborhood of L^i in $G^i(Q^i, L^i, E^i)$. The prospective future node they would be coarsened into is the node $l'' \in \tilde{L}^{i+1}$. We denote the vector of input features as $h^i_{L,l,l'}$. The input features of the neural network are:

1. r^i_l, $\bar{d}^i_l$
2. $r^i_{l'}$, $\bar{d}^i_{l'}$
3. $\tilde{r}^{i+1}_{l''}$, $\tilde{\bar{d}}^{i+1}_{l''}$
4. $\sum_{q \in N^i(l) \cap N^i(l')} d_q$, $\sum_{q \in N^i(l) \Delta N^i(l')} d_q$, $\frac{\sum_{q \in N^i(l) \cap N^i(l')} d_q}{\sum_{q \in N^i(l) \cup N^i(l')} d_q}$
5. $\min(\sum_{q \in N^i(l)} e^i_{ql}, \bar{d}^i_l)$, $\min(\sum_{q \in N^i(l')} e^i_{ql'}, \bar{d}^i_{l'})$, $\min(\sum_{q \in \tilde{N}^{i+1}(l'')} \tilde{e}^{i+1}_{ql''}, \tilde{\bar{d}}^{i+1}_{l''})$
6. $|N(l) \cap N(l')|$, $\frac{|N(l) \cap N(l')|}{|N(l) \cup N(l')|}$
7. s, c, i

Features 1, 2 and 3 represent the distinctive features of each node l, l', l''. Features 4 represent the possible demand combinations of the neighboring q nodes to l, l'. Features 5 represent the possible demand for an area l, l', l''. Features 6 are the number of common neighbors as well as the Jaccard similarity. Features 7 are global properties of the problem instance as well as the current level in the MLO that is being handled.

Neural Network Features for Q

We describe the neural network features in terms of a problem instance $\tilde{G}^i(\tilde{Q}^i, \tilde{L}^i, \tilde{E}^i)$ and the respective coarsened problem instance $G^i(Q^i, L^i, E^i)$ where the $\tilde{Q}^i$ nodes have been coarsened.

The nodes $q, q' \in \tilde{Q}^i$ are neighbored considering the two-hop neighborhood of $\tilde{Q}^i$ in $\tilde{G}^i(\tilde{Q}^i, \tilde{L}^i, \tilde{E}^i)$. The prospective future node they would be coarsened into is the node $q'' \in Q^i$. We denote the vector of input features as $h^i_{Q,q,q'}$. The input features of the neural network are:

1. $\tilde{d}^i_q$, $\tilde{d}^i_{q'}$, $d^i_{q''}$
2. $\sum_{l\in\tilde{N}^i(q)\cap\tilde{N}^i(q')} \tilde{\bar{d}}^i_l$, $\sum_{l\in\tilde{N}^i(q)\Delta\tilde{N}^i(q')} \tilde{\bar{d}}^i_l$, $\frac{\sum_{l\in\tilde{N}^i(q)\cap\tilde{N}^i(q')} \tilde{\bar{d}}^i_l}{\sum_{l\in\tilde{N}^i(q)\cup\tilde{N}^i(q')} \tilde{\bar{d}}_l}$
3. $\sum_{l\in\tilde{N}^i(q)\cap\tilde{N}^i(q')} \tilde{r}^i_l$, $\sum_{l\in\tilde{N}^i(q)\Delta\tilde{N}^i(q')} \tilde{r}^i_l$, $\frac{\sum_{l\in\tilde{N}^i(q)\cap\tilde{N}^i(q')} \tilde{r}^i_l}{\sum_{l\in\tilde{N}^i(q)\cup\tilde{N}^i(q')} \tilde{r}^i_l}$
4. $\min(\sum_{l\in\tilde{N}^i(q)} \tilde{e}^i_{ql}, \tilde{d}^i_q)$, $\min(\sum_{l\in\tilde{N}^i(q')} \tilde{e}^i_{q'l}, \tilde{d}^i_{q'})$, $\min(\sum_{l\in N^i(q'')} e^i_{q''l}, d^i_{q''})$
5. $|\tilde{N}^i(q) \cap \tilde{N}^i(q')|$, $\frac{|\tilde{N}^i(q)\cap\tilde{N}^i(q')|}{|\tilde{N}^i(q)\cup\tilde{N}^i(q')|}$
6. i

Features 1 are the demands of the nodes q, q', q''. Features 2 represent different set combinations of $\tilde{\bar{d}}^i$ in the neighboring nodes of q and q' in the L^i set. Feature 3 is similar it contains information of $\tilde{r}^i_l$ in different set combinations of the neighboring nodes in the L^i set. Feature 4 is the demand an OD-pair q, q', q'' can maximally assign to any neighboring area l. The second to last Feature 5 contains the Jaccard similarity as well as the total number of common neighbors of two nodes. Lastly, Feature 6 is the level information of the MLO.

5.4 Instance Generation

For training the model that represents the similarity measure we need to create a sizeable amount of different representative problem instances. To create a DMBSSLP instance $G(Q, L, E)$ we define $|L| = n$ and $|Q| = m$. Before we create the bipartite network we create a graph $P(V, E)$ as discussed in Section 2.1. The steps that are carried out to create a problem instance are explained in the following paragraphs. Our approach reuses the approach from Jatschka et al. [JORR20] where a detailed explanation of the instance creation for MBSSLP is given. We use a slightly adapted version of this to create instances for the DMBSSLP.

First, n points with coordinates (x, y) uniformly distributed in an euclidean square of size $800\sqrt{n}$ representing the areas V are created.

The primal edges E of $P(V, E)$ are composed of an euclidean spanning tree (EST). This ensures that all areas are connected. The EST is calculated by first creating the intermediate Graph containing the Delaunay triangulation [Del34] of the areas V and secondly calculating an EST on the Delaunay graph.

Additionally n edges $(u, v) \in V \times V$ are added. These edges represent faster connections between areas e.g. direct connections in an urban street network.

We fill the instance with the s, c, b values. In the MBSSLP the c values are associated with an area itself and thus can differ. The s values which represent the maximum battery slots per station are set to 70. This value is hardly ever used thou as the maximum number of battery slots is reduced during the demand calculation to the maximum demand a location can fulfill $\bar{d}$ and 70 is rarely reached. In the end, these values are averaged for all areas. For the cost of building a battery slot b we choose 40. Each location then gets a value c chosen uniformly from $40 \leq c \leq 70$. For the DMBSSLP we can thus just take s and b values as constants as they already are and set the c value to the mean of all c values in the MBSSLP.

In the next step, the OD-pairs are created. The origin and destination are chosen from a random subset $V' \subseteq V$ with $|V'| = \min(\frac{m}{2}, n)$. This creates demand hubs for smaller instances where the demand is concentrated in fewer nodes. For instances with $m > 2n$ all nodes are eligible and the demand is more evenly distributed. Each area has a weight w_l assigned to it according to a log-normal distribution with $\mu = \log(100)$ and $\sigma = 0.5$.

So far there have been no major changes between the instance generation for the MBSSLP and the DMBSSLP. Where the methods differ is how the demands for the OD-pairs for the two problems are generated. As the MBSSLP operates on different time steps, 24 different demand values, one for each hour of the day are generated. The total demand of an OD-pair (u, v) is calculated according to the weights of the nodes u, v and their distance p_q according to $d_{tot} = w_u \cdot w_v \cdot f_{PDF}(p_q, \mu, \sigma)$ where $f_{PDF}(p_q, \mu, \sigma)$ represents the probability density function (PDF) of a log-normal distribution with $\mu = 5000$ and $\sigma = 0.2$. The PDF is used so that the trips customers take are on average 5km long. Very short trips $<$ 2km result in the PDF being almost 0 as it is unlikely that a customer would need to refuel during such a short trip. Very long trips $>$ 10km are also unlikely as the goal is to represent trips in an urban network. To accommodate that such trips might still be possible we set the demand for OD-pairs where the demand would be 0 to at least 0.01.

With the premise that most traffic is concentrated in the early morning hours when most people drive to work and the late evening hours when people drive home, two separate normal distributions, one for the morning traffic with $\mathcal{N}_{morning}(8,1)$ with the mean at 8 hours and a standard deviation of 1 and one for the evening $\mathcal{N}_{evening}(18,2)$ are created. From each distribution, 100 values t_i are drawn and transformed to integral values according to $t = (\lceil t_i \rceil \mod 24) + 1$. Finally, the demand d_{tot} is distributed over all time steps according to how often a time step appears in the random drawings.

Because we do not consider all 24 hours of the day in the DMBSSLP we simplify the created demand and take the hour of the day which has the highest sum of all demands associated with it. This allows us to still solve the most challenging variant of the problem while simplifying the problem enough.

Next, we compute the pairwise shortest paths between all nodes using the Johnson Shortest Path algorithm [CLRS22]. This algorithm runs in $O(|V|^2 \log |V| + |V||E|)$ which can be simplified to $O(|V|^2 \log |V|)$ considering the graph has $2n - 1$ edges and thus $O(V) = O(E)$.

The graph $P(V,E)$ is now transformed into the bipartite graph $G(Q,L,E)$ that we already know of. We set the maximum detour length w_{detour} to 400m. Every OD-pair q with starting and end areas $u, v \in V$ will now be connected to every area $x \in V$ such that the distance p_{uv} in the graph $P(V,E)$ does not exceed the maximum detour length. Meaning if the inequality $p_{ux} + p_{xv} \leq p_{uv} + w_{detour}$ holds, there will be a connection from $q \in Q$ to $x \in L$ in $G(Q,L,E)$

Second to last we calculate the Budget for the instance as seen in Equation (5.24)

$$B = 0.5 \left(c \cdot \left\lceil \frac{\sum_{q \in Q} d_q}{s} \right\rceil + b \cdot \left\lceil \sum_{q \in Q} d_q \right\rceil \right) \tag{5.24}$$

This ensures that close to 50% of the demand can be fulfilled for every instance. The first fraction gives the number of stations that are needed to fulfill all demand and the second sum gives the number of slots needed. This assumes ideal scenarios where the demand can be distributed easily among those stations and slots.

Finally, the created graph $G(Q,L,E)$ is trimmed meaning areas that have no connection to any OD-pair are removed.

5.5 Data Generation

Our goal is to train two neural networks which can be used to create a partitioning in the MLO. The idea is not to directly predict the similarity of two nodes but to predict the error that is implied when merging two nodes. To train this NN in a classical supervised fashion we first need to create training data.

This training data creation follows the scheme of: Create a training instance; Coarsen the instance; Solve the coarse problem instance as well as the fine and all intermediate problem instances using the MILP formulation (2.3)-(2.10).

The training data is then composed of two parts, the independent variables $\mathbf{X}$, which represent the input features of the NN and are composed of the row vectors $h^i_{L,l,l'}$ and $h^i_{Q,q,q'}$ for the two separate NNs and the dependent variable $\mathbf{z}$ which represents the error we created when merging the two nodes.

The first strategy we tried was to coarsen a single node pair in each iteration. This proved to be far too slow to create a meaningful amount of data points because a MILP needs to be solved for every intermediate result.

Therefore we developed two more meaningful ways to create training data. The first strategy is the so-called subproblem data strategy. In this strategy, we create the dependant data directly when solving the projection of every single subproblem as described in Section 5.2.4. In the second strategy, the k-data strategy, we coarsen k nodes and take the overall error created during the coarsening as dependent variable.

The following sections present both strategies in detail.

5.5.1 Subproblem Data Strategy

To create data according to the subproblem data strategy we first coarsen the instance. We use two different partitioning strategies here.

The first is random partitioning. In this strategy, two random nodes that are neighboring, in the respective two-hop graph, are merged. This strategy proposes the greatest diversity as we do not introduce any bias when choosing nodes to partition.

The second partitioning strategy follows the Jaccard similarity as described in Section 5.3.1. On the one hand, this strategy gives us a lot more data on nodes that might be good to partition but also introduces a bias.

After the coarsening is done the independent data can be collected in terms of the $h^i_{L,l,l'}$ and $h^i_{Q,q,q'}$ vectors of the merged nodes.

First, we explain how the dependent data for two nodes $q_1, q_2 \in \tilde{G}^i$ that have been merged in a node $q_3 \in G^i$ can be obtained. After a solution x^i, y^i, a^i has been found for G^i it is projected to the next finer instance $\tilde{G}^i$ by solving the independent LP (5.6)-(5.9). The dependant data for the merged nodes q_1, q_2 is described as $z_{q_1 q_2}$ and is obtained as the difference between the demand that has been satisfied in the coarse and the fine solution as can be seen in Equation (5.25)

$$z_{q_1 q_2} = \left(\sum_{l \in N^i(q_3)} a^i_{q_3 l} \right) - \left(\sum_{l \in N^i(q_3)} \tilde{a}^i_{q_1 l} + \tilde{a}^i_{q_2 l} \right) \tag{5.25}$$

Notice that $N^i(q_3) = \tilde{N}^i(q_1) \cup \tilde{N}^i(q_2)$ and thus we can just sum over the earlier neighborhood.

With this definition we can obtain the independant variable h^i_{Q,q_1,q_2} and the respective dependant variable $z_{q_1q_2}$ for two merged nodes $q_1, q_2 \in \tilde{Q}^i$. A caveat is that if the merged OD-pairs q_1, q_2 are not part of the found solution we delete the observation. The motive for this is to avoid accidentally giving bad merges 0 error that usually would cause some error if their representative nodes would have played a part in the found solution. To detect if a node pair is part of the solution we check if any demand has been allocated from them. This is done by checking if Inequality (5.26) holds.

$$\left(\sum_{l \in N^i(q_3)} \tilde{a}^i_{q_1 l} + \tilde{a}^i_{q_2 l} \right) > 0 \tag{5.26}$$

To obtain the dependent variable for two nodes $l_1, l_2 \in L^{i-1}$ from the solution $\tilde{x}^i, \tilde{y}^i, \tilde{a}^i$ of a graph $\tilde{G}^i$ we use two different strategies. Recall that solving the subproblems (5.10) - (5.17) when projecting a node $l \in \tilde{L}^i$ the individual subproblems are not independent of each other because of δ being able to change depending on the order of subproblems.

The first strategy we are going to explore just uses the dependant δ as is. We obtain the dependent variable $z_{l_1l_2}$ for two nodes $l_1, l_2 \in L^{i-1}$ and $l_3 \in \tilde{L}^i$ by solving the LP stated by inequalities (5.10) - (5.17). Similar to before we use the difference in fulfilled demands as the dependent variable as described in Equation (5.27)

$$z_{l_1l_2} = \left(\sum_{q \in \tilde{N}^i(l_3)} \tilde{a}^i_{ql_3} \right) - \left(\sum_{q \in \tilde{N}^i(l_3)} a^{i-1}_{ql_2} + a^{i-1}_{ql_3} \right) \tag{5.27}$$

If an area is deemed unimportant for a solution it is not recorded. This is checked by observing the number of slots built in areas l_1, l_2. If $y^{i-1}_{l_1} > 0 \vee y^{i-1}_{l_2} > 0$ the observation is important and is not deleted.

The second strategy to collect the dependant variable $z_{l_1l_2}$ involves making the LP (5.10) - (5.17) that is solved when calculating a projection independent. To adapt the projection Algorithm 5.3 the Lines 2 and 7 are changed to set δ to 0. These are the lines where δ is first initialized and updated after a single subproblem is solved. With δ being set to 0 the subproblems can be solved independently of each other as the order of solving the individual subproblems no longer influences the solution.

Overall this leaves us with four variants of the subproblem data strategy. The variants Jaccard and random in which different partitioning strategies are used to create a coarsening and the dependant and independent δ strategy where the δ variable in algorithm 5.3 is changed to create the data.

5.5.2 k-data Strategy

With the subproblem data strategy we pursue the idea that every individual node is responsible for the error it creates. For a complex network structure, this might not be the case as there might be numerous interdependencies between nodes during coarsening that we cannot capture like this. We, therefore, try to obtain a more complete view of the problem with the k-data strategy. We want to avoid bias to understand the network structure more completely and thus only use random partitioning. Additionally solving for two different partitioning strategies would also double the already large number of models being trained.

In the k-data strategy, an observation is no longer only the features $h^i_{Q,q,q'}$ or $h^i_{L,l,l'}$ for a single merge but an observation is the entirety of all features h^i of all coarsened nodes. This means we can no longer represent our data as a two-dimensional independent feature matrix $\mathbf{X}$ with a dependent variable vector $\mathbf{z}$.

A single observation now is a two-dimensional matrix $\mathbf{A}$. We thus have to upscale the two-dimensional feature matrix $\mathbf{X}$ to a three-dimensional one with the dimensions $k \times |h^i| \times N$ where N is the number of observations and $|h^i|$ is the number of features. To create a single observation $\mathbf{A}$ we coarsen k nodes and record all their features $|h^i|$. To create the dependent variable for a single observation which we denote as z_j we take the difference between the objective value after it was projected and compare it to the optimal solution which we obtain by solving MILP (2.3)-(2.10). The equation for z_j can be seen in Equation (5.28). The values $a^{\mathrm{MLO},i}_{ql}$ and $a^{\mathrm{MILP},i}_{ql}$ describes the a values obtained after solving the MLO and MILP for graph G^i respectively.

$$z_j = \left(\sum_{q \in Q^i} \sum_{l \in L^i} a^{\mathrm{MILP},i}_{ql} \right) - \left(\sum_{q \in Q^i} \sum_{l \in L^i} a^{\mathrm{MLO},i}_{ql} \right) \tag{5.28}$$

Unlike for the subproblem strategy, data is not deleted immediately in the k-data strategy if it does not play a role in the solution. Instead, the data is flagged as being relevant or not. In the final training described in Section 6.4 we will distinguish between those strategies as using only relevant data and using all data.

5.5.3 Training Data Creation

As described in the section above we use four different data creation strategies to create data with the subproblem data strategy. We create samples from two different instance sizes. One dataset with constant instance size $m, n = 500$ and another dataset with varying instance size with m, n uniformly at random chosen in the range 100 to 1000.

We create datasets using different instance sizes, different partitioning strategies, and different delta strategies. Additionally, the data is also split between data for the Q and L nodes. An overview of the created data can be seen in Table 5.1. We create about 10000 instances each to collect data from. This results in about 2.5 to 3 million data

Table 5.1: Subproblem data strategy training data overview

size	similarity	δ	nodes	instances	datapoints	memory in MB
500	Jaccard	independant	Q	10000	2720000	538
500	Jaccard	independant	L	10000	388000	79
500	Jaccard	dependant	Q	9900	2540000	502
500	Jaccard	dependant	L	9900	361000	73
500	Random	independant	Q	9700	2380000	465
500	Random	independant	L	9700	398000	81
500	Random	dependant	Q	10000	2510000	490
500	Random	dependant	L	10000	381000	78
100-1000	Jaccard	independant	Q	10000	3050000	598
100-1000	Jaccard	independant	L	10000	377000	75
100-1000	Jaccard	dependant	Q	9900	2960000	580
100-1000	Jaccard	dependant	L	9900	346000	69
100-1000	Random	independant	Q	9600	2662000	519
100-1000	Random	independant	L	9600	379000	76
100-1000	Random	dependant	Q	10000	2868000	559
100-1000	Random	dependant	L	10000	367000	73

points for Q and about 350 to 400 thousand data points for the L data. The L data is much more sparse because we delete data points that do not play a role in the solution. A datapoint in L is deleted if there are no slots y built at that area while a datapoint in Q is only deleted if all neighboring a_{ql} are 0. Making it much more likely that a data point in L will be deleted.

For the k-data strategy, we create data using different values, for k. A very small value $k = 5$, a medium value $k = 30$, and a large value $k = \max$. $k = \max$ means the whole instance is partitioned meaning every node that can be paired will be paired if there is a suitable partner left. For smaller k values the created data represents the individual nodes a lot better. Larger k values contract the whole problem instance and thus are much closer to how the MLO will be eventually applied. For each strategy, we create a sizeable dataset using the different training data sizes as can be seen in Table 5.2. The reason we have so much fewer data points than with the subproblem strategy although the data takes up a lot more memory is because a single data point now contains multiple coarsened nodes.

Table 5.2: k data strategy training data overview

size	k	nodes	instances	datapoints	memory in MB
500	5	Q	7050	418000	523
500	5	L	7050	418000	502
500	30	Q	5000	65000	405
500	30	L	5000	65000	383
500	max	Q	7500	22500	646
500	max	L	7500	22500	644
100-1000	5	Q	6000	297000	370
100-1000	5	L	6000	297000	353
100-1000	30	Q	5000	51500	318
100-1000	30	L	5000	51500	300
100-1000	max	Q	7500	22500	715
100-1000	max	L	7500	22500	677

CHAPTER 6

Experiments and Results

In this chapter, we describe our experiments and the results we obtained. We first start by giving a short description of the computing environment and continue with a data analysis of the training data. Then the benchmark instances we used for testing are described. In the following section, we explain the different model parameters we evaluated how they compare to each other, and how they compare to the conventional Jaccard similarity strategy. In the final section, we analyze the running times of the more complex ML strategy.

6.1 Computing Environment

The computation cluster that was used for the experiments consists of two main components. A fast one with a lot of memory and a slower one with a medium amount of memory. The fast compute node uses an AMD EPYC 7402, 2.80GHz 24-core with up to 1024GB of RAM. This node was used for memory-intensive tasks such as large instance creation. As this is the fastest node available on the server it receives a lot more traffic than the slower node. Therefore we used the slower node with an Intel Xeon E5-2640 v4, 2.40GHz 10-core, and 160GB of RAM for time-sensitive experiments to obtain consistent results. Still, traffic might have influenced some results.

6.2 Training Data Analysis

We continue with a data analysis to explore how the different features $h^i_{L,l,l'}$ and $h^i_{Q,q,q'}$ are distributed and correlate with our target feature. We make comparisons between the different data collection strategies as well as grouping the features by the MLO level to see how and if they change during the coarsening.

In Tables 6.1 and 6.2 we see the correlations of the most significant features for the L and Q set respectively. The most significant features seem to be the symmetric difference of

Table 6.1: Features of L with the strongest correlation to the objective

	Δd_q	$\bar{d}_{l3}$
dependent 100-1000 jaccard	-0.354	-0.39
dependent 100-1000 random	-0.281	-0.3377
dependent 500 jaccard	-0.128	-0.0569
dependent 500 random	0.00258	0.00283
independent 100-1000 Jaccard	0.323	0.327
independent 100-1000 random	0.446	0.385
independent 500 Jaccard	0.086	0.227
independent 500 random	0.325	0.336

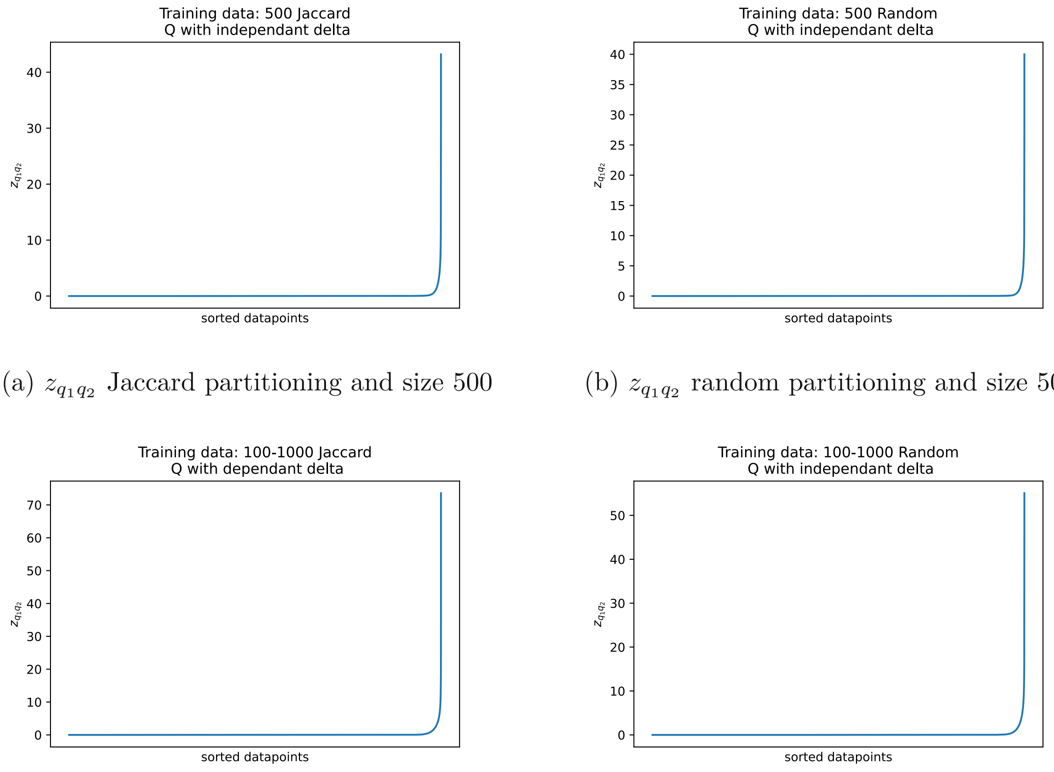
Table 6.2: Features of Q with the strongest correlation to the objective

	d_{q_1}	d_{q_2}	d_{q_3}
dependent 100-1000 jaccard	0.229	0.23	0.298
dependent 100-1000 random	0.223	0.222	0.289
dependent 500 jaccard	0.192	0.187	0.263
dependent 500 random	0.182	0.176	0.248
independent 100-1000 Jaccard	0.229	0.229	0.298
independent 100-1000 random	0.223	0.222	0.289
independent 500 Jaccard	0.186	0.181	0.256
independent 500 random	0.177	0.172	0.243

the neighboring d_q values of the merged l nodes and the future $\bar{d}_l$ value when the nodes are merged. For the Q set the former and future demands d_q of the nodes are the most significant.

We inspect how the target values $z_{qq'}$, $z_{ll'}$, and z_j are distributed referencing the different data collection strategies. In Figure 6.1 we see the distribution of the dependent variable $z_{q_1q_2}$ for the Q set with different parameters for partitioning and size. The distributions are very similar. Almost all node merges cause no change in the target variable but the few that do cause a significant reduction in the target variable. The different merging strategies do not have a significant impact on the distributions of $z_{qq'}$. The different strategies for the δ variable do not influence the shown data of the Q set as δ only influences the L projection.

Considering the L set we explore the different strategies for the δ variable and partitioning strategy in Figure 6.2. First, we compare the dependent δ to the independent δ strategy in Figures 6.2a and 6.2b. The dependent δ strategy allows a single subproblem to improve the objective value we therefore can obtain negative values of $z_{l_1l_2}$. In contrast when setting δ independent as described in section 5.5.1 we can not obtain negative values. When comparing the partitioning strategies as seen in Figures 6.2b and 6.2d the differences are only minimal. Random partitioning can cause slightly higher errors but the overall difference is marginal. The Figures 6.2 contain the data for the training



(a) $z_{q_1q_2}$ Jaccard partitioning and size 500 (b) $z_{q_1q_2}$ random partitioning and size 500

(c) $z_{q_1q_2}$ Jaccard partitioning and size 100-1000 (d) $z_{q_1q_2}$ random partitioning and size 100-1000

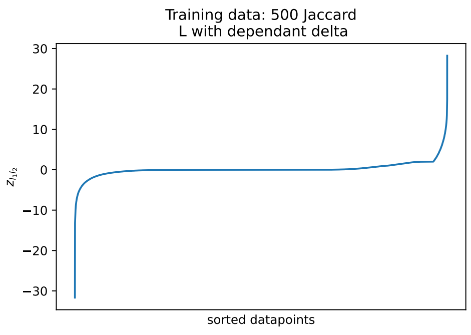
Figure 6.1: dependent variable $z_{q_1q_2}$ for different subproblem strategies

data of size 500. For problem instances of varying sizes in the range of 100-1000, the maximum error increases but the relative distribution remains the same.

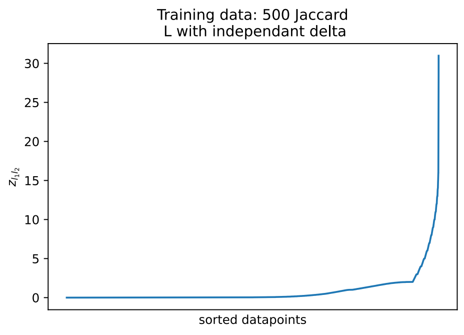
An interesting observation is that there seems to be a break in continuity to the right of the distribution where $z_{l_1l_2}$ shortly plateaus and afterward the values sharply increase. A plot with a zoomed-in version on this part can be seen in Figure 6.3. We see a plateau at integer values e.g. 1, 2, 3, or 4. This is likely because the number of slots y_l at a station are integer. We cannot explain why the gradient sharply increases after the plateau at 2.

As a final bit of analysis, we take a look at the estimated probability density functions (PDF) of the features with the most important correlations. The PDF for $\bar{d}_{l_3}$ and Δd_q can be seen in Figure 6.4. Both functions look left skewed with a majority of the values being low.

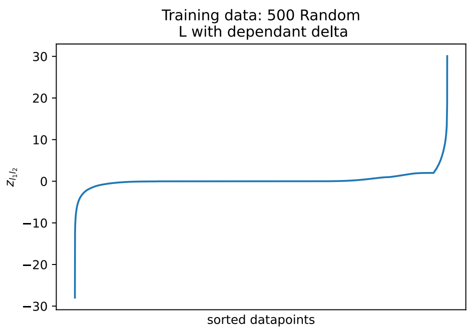
When examining the distributions split up over 3 coarsening levels we see distributions as follows in Figure 6.5 and 6.6. Especially interesting are the ripples in 6.5a these are caused by the instance generation making certain demand values more likely to appear than others. Over the coarsening the PDF of both distributions smooth out and get wider. When nodes are being merged their properties often are added up which creates



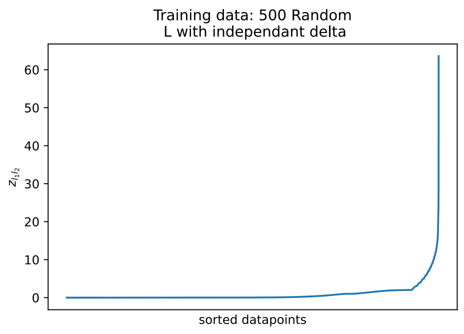
(a) $z_{q_1 q_2}$ Jaccard partitioning and dependent δ



(b) $z_{q_1 q_2}$ Jaccard partitioning and independent δ



(c) $z_{q_1 q_2}$ random partitioning and dependent δ



(d) $z_{q_1 q_2}$ random partitioning and independent δ

Figure 6.2: dependent variable $z_{q_1 q_2}$ for different subproblem strategies

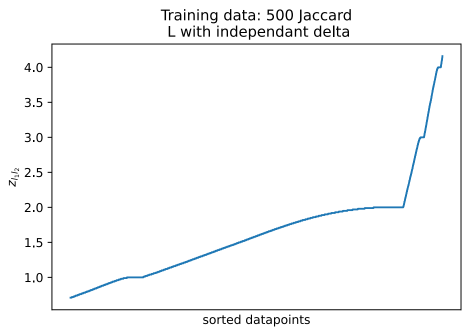
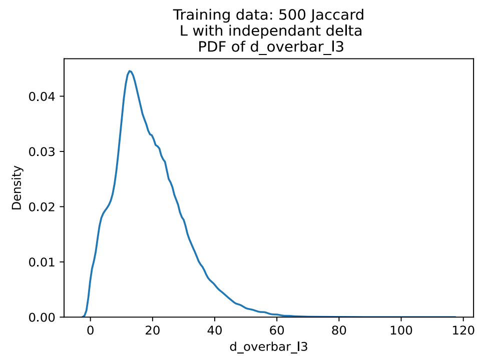
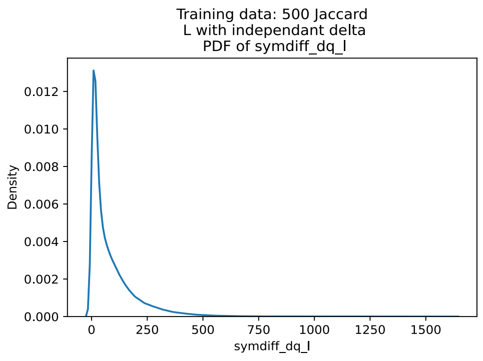


Figure 6.3: Zoom for $z_{q_1 q_2}$ jaccard partitioning and independent δ



(a) PDF of $\bar{d}_{l_3}$



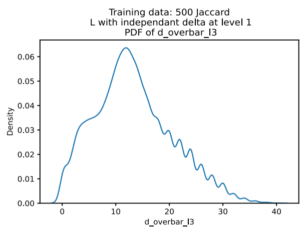
(b) PDF of Δd_q

Figure 6.4: PDF of most correlated features

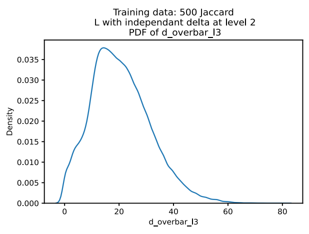
the increase as well as the smoothing effect.

The PDF of d_{q_1}, d_{q_2} and d_{q_3} which can be seen in Figure 6.7 are not that meaningful. Most OD-pairs have very low demand in the range of $0.01 - 0.1$ with few containing high demands of up to 30 expected users. The PDFs are therefore very skewed to the left. After merging, the PDF for d_{q_3}, which represents the prospective merged node, gets a little wider but overall not a huge difference can be seen. The same follows when comparing the feature over the different levels as seen in Figure 6.8. Notice that 6.8g look the same as 6.8b and 6.8e and 6.8h look the same as 6.8c and 6.8f because the d_{q_1} and d_{q_2} merged to d_{q_3} represent the merged node. Therefore the distribution of future primary nodes d_{q_1} and d_{q_2} at higher levels are the same.

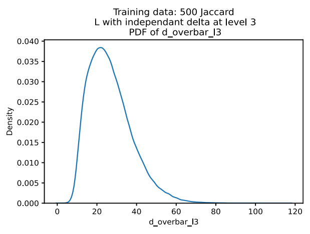
Because datapoints of the k-data strategy consist of multiple observations of $h_{L,l,l'}$ or $h_{Q,q,q'}$ it is not entirely obvious how one would plot a single datapoint for this strategy. We refer to the analysis above for the subproblem data strategy as the way the features are calculated is the same as for the random partitioning case. The only real difference is in the calculation of the dependent variable z_j. A short analysis for different k values for the problem instances of size 500, for the Q and L nodes, can be seen in Figures 6.9 and 6.10 respectively. High k values cause the curves to be very smooth but also comparatively steep to the earlier subproblem strategy plots in Figure 6.1 and 6.2. For the L set the curve again shows the bumps we already have seen during the investigation of the subproblem strategy. Especially for low k values the curve gets very bumpy. This suggests that single nodes are responsible for the majority of the error and if it just so happens that these nodes are being merged the error increases dramatically. In Figure 6.11 we show a zoomed version of the bumps in Figure 6.10b. Again this plateaus at integer values and behaves relatively continuously in between.



(a) PDF of $\bar{d}_{l_3}$ for level 1

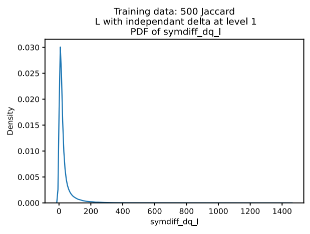


(b) PDF of $\bar{d}_{l_3}$ for level 2

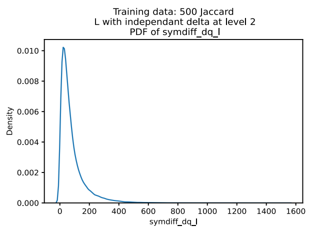


(c) PDF of $\bar{d}_{l_3}$ for level 3

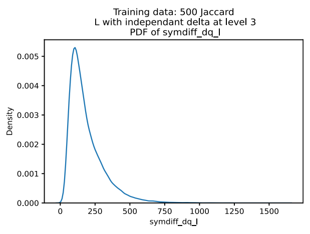
Figure 6.5: PDF of $\bar{d}_{l_3}$ for different levels



(a) PDF of Δd_q level 1

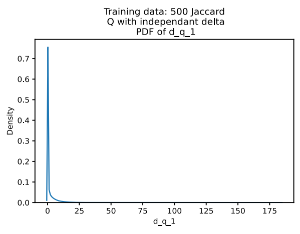


(b) PDF of Δd_q level 2

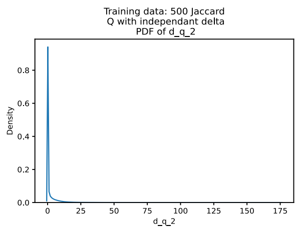


(c) PDF of Δd_q level 3

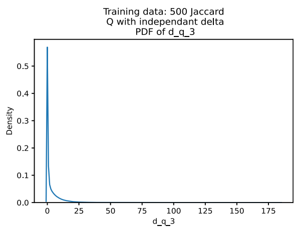
Figure 6.6: PDF of Δd_q for different levels



(a) PDF of d_{q_1}



(b) PDF of d_{q_2}



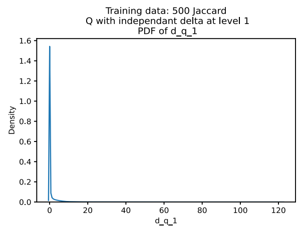
(c) PDF of d_{q_3}

Figure 6.7: PDF of d_q

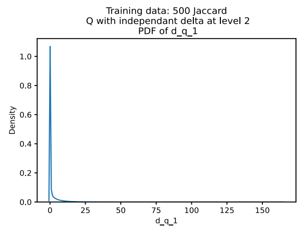
6.3 Benchmark instances

To thoroughly test the models we trained we define a set of independent benchmark instances. These instances represent instances of similar size to the training instances as well as larger instances to understand how well the trained models generalize to instances of larger sizes.

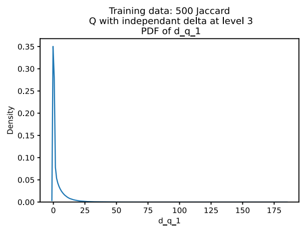
We created benchmark instances with four size ranges. The first and smallest set contains 20 instances of size $n = 500$ and $m = 500$. The second set contains 20 instances with n, m chosen uniformly at random from the range 100 to 1000. The second to last set contains 18 instances with $n, m \in \{2000, 4000, 8000\}$ with 2 instances for each possible combination of n and m. The final set contains two very large instances with $n, m = 10000$.



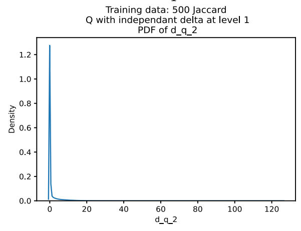
(a) PDF of d_{q_1} at level 1



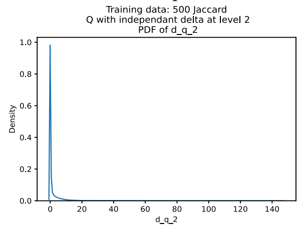
(b) PDF of d_{q_1} at level 2



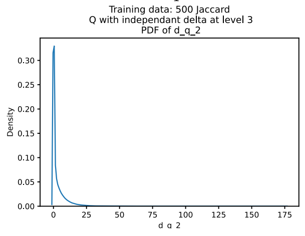
(c) PDF of d_{q_1} at level 3



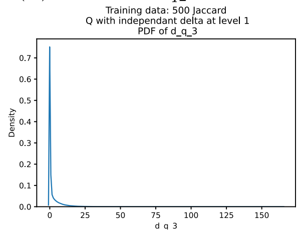
(d) PDF of d_{q_2} at level 1



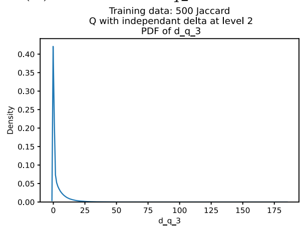
(e) PDF of d_{q_2} at level 2



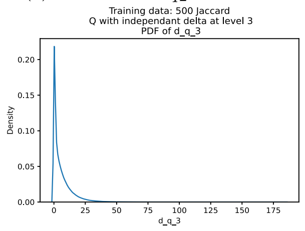
(f) PDF of d_{q_2} at level 3



(g) PDF of d_{q_3} at level 1

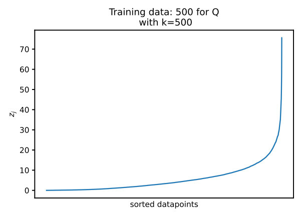


(h) PDF of d_{q_3} at level 2

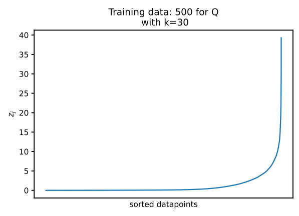


(i) PDF of d_{q_3} at level 3

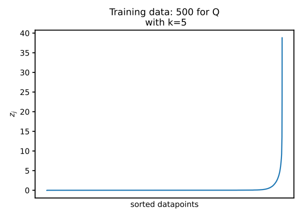
Figure 6.8: PDF of d_q at different levels



(a) z_j for $k = 500$

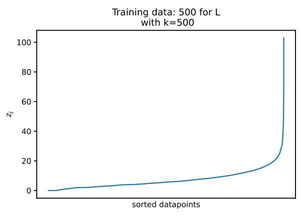


(b) z_j for $k = 30$

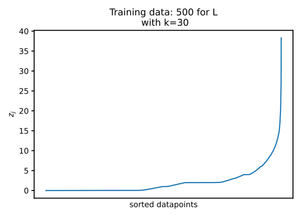


(c) z_j for $k = 5$

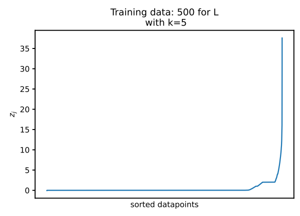
Figure 6.9: z_j value for Q with different values for k



(a) z_j for $k = 500$



(b) z_j for $k = 30$



(c) z_j for $k = 5$

Figure 6.10: z_j value for L with different values for k

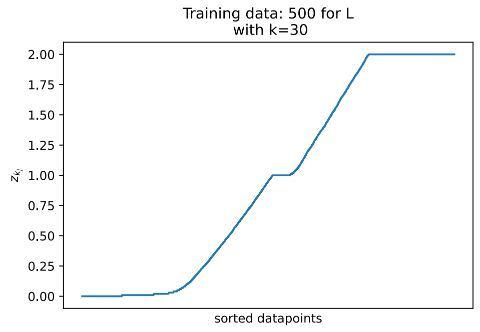


Figure 6.11: Zoom for z_j for $k = 30$

6.4 Model Training

Finding the optimal settings for models in terms of hyperparameters, architecture, and training data is often a difficult task. We, therefore, conducted a series of tests to find good values for all parameters. We filtered the results for the best parameters to compare only the models which produced the best results.

6.4.1 Model Architecture

The first thing we experimented with was the architecture of the models. The input size is defined by the size of $h_{L,l,l'}$ and $h_{Q,q,q'}$ and the output is just a single node with no activation function as we want a numeric value for the similarity. As an activation function for our hidden layers, we choose ReLu (Rectified Linear Unit). For the remaining analysis, we describe the architecture of a model within square brackets with the individual number of nodes being separated by a comma in the fashion [1^{st} layer, 2^{nd} layer].

We experimented with deep networks with up to 10 layers and also wide but shallow networks with up to 640 nodes in a single layer. We mostly focused our analysis on shallow networks with only two dense layers. A comparison between these models according to the number of trainable weights is shown in Figure 6.12. These results were obtained with models trained on the dataset of size 100 to 1000 obtained using the subproblem-data strategy. The results are shown for the test data of the same size. We see that increasing the number of trainable weights in a network does not necessarily improve the results.

In the next step, we considered regularization for which we used dropout. A dropout layer with 30% dropout was added after the first hidden layer. In Figure 6.13 we see a plot comparing the results of the dropout models to the models without. On the x-axis, we see the average fulfilled demand for the model without dropout while on the y-axis we see the fulfilled demand for the same model with the added dropout layer. Points below the line support the hypothesis that models without dropout perform better. Points above the line support the inverse hypothesis that dropout models are better. The mean of all dropout models is 47.7% while the mean for all models without dropout is 48.4%.

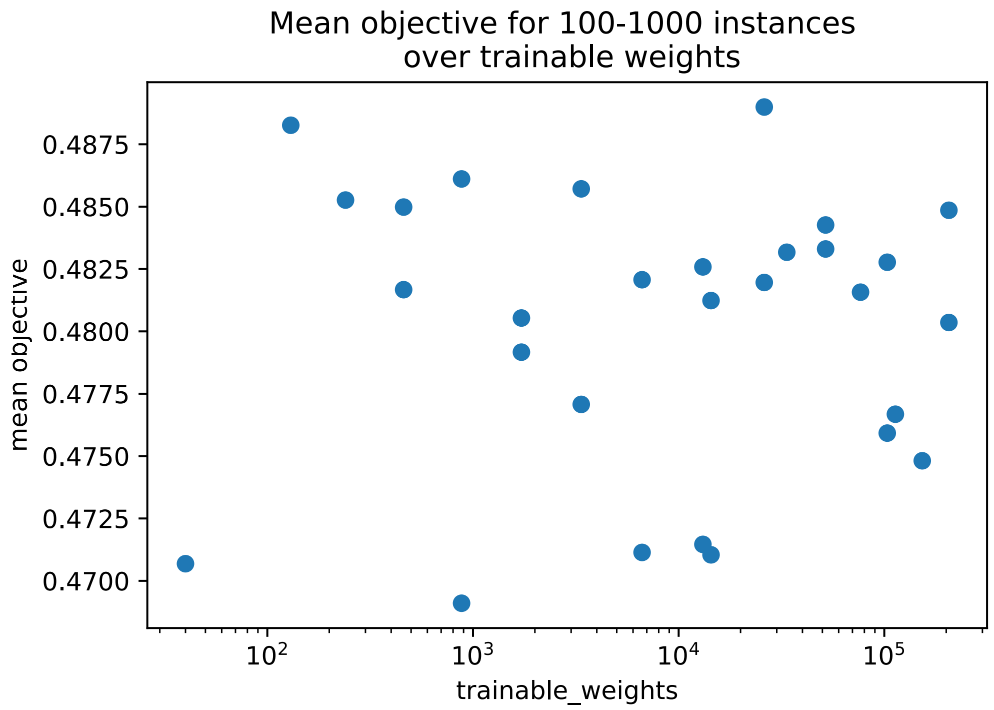


Figure 6.12: Average fulfilled demand over number of trainable weights

With a Wilcoxon Ranked Sum Test (WRST) [Wil92] we confirmed that the distributions are significantly different ($p = 10^{-5}$).

We also compared the results for individual models as it could be that the overall number of models performs worse when using dropout but individual models might perform better. In Figure 6.14 we see the comparison of a select number of models with and without dropout. The two best-performing models use the architecture [40,20] and [160,Dropout,160]. When comparing the two models in figure 6.14b we see no significant advantage for either model which is confirmed with a WRST with a p-value of 0.7. On the other hand, it seems like bigger models like 6.14d do not automatically perform better when dropout is added but smaller models perform worse with dropout. We also found no advantage when training deeper models with more layers. We, therefore, continue the analysis only with smaller models as they perform equally as well or better than larger models and offer an advantage in terms of training time and practicability. The structures we continue to evaluate are [40,20], [40,40], and [80,40]. Smaller models than [40,20] also drop in performance very quickly and are thus discarded as well.

After we conducted the preliminary tests we tested the three chosen architectures with the test instances described in Section 6.3. We aggregated the results for all benchmark instances and compared the two different training data creation strategies. The results can be seen in Table 6.3.

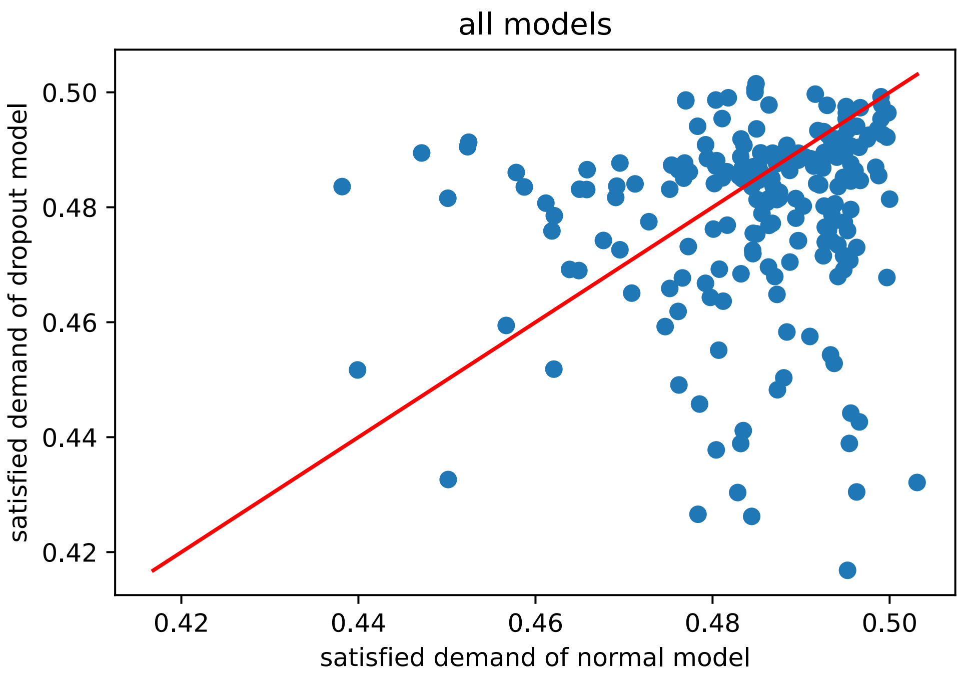


Figure 6.13: Dropout comparison all models

Table 6.3: Mean objective for different architectures

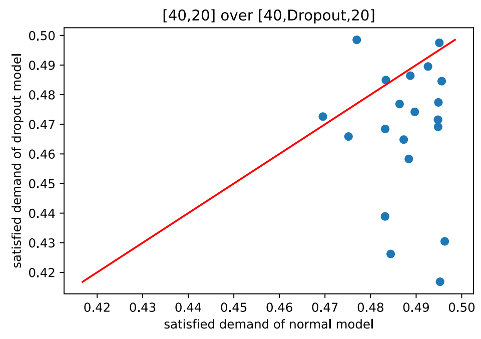
architecture	subproblem data strategy	k-data strategy
$[40, 20]$	47.6%	47.3%
$[40, 40]$	47.7%	46.9%
$[80, 40]$	47.6%	46.9%

We find that for the subproblem strategy the models of size [40,40] and for the k-data strategy the [40,20] model outperform the other models. This is confirmed by WRST with $p = 10^{-2}$ and $p = 10^{-6}$ respectively to the next best model.

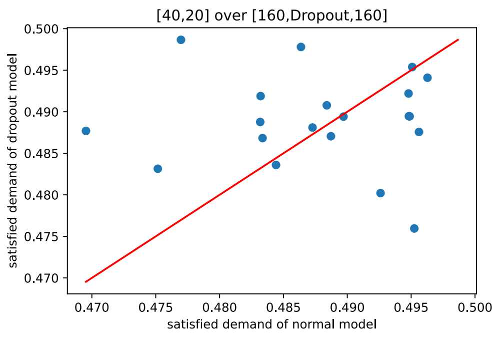
6.4.2 Learning Rate and Batch Size

Two hyperparameters that influence the learning curve dramatically are the batch size and the learning rate. We tested two different batch sizes 64 and 256 as well as two learning rates 10^{-3} and 10^{-4} against each other and compared the learning curve as well as the performance of the final models.

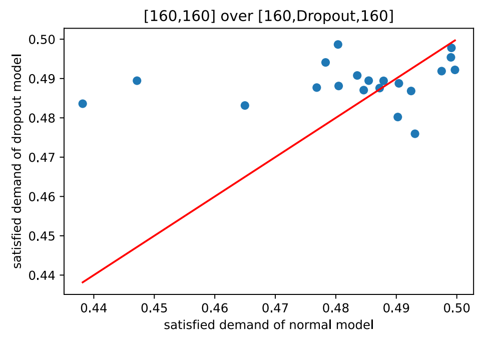
We compare the result of all trained models on all test sets. We only split the analysis between the models trained on the subproblem data strategy and the k-data strategy. The average performance for each batch size can be seen in Table 6.4. Although the results are not much better for either batch size. A higher batch size ensures a smoother



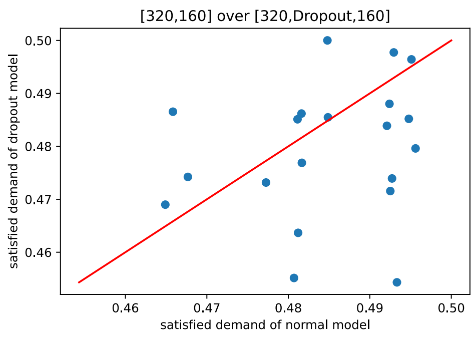
(a) [40,Dropout,20] over [40,20]



(b) [40,20] over [160,Dropout,160]



(c) [160,Dropout,160] over [160,160]



(d) [320,Dropout,160] over [320,160]

Figure 6.14: Comparison of different architectures with and without dropout

Table 6.4: Mean objective for different batch size

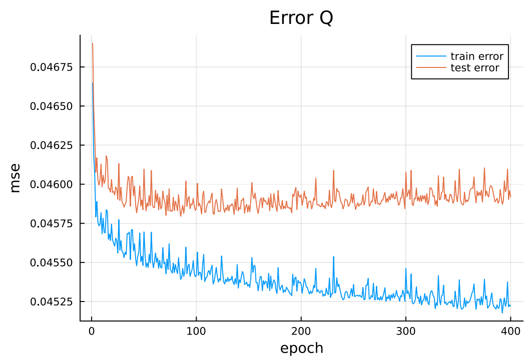
batch size	subproblem data strategy	k-data strategy
64	47.6%	46.9%
256	47.7%	47%

Table 6.5: Mean objective for different learning rates

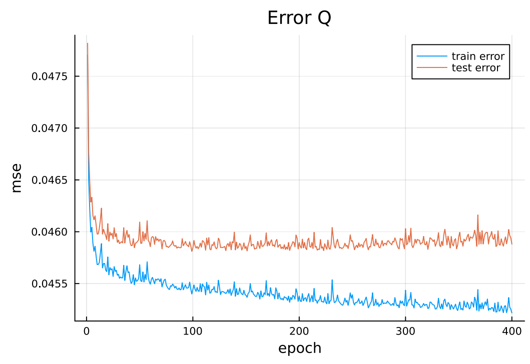
learning rate	subproblem data strategy	k-data strategy
10^{-3}	47.7%	46.7%
10^{-4}	47.6%	47.1%

learning curve as can be seen in Figures 6.15 and was thus our preferred strategy. The plots were created for the models of size [80,40] using a low learning rate and trained on the subproblem data with independent δ.

We now inspect the results for different learning rates. We experimented with two levels of learning rate a high and a low one with the values 10^{-3} and 10^{-4} respectively. In Table 6.5 we see the results for different learning rates split for the different data collection

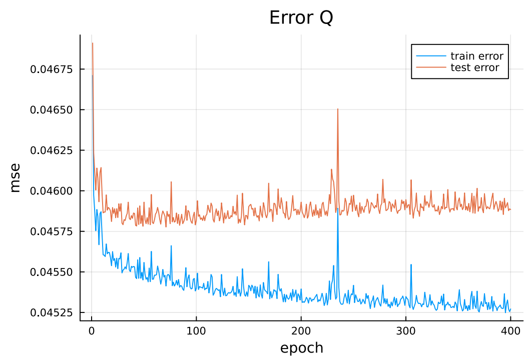


(a) Learning curve for [80,40] model with batch size 64

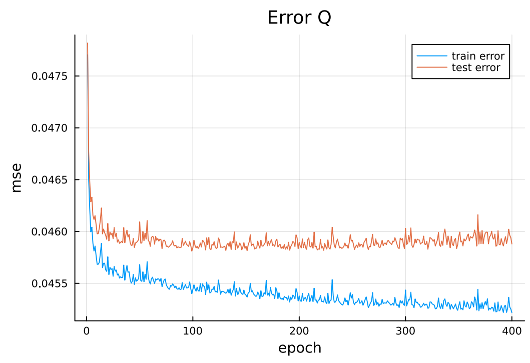


(b) Learning curve for [80,40] model with batch size 256

Figure 6.15: Learning curves for different batch sizes for the model [80,40]



(a) Learning curve for [80,40] model with learning rate 10^{-3}



(b) Learning curve for [80,40] model with learning rate 10^{-4}

Figure 6.16: Learning curves for different learning rates for the model [80,40]

strategies. Although a lower learning rate performs slightly worse for the subproblem data strategy this difference is not significant $p = 0.6$. The lower learning rate does improve the results for the k-data strategy significantly $p = 4 \cdot 10^{-5}$. Lowering the learning rate also has a smoothing effect as seen in Figure 6.16 similar to the higher batch size. Lowering the learning rate can prevent the creation of large spikes as we see in Figure 6.16a around epoch 250.

As a final remark here we state that we probably could have turned the learning rate even lower to smooth out the spikes that remain in the learning curve.

6.4.3 Training Data Size

Models were trained using two different instance sizes. For the first training data set the instances were created with $m = 500$ and $n = 500$. For the other training data set the individual values m and n were chosen uniformly at random from the range 100 to 1000

Table 6.6: Mean objective for different training data sizes

Training data size	subproblem data strategy	k-data strategy
500/500	47.6%	47.5%
100-1000/100-1000	47.7%	46.8%

Table 6.7: Mean objective for different partitioning strategies

partitioning	subproblem data strategy
Random	47.9%
Jaccard	47.4%

for each instance. We split all trained models according to the size of their training data and compared the results on the benchmark instances in Table 6.6.

When applying the models to the benchmark instances their generalizability is tested because the benchmark instances are of different size than the training data. For the subproblem data strategy, this does not seem to be a problem to generalize from the smaller training data to the bigger benchmark data although the training data set with varying sizes outperforms the training data with constant size ($p = 0.01$) slightly.

For the k-data strategy, the difference is more significant. A very likely cause of this is that with varying instance sizes in the training data, the number of OD-pairs and areas may not be equal. When exactly k nodes are merged this increases the relative imbalance between both sets further possibly leading to bad training data. A fix for that would be to draw a random number from the training set and set both OD-pairs and the number of areas equal to that number.

6.4.4 Specifics for Subproblem Data Strategy

In this section, we compare parameters that are specific to the subproblem data strategy.

Partitioning

Here we compare the models created on training data using the random partitioning strategy and the Jaccard partitioning strategy during data creation. In Table 6.7 we see the results for the different partitioning strategies. As we always use random partitioning for the k-data strategy it is not shown here.

Random partitioning significantly ($p = 10^{-30}$) outperforms the biased Jaccard partitioning.

dependent and Independent δ

Here we compare the strategies that use the independent δ versus the strategy using dependent δ. In Table 6.8 we see the average performance on the benchmark instances for the two strategies.

Table 6.8: Mean objective for different δ strategies

δ	subproblem data strategy
dependent	47.7%
independent	47.6%

Table 6.9: Mean objective for different k values

k-values	k-data strategy
$k = 5$	47.7%
$k = 30$	47.5%
$k = \max$	47.4%

The results are very close and no significant difference is found using WRST ($p = 0.2$).

6.4.5 Specifics for k-data Strategy

In this section, we compare parameters that are specific to the k-data strategy.

The Values of k

The primary tuning factor for this strategy is the value of k. We tested three different values a small one $k = 5$, a medium one $k = 30$, and a large value $k = \max$. The instances during training had the sizes of $m, n = 500$ or $m, n \in 100 \dots 1000$. When using $k = \max$ all nodes in an instance are coarsened if suitable partitioning partners exist. The results for the different k values on the benchmark instances can be seen in Table 6.9.

These results have to be taken with a grain of salt because we could not obtain results for some benchmark instances when using $k = 5$ or $k = 30$. The results in Table 6.9 are only for the instances where all strategies were able to produce results. These results therefore can not be used to compare to other strategies yet. The reason why lower k values can not obtain all results is because of the increased size of the feature table. A feature of $h_{L,l,l'}$ and $h_{Q,q,q'}$ is the level i. During training the lower k values produce many levels and thus when the values later were one hot encoded created a larger feature matrix than other strategies. This resulted in memory errors during the application of these strategies although a larger amount of memory (100GB of RAM) was allocated for these experiments. To eliminate the problem I would suggest to either only use large k values or improve the memory management by using sparse matrices for this part.

All data vs. only relevant

During the k-data creation, all observations are recorded even if they are not part of the solution. This is done to have a consistent size for the features in matrix $\mathbf{A}$. Still, it is tracked for each individual observation if that coarsening was part of the solution so we can split the data into use only relevant or use all. When using only the relevant

Table 6.10: Mean objective for different relevant features

data relevance	k-data strategy
all data	46.7%
only relevant	47.5%

observation, the **A** matrix is pruned to only contain the relevant rows. The comparison of the results using models trained on all observations versus only the relevant observations can be seen in Table 6.10.

The results show that using only the relevant data gives a significant increase in performance ($p = 10^{-33}$).

6.4.6 Summary of Parameters

The parameters that were best for both strategies are a low learning rate and a high batch size. This decreases the learning speed and thus over many epochs allows the model to get closer to the optimum.

For the subproblem data strategy, the optimal model architecture seems to contain 40 hidden nodes in both layers. It uses dependent δ and random partitioning.

For the k-data strategy the optimal configuration of parameters uses a smaller architecture of [40,20], the value of k should be set as small and only relevant nodes should be considered during training. The fact that lower k values seem to produce better results suggests that the subproblem strategy is superior to the k-data strategy.

6.4.7 Model similarity vs. Jaccard similarity

As a final part of our model evaluation, we are going to evaluate the models in contrast to the previously defined Jaccard similarity (see Section 5.3.1) on our benchmark instances.

Comparison to Jaccard Similarity

To compare the overall applicability of ML in this setting we compare the results of all models to the Jaccard similarity and thus show that using ML for the similarity calculation dominates the use of the Jaccard similarity.

At first, we take a look at the PDF induced by the results of all models on all benchmark instances and compare that to the result of the Jaccard similarity. The Figure depicting this can be seen in 6.17. Although there is some overlap we see that using pretty much any ML strategy outperforms the Jaccard similarity. The overlap stems from the great variance of the Jaccard similarity. We will later see this explored further in Figure 6.19.

A clearer picture is drawn when filtering for the individual benchmark instance sets. We see in Figure 6.18a and 6.18b that the models perform best on the benchmark instances of the same size as their training data. There they very clearly dominate the Jaccard

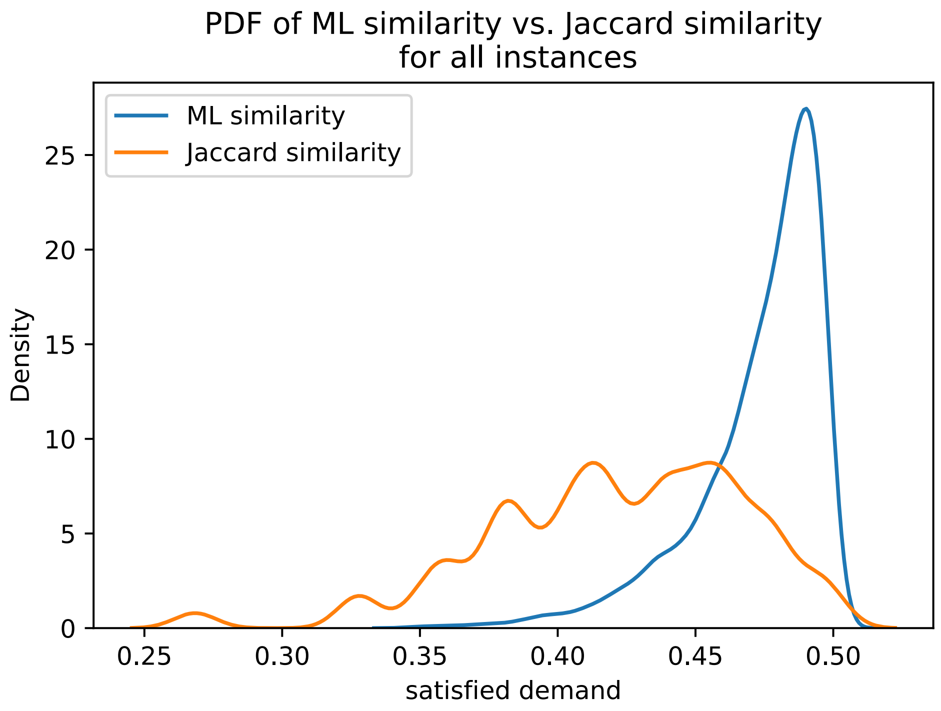
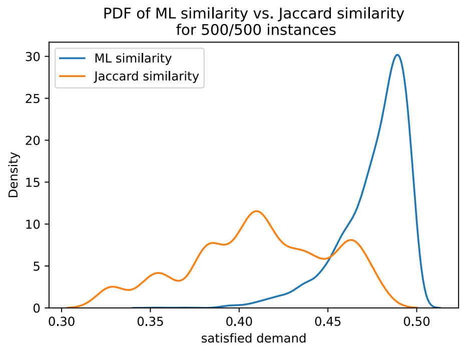


Figure 6.17: PDF comparison of all models vs. Jaccard similarity

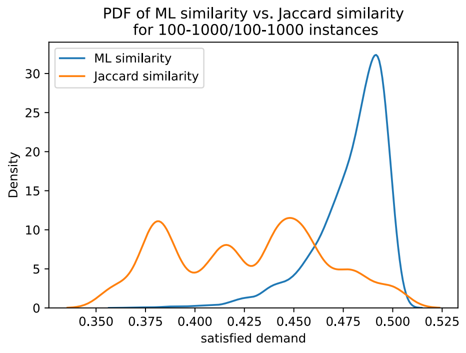
similarity which has an average performance far to the left. When the instances get larger and reach the size of 2k-10k 6.18c, 6.18d the PDF representing the model results gets wider suggesting that some models are better suited for generalization than others. Overall the performance still clearly dominates the Jaccard similarity. For the large 10k/10k instances we see a bimodal distribution for the Jaccard similarity this is because there are only 2 entries in this dataset.

As a final piece of analysis in this chapter we evaluate the best-performing models of the subproblem strategy and the k-data strategy in detail on all benchmark instances. The best-performing model for the subproblem strategy was trained on instances of size 500/500 and uses random partitioning, independent δ, a batch size of 64, a high learning rate of 10^3 and an architecture of [40,20]. The best-performing model for the k-data strategy was trained on instances of size 500/500, using all features not just relevant ones, had $k = 30$, a batch size of 64, a high learning rate of 10^{-3} and an architecture of [40,20].

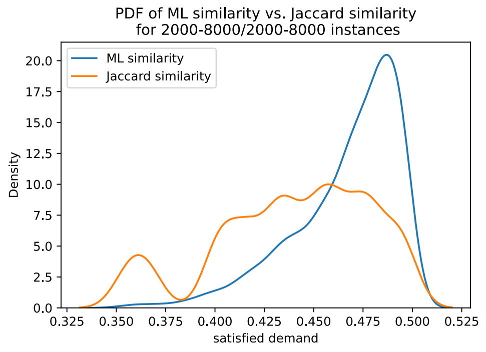
These results were a bit surprising to us as a lot of settings we deemed worse resulted in the best models. For example, using a low batch size and high learning rate usually results in a more chaotic learning process. An interpretation of why this still performs better is that this increases the variance during the learning process and with the sheer amount of different models we trained it was likely that a model relying on high variance would perform well on a limited benchmark set.



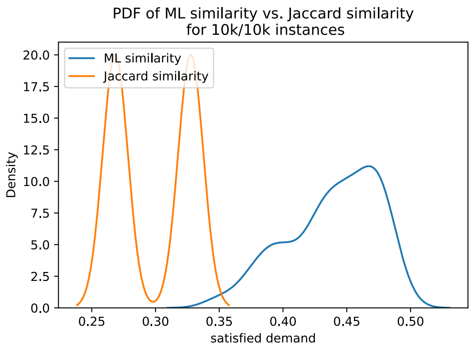
(a) PDF comparison for the 500/500 instances



(b) PDF comparison for the 100-1000/100-1000 instances



(c) PDF comparison for the 2000-8000/2000-8000 instances



(d) PDF comparison for the 10k/10k instances

PDF comparison of all models vs. Jaccard similarity

Nonetheless, we evaluate the results of these models on all benchmark instances. In Figure 6.19 we see boxplots of the results of the models on the individual benchmark sets. A clear difference is that the results produced by the Jaccard similarity have a lot higher variance while the ML similarity produces a lot more consistent results. We also see the subproblem strategy outperform the k-data strategy in terms of consistency most of the time.

For a detailed look into the results of different models on the individual benchmark instances, we refer to Tables 6.12 and 6.13. The best results for a benchmark data set are marked in bold numbers. Results that are indicated with a * only contain partially obtained values. For these values, the set limit of 100GB was exceeded and not all instances were able to be solved.

6.4.8 Time Analysis

In this section, we take a look at the running time of the ML similarity strategies and compare them to the runtime of the Jaccard similarity.

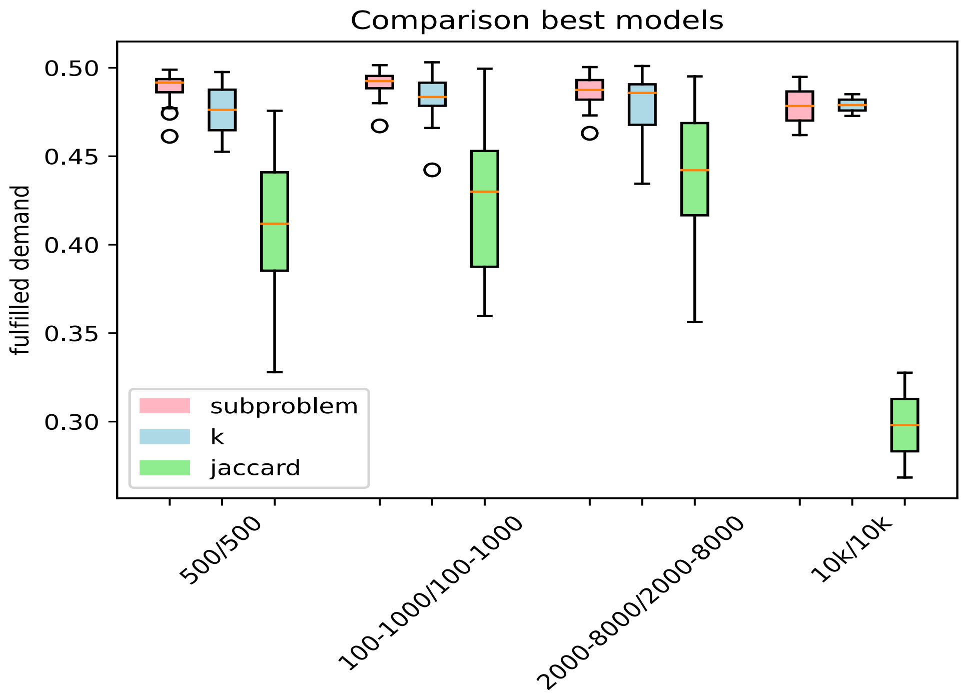


Figure 6.19: Boxplot comparison of the best models to Jaccard

In Figure 6.20 we see a comparison of the runtimes for the Jaccard similarity and the ML similarity. At first glance, no real differences can be established but when filtering the plots for the different benchmark sets the differences become apparent. In Figure 6.21 we see the PDFs split for each benchmark set. A very interesting observation is the bimodal distribution that can be seen in Figure 6.21a. This likely stems from the underlying MILP that is performed when solving the coarsest instance. The difference between the two hills is about 60 seconds which is the same time we had as a time limit for solving the MILP. This means that small instances like the 500/500 instances, are coarsened to such a small level that the MILP can find the optimal solution nearly instantly sometimes. Another interesting observation is that the instances for which the ML similarity produced fast solutions also produced fast solutions using the Jaccard similarity. The other way around was not always the case meaning that if the Jaccard similarity produced a quickly solvable problem instance the ML similarity not necessarily did. Overall our interpretation here is that some problem instances are a lot easier to solve than others and using the Jaccard similarity it is more likely that such instances are produced after the coarsening. In contrast, the ML similarity seems to not create such easy to solve problem instances as often we thus assume that defining features that make a problem hard to solve are more likely preserved through the coarsening levels when applying the ML similarity.

For the other distributions 6.21b 6.21c and 6.21d we see a steady move to the right.

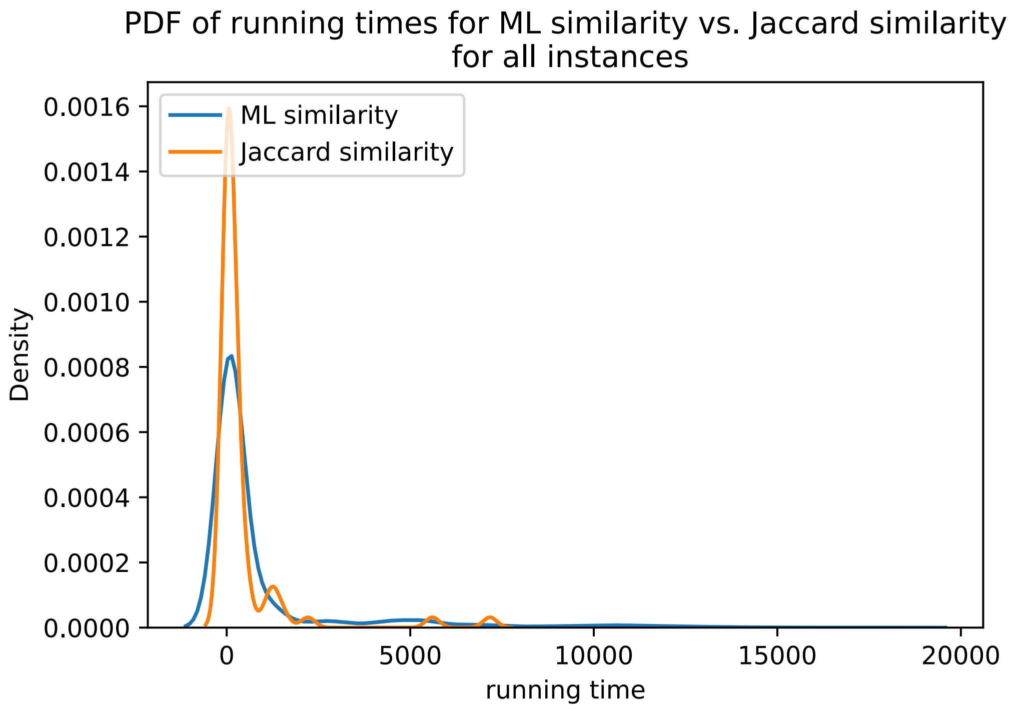


Figure 6.20: PDF comparison of all models vs. Jaccard similarity

Table 6.11: Mean runtime for benchmark datasets

benchmark dataset	ML similarity runtime	Jaccard similarity runtime
500/500	68s	43s
100-1000/100-1000	64s	46s
2000-8000/2000-8000	31min	11min
10k/10k	180min	105min

Especially very large instances like the 10k/10k instances in 6.21d or the 8000/8000 instance in 6.21c which causes the rightmost bump, create a significant difference in running times.

Overall our observation is that for large instances the increased running time becomes noticeable. In Table 6.11 we see the runtimes for the different benchmark datasets listed. The increased calculation effort induces a large increase in running times on instances of all sizes.

Table 6.12: Mean objective for all models on all benchmark instances [a]

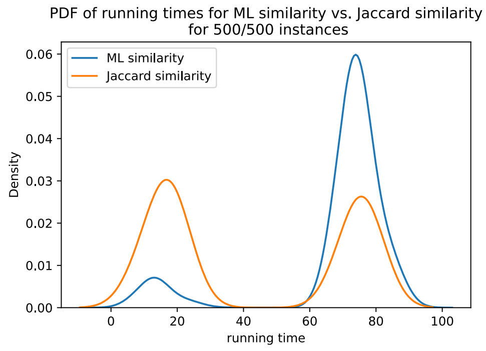
architecture	[40,20]															
batch size	batch size 256															
learning rate	0.0001															
training data size	100-1000/100-1000								500/500							
subproblem or k-data specifics	dependent		independent		k=max		k=30		dependent		independent		k=30		k=5	
subproblem or k-data specifics	jaccard	random	jaccard	random	all data	relevant	all data	relevant	jaccard	random	jaccard	random	all data	relevant	all data	relevant
500/500	48.1%	48.6%	47.9%	48.1%	46.3%	48.2%	46.8%	46.9%	47.5%	**49.0%**	47.7%	48.3%	47.0%	48.2%	47.0%	48.0%
100-1000/100-1000	48.5%	48.5%	48.0%	48.5%	46.7%	47.9%	47.5%	48.1%	47.6%	48.0%	47.8%	48.8%	47.4%	48.1%	47.1%	48.3%
2000-8000/2000-8000	46.6%	48.3%	44.8%	46.2%	46.0%	47.1%	*45.3%	47.4%	45.0%	45.4%	46.5%	47.7%	45.6%	*48.9%	45.6%	47.8%
10k/10k	42.5%	47.4%	38.0%	39.6%	41.9%	44.5%	*nan%	44.2%	44.7%	40.5%	38.7%	48.4%	40.8%	46.9%	45.9%	*nan%
architecture	[40,40]															
batch size	batch size 256															
learning rate	0.0001															
training data size	100-1000/100-1000								500/500							
subproblem or k-data specifics	dependent		independent		k=max		k=30		dependent		independent		k=30		k=5	
subproblem or k-data specifics	Jaccard	random	Jaccard	random	all data	relevant	all data	relevant	Jaccard	random	Jaccard	random	all data	relevant	all data	relevant
500/500	47.8%	48.6%	48.0%	48.9%	46.7%	46.8%	46.7%	47.8%	48.5%	48.5%	48.1%	48.9%	46.5%	48.5%	46.6%	47.8%
100-1000/100-1000	47.9%	48.7%	48.1%	48.8%	46.9%	47.6%	47.6%	48.3%	48.7%	48.2%	48.3%	48.4%	48.1%	48.4%	47.4%	48.0%
2000-8000/2000-8000	48.1%	46.4%	46.2%	46.9%	45.0%	46.2%	*48.1%	46.1%	47.0%	45.7%	48.2%	47.8%	46.3%	*47.9%	45.8%	*46.6%
10k/10k	45.0%	43.1%	39.2%	40.7%	41.3%	45.6%	*nan%	42.0%	41.4%	39.3%	45.5%	40.3%	39.3%	*nan%	42.1%	*nan%
architecture	[80,40]															
batch size	batch size 256															
learning rate	0.0001															
training data size	100-1000/100-1000								500/500							
subproblem or k-data specifics	dependent		independent		k=max		k=30		dependent		independent		k=30		k=5	
subproblem or k-data specifics	Jaccard	random	Jaccard	random	all data	relevant	all data	relevant	Jaccard	random	Jaccard	random	all data	relevant	all data	relevant
500/500	48.4%	48.5%	48.4%	47.6%	47.0%	48.2%	46.0%	47.6%	47.8%	48.9%	48.7%	48.9%	46.2%	47.0%	48.0%	48.2%
100-1000/100-1000	48.6%	48.8%	47.6%	48.2%	46.2%	48.1%	47.9%	48.2%	47.6%	48.6%	48.4%	49.0%	47.7%	47.8%	47.6%	48.5%
2000-8000/2000-8000	47.5%	48.5%	46.0%	47.5%	46.0%	47.0%	*46.3%	47.2%	44.2%	46.5%	48.0%	48.0%	45.9%	*48.9%	46.1%	*47.5%
10k/10k	44.1%	47.4%	45.0%	41.9%	41.2%	46.8%	*nan%	46.5%	39.1%	44.4%	47.7%	45.5%	46.6%	*nan%	46.2%	*nan%
architecture	[40,20]												[40,20]			
batch size	batch size 64												batch size 256			
learning rate	0.0001												0.001			
training data size	100-1000/100-1000						500/500						100-1000/100-1000		500/500	
subproblem or k-data specifics	dependent		independent		k=30		dependent		independent		k=30		k=30		k=30	
subproblem or k-data specifics	Jaccard	random	Jaccard	random	all data	relevant	Jaccard	random	Jaccard	random	all data	relevant	all data	relevant	all data	relevant
500/500	48.5%	48.3%	47.7%	48.1%	46.0%	47.5%	47.7%	48.3%	47.0%	47.8%	46.5%	48.8%	46.2%	48.2%	47.3%	48.0%
100-1000/100-1000	48.4%	48.4%	48.1%	48.7%	46.8%	48.2%	48.4%	48.6%	47.7%	48.8%	47.8%	48.5%	46.6%	47.5%	47.8%	48.3%
2000-8000/2000-8000	48.3%	48.0%	45.5%	47.4%	45.1%	47.0%	46.0%	44.6%	45.2%	46.9%	46.1%	*48.0%	45.3%	48.0%	47.8%	*47.4%
10k/10k	47.3%	48.2%	44.3%	44.8%	46.9%	44.6%	46.2%	37.4%	44.1%	43.4%	42.6%	47.9%	*nan%	46.4%	44.3%	40.1%
architecture	[40,20]												[40,40]			
batch size	batch size 64												batch size 256			
learning rate	0.001												0.001			
training data size	100-1000/100-1000						500/500						100-1000/100-1000		500/500	
subproblem or k-data specifics	dependent		independent		k=30		dependent		independent		k=30		k=30		k=30	
subproblem or k-data specifics	Jaccard	random	Jaccard	random	all data	relevant	Jaccard	random	Jaccard	random	all data	relevant	all data	relevant	all data	relevant
500/500	48.0%	48.3%	47.7%	48.3%	47.2%	47.9%	48.1%	48.3%	48.4%	48.9%	47.6%	48.8%	46.4%	44.1%	45.6%	48.1%
100-1000/100-1000	48.4%	48.3%	48.2%	48.1%	46.7%	47.5%	47.9%	48.1%	48.1%	49.1%	48.3%	48.5%	46.0%	45.4%	47.5%	47.5%
2000-8000/2000-8000	46.0%	47.9%	46.0%	46.3%	45.3%	46.6%	47.5%	46.0%	46.5%	48.6%	47.6%	*48.7%	46.1%	44.8%	46.8%	*49.0%
10k/10k	44.9%	45.6%	39.4%	45.0%	39.2%	43.9%	47.5%	45.1%	43.1%	47.8%	47.9%	45.6%	*nan%	43.7%	42.6%	*nan%

[a] Only partial results were obtained for elements signed with *

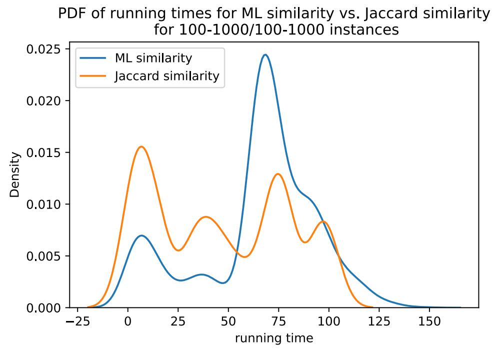
architecture	[40,40]											
batch size	batch size 64											
learning rate	0.0001											
training data size	100-1000/100-1000						500/500					
subproblem or k-data specifics	dependent		independent		k=30		dependent		independent		k=30	
subproblem or k-data specifics	Jaccard	random	Jaccard	random	all data	relevant	Jaccard	random	Jaccard	random	all data	relevant
500/500	48.8%	48.5%	47.8%	48.1%	47.0%	46.7%	47.9%	**49.0%**	48.1%	48.7%	46.4%	48.7%
100-1000/100-1000	48.6%	48.8%	47.9%	48.2%	47.2%	47.2%	47.5%	48.4%	48.0%	48.5%	47.3%	48.2%
2000-8000/2000-8000	48.0%	47.2%	46.4%	48.1%	47.3%	47.1%	44.9%	47.4%	42.1%	47.5%	44.4%	*49.2%
10k/10k	48.1%	44.2%	44.7%	46.6%	46.0%	46.6%	43.6%	47.8%	37.9%	42.5%	38.4%	44.4%
architecture	[40,40]											
batch size	batch size 64											
learning rate	0.001											
training data size	100-1000/100-1000						500/500					
subproblem or k-data specifics	dependent		independent		k=30		dependent		independent		k=30	
subproblem or k-data specifics	Jaccard	random	Jaccard	random	all data	relevant	Jaccard	random	Jaccard	random	all data	relevant
500/500	48.2%	48.5%	47.5%	48.6%	46.4%	45.8%	48.4%	47.9%	48.3%	48.2%	45.6%	46.3%
100-1000/100-1000	48.3%	48.5%	47.3%	48.6%	47.1%	46.6%	46.8%	48.5%	**49.2%**	48.5%	47.3%	47.6%
2000-8000/2000-8000	48.1%	**49.0%**	46.9%	48.0%	45.5%	45.8%	45.6%	47.1%	48.9%	46.0%	46.8%	*47.0%
10k/10k	46.9%	**48.5%**	38.8%	44.7%	45.7%	34.6%	44.9%	43.7%	48.4%	43.1%	45.8%	*nan%
architecture	[80,40]											
batch size	batch size 64											
learning rate	0.0001											
training data size	100-1000/100-1000						500/500					
subproblem or k-data specifics	dependent		independent		k=30		dependent		independent		k=30	
subproblem or k-data specifics	Jaccard	random	Jaccard	random	all data	relevant	Jaccard	random	Jaccard	random	all data	relevant
500/500	47.5%	48.6%	47.6%	48.7%	45.9%	46.5%	48.4%	48.0%	47.2%	48.4%	46.5%	47.7%
100-1000/100-1000	47.8%	48.3%	47.3%	48.5%	46.8%	47.7%	46.9%	48.3%	48.0%	48.8%	47.1%	48.0%
2000-8000/2000-8000	47.4%	47.7%	46.6%	46.7%	43.9%	47.0%	47.0%	46.4%	48.1%	48.6%	47.0%	*48.7%
10k/10k	42.8%	46.3%	41.3%	38.9%	39.7%	45.8%	46.8%	40.9%	47.3%	48.4%	43.3%	47.7%
architecture	[80,40]											
batch size	batch size 64											
learning rate	0.001											
training data size	100-1000/100-1000						500/500					
subproblem or k-data specifics	dependent		independent		k=30		dependent		independent		k=30	
subproblem or k-data specifics	Jaccard	random	Jaccard	random	all data	relevant	Jaccard	random	Jaccard	random	all data	relevant
500/500	47.8%	48.5%	47.6%	48.4%	46.1%	43.8%	46.6%	47.4%	48.5%	47.4%	46.1%	47.0%
100-1000/100-1000	48.3%	48.1%	48.0%	48.2%	45.8%	47.0%	47.5%	48.5%	48.2%	47.9%	48.0%	48.0%
2000-8000/2000-8000	47.7%	48.0%	44.1%	46.4%	46.3%	46.9%	43.4%	48.2%	47.1%	47.3%	47.4%	*45.6%
10k/10k	47.4%	47.5%	40.7%	44.2%	40.4%	43.1%	40.3%	47.6%	47.6%	46.2%	45.8%	*nan%
architecture	[80,40]											
batch size	batch size 256											
learning rate	0.001											
training data size	100-1000/100-1000						500/500					
subproblem or k-data specifics	k=30						k=30					
subproblem or k-data specifics	all data			relevant			all data			relevant		
500/500	45.9%			44.9%			45.4%			47.8%		
100-1000/100-1000	46.8%			45.7%			47.4%			48.1%		
2000-8000/2000-8000	45.6%			46.5%			45.6%			*47.5%		
10k/10k	*nan%			43.4%			36.2%			*nan%		

Table 6.13: Mean objective for all models on all benchmark instances [a]

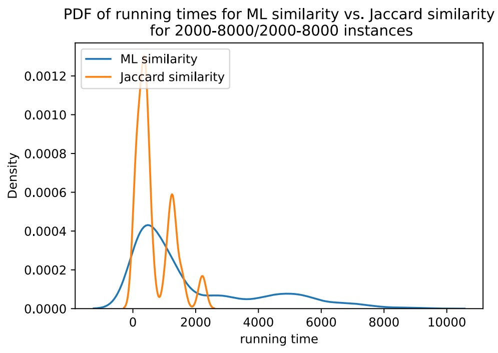
[a]Only partial results were obtained for elements signed with *



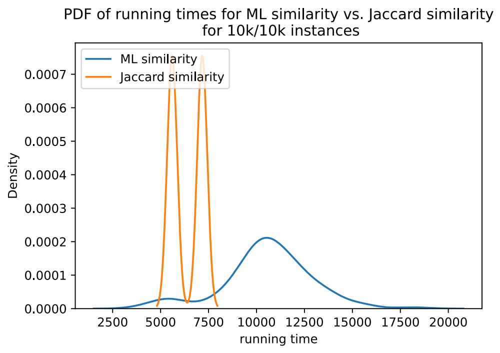
(a) PDF comparison for the 500/500 instances runtimes



(b) PDF comparison for the 100-1000/100-1000 instances runtimes



(c) PDF comparison for the 2000-8000/2000-8000 instances runtimes



(d) PDF comparison for the 10k/10k instances runtimes

Figure 6.21: PDF comparison of all models vs. Jaccard runtimes

CHAPTER 7

Conclusion and Future Work

In this work, we considered the Demand Maximizing Battery Swapping Station Location Problem, a maximum coverage location problem. The motivation behind solving this problem is to optimize the infrastructure for a network of battery swapping stations for electric vehicles such that as many users as possible can swap batteries during their trips. This infrastructure involves the areas containing the battery swapping stations, the number of slots where batteries can be charged at a station, and the specific allocation of users to certain stations.

We first defined a graph representation of the DMBSSLP on a graph $P(V, E)$ and then transformed this representation into a bipartite graph representation $G(Q, L, E)$.

To solve the DMBSSLP we developed a novel machine learning supported Multilevel Optimization algorithm called the Learning Multilevel Optimization. This algorithm is built upon the Multilevel Optimization algorithm and uses two neural networks to guide the partitioning process of the algorithm. To create an implementation of this algorithm we first devised a Mixed Integer Linear Programming formulation for the DMBSSLP which allows us to solve smaller instances efficiently and accurately. We used this MILP formulation and the standard MLO to create training data for the neural networks. Two different strategies were applied to create training data: the subproblem strategy, which creates data for two individually merged nodes, and the k-data strategy which creates aggregated data points for multiple merges.

These data collection strategies were then divided into multiple further categories. For the subproblem strategy, we experimented with different partitioning strategies of the training data, and different settings for the variable δ during the subproblem solving. For the k-data strategy, we experimented with different settings for the value k and different parameters for the data relevance, where we used only relevant data or all data for training.

For the neural network parameters, we experimented with different settings for regularization, architecture, learning rate, and batch size.

In total, this resulted in 132 different parameter combinations for which we trained models and which we investigated closer. Many more models did not make it into the final analysis as they were rejected early.

From all these parameters we identified some that are important for the applicability of the models. The most important one for the subproblem strategy was an unbiased partitioning strategy during data creation. For the k-data strategy smaller values of k and exclusively using relevant data during training resulted in the best performance.

For a better overall model performance, it was also important that the learning rate was low and the batch size was high resulting in a slower more accurate learning process, although the very best models violated this principle and used a high learning rate and low batch size.

Overall we showed that using the LMLO can improve the performance of conventional MLO strategies significantly. The cost associated with this increase in objective value is an increased time demand for the neural network predictions. This increase in runtime and objective value becomes more distinct the bigger the problem instances that are solved.

7.1 Future Work

At the moment our implementation of the algorithm is rather memory intensive. Part of the reason for this is that during the similarity calculation a dataframe of size $|h| \times |E|$ is created. This dataframe can get very large and thus takes a lot of time to create and predict. A possible solution for that would be to apply a different partitioning algorithm like heavy edge matching which we expect to produce worse results as it relies much more on randomness but could solve the problem of large memory requirements.

Also concerning the partitioning: At the moment if two nodes are merged they must not be merged any further in this coarsening step. This creates a partnering problem at the end of the partitioning process. When only few nodes are left unpartitioned they will be merged with each other regardless of their similarity value. This means that nodes that are bad to merge will be merged anyway. A solution would be to allow the partitioning of multiple nodes at the same time. If two nodes are merged they create a temporary merged node which can be merged again in the same step. This strategy could potentially decrease the number of bad merges being made but creates an increased calculation demand because, for each temporary node, all similarities of neighbors have to be added to the heap of similarities while similarities with the old already merged nodes need to be deleted. Overall this additional computational cost could probably improve the results but further tests would be needed.

As a final remark for the future outlook, we want to discuss the potential applicability of graph neural networks (GNNs) instead of dense neural networks for similarity calculation.

As the problem is already defined on a graph, applying GNNs to this structure could potentially improve the results. GNNs use message passing and excel at capturing more information on the complete structure of a graph. We did not apply this method yet as the time constraints of this work did not allow deviations from the initial plan we proposed but we hope to explore this approach further in future work.

List of Mathematical Symbols

a_{ql}	demand of OD-pair q assigned to be satisfied in area l
$\mathbf{A}$	The matrix of size $k \times \lvert h \rvert$ representing an observation in the k-data strategy
b	costs for one additional charging slot
B	The Budget available for a project
c	costs for setting up a charging station
d_q	demand of an OD-pair $q \in Q$
$\bar{d}_l$	The sum of all demands that might be satisfied at area l
δ	Leftover or so far unused demand that may be assigned again in later subproblem solutions
e_{ql}	The maximum demand that can be assigned from OD-pair q to area l.
E	edge set of graph G
G	bipartite Graph
h_L^0	feature matrix for all 2-hop connected pairs of nodes in L
h_Q^0	feature matrix for all 2-hop connected pairs of nodes in Q
l	an area of L or the respective area set of the coarsened graphs
L	set of considered areas for placing battery swapping station
m	number of OD-pairs
n	number of areas in L
$N(l), N(q)$	set of neighboring nodes of an area node l or OD-pair node q in G
q	an OD-pair from Q or the respective sets of the coarsened graphs
Q	a set of m O/D pairs
r_l	maximum number of stations in area l
s	maximum number battery charging slots in any station
x_l	number of stations to be built in area l
$\mathbf{X}$	The matrix of dependant features h for the subproblem strategy or $\mathbf{A}$ for the k-data strategy
y_l	total number of charging slots to be built in the stations of area l
$z_{q_1 q_2}$	The error of merging nodes $q_1, q_2 \in Q$ in the subproblem strategy
$z_{l_1 l_2}$	The error of merging nodes $l_1, l_2 \in L$ in the subproblem strategy
z_j	The error caused by observation j in the k-data strategy

List of Abbreviations

ADAM	Adaptive Momentum Estimation
CH	Construction Heuristic
CMCLP	Capacitated Maximum Coverage Location Problem
CP	Covering Problems
DMBSSLP	Demand Maximizing Battery Swapping Station Location Problem
EST	Euclidean Spanning Tree
EV	Electric Vehicle
GHEM	Greedy Heavy Edge Matching
GNN	Graph Neural Network
HEM	Heave Edge Matching
LMLO	Learning Multi Level Optimization
LP	Linear Program
MBSSLP	Multi-Period Battery Swapping Station Location Problem
MCLP	Maximum Coverage Location Problems
MCMCLP	Modular Capacitated Maximal Covering Location Problem
MCMCLPSC	Modular Capacitated Maximum Coverage Location Problem with Setup Cost
MILP	Mixed Integer Linear Program
MLO	Multi Level Optimization
MLR	Multi Level Refinement
MSE	Mean Squared Error
NN	Neural Network
PDF	Probability Density Function
WRST	Wilcoxon Rank Sum Test

List of Figures

List of Tables

List of Algorithms

Bibliography

[AMVC15] Sajjad Allahi, Mohammadsadegh Mobin, Amin Vafadarnikjoo, and Christian Salmon. An integrated AHP-GIS-MCLP method to locate bank branches. In *Proceedings of the 2015 Industrial and Systems Engineering Research Conference*, pages 1104–1113, 2015.

[Ber57] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957.

[BLM15] Shahzad F. Bhatti, Michael K. Lim, and Ho-Yin Mak. Alternative fuel station location model with demand learning. *Annals of Operations Research*, 230(1):105–127, 2015.

[BS94] Stephen T Barnard and Horst D Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.

[BT97] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to Linear Optimization*. Athena scientific, 3rd edition, 1997.

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2022.

[Coo63] Leon Cooper. Location-allocation problems. *Operations Research*, 11(3):331–343, 1963.

[CR74] Richard Church and Charles ReVelle. The maximal covering location problem. *Papers of the Regional Science Association*, 32(1):101–118, 1974.

[CS88] John Richard Current and James Edward Storbeck. Capacitated covering models. *Environment and Planning B: Planning and Design*, 15(2):153–163, 1988.

[Del34] Boris Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[DOW55] George B Dantzig, Alex Orden, and Philip Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

[EA16] Sahar K Elkady and Hisham M Abdelsalam. A modified multi-objective particle swarm optimisation algorithm for healthcare facility planning. *International Journal of Business and Systems Research*, 10(1):1–22, 2016.

[GJ79] Michael R Garey and David S Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

[Hak65] Seifollah Louis Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13(3):462–475, 1965.

[HGNN21] Masoud Hatami Gazani, Seyed Armin Akhavan Niaki, and Seyed Taghi Akhavan Niaki. The capacitated maximal covering location problem with heterogeneous facilities and vehicles and different setup costs: An effective heuristic approach. *International Journal of Industrial Engineering Computations*, 12(1):79–90, 2021.

[HL$^+$95] Bruce Hendrickson, Robert W Leland, et al. A multi-level algorithm for partitioning graphs. *SC*, 95(28):1–14, 1995.

[HPBL23] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[Jac02] Paul Jaccard. Lois de distribution florale dans la zone alpine. *Bulletin de la Société vaudoise des sciences naturelles*, 38:69–130, 01 1902.

[JORR20] Thomas Jatschka, Fabio F. Oberweger, Tobias Rodemann, and Günther R. Raidl. Distributing battery swapping stations for electric scooters in an urban area. In Nicholas Olenev, Yuri Evtushenko, Michael Khachay, and Vlasta Malkova, editors, *Optimization and Applications, Proceedings of OPTIMA 2020 – XI International Conference Optimization and Applications*, volume 12422 of *LNCS*, pages 150–165. Springer, 2020.

[JRR23] Thomas Jatschka, Tobias Rodemann, and Günther R. Raidl. A multilevel optimization approach for large scale battery exchange station location planning. In Leslie Pérez Cáceres and Thomas Stützle, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 13987 of *LNCS*, page 50–65. Springer, 2023.

[KAKS97] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In

Proceedings of the 34th annual Design Automation Conference, pages 526–529, 1997.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KBP14] Tony Lee Kerzmann, Gavin A. Buxton, and Jonathan Preisser. A computer model for optimizing the location of natural gas fueling stations. *Sustainable Energy Technologies and Assessments*, 7:221–226, 2014.

[LNS15] Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama. *Location Science*. Springer, 1st edition, 2015.

[MG19] Jean-Yves Potvin Michel Gendreau. *Handbook of Metaheuristics*. Springer, 3rd edition, 2019.

[MSS14] Henning Meyerhenke, Peter Sanders, and Christian Schulz. Partitioning complex networks via size-constrained clustering. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Experimental Algorithms: 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29–July 1, 2014. Proceedings 13*, pages 351–363. Springer, 2014.

[Obs22] Daniel Obszelka. Lecture notes in Algorithmics. Lecture Notes by Algorithms and Complexity institute TU Wien, 2022.

[PS91] Hasan Pirkul and David A Schilling. The maximal covering location problem with capacities on total workload. *Management Science*, 37(2):233–248, 1991.

[RAK07] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[RKJ$^+$23] Tobias Rodemann, Hiroaki Kataoka, Thomas Jatschka, Günther R. Raidl, Steffen Limmer, and Meguro Hiromu. Optimizing the positions of battery swapping stations. In *Proceedings to the 6th International Electric Vehicle Technology Conference*, 2023.

[Rot69] R Roth. Computer solutions to minimum-cover problems. *Operations Research*, 17(3):455–465, 1969.

[RSW07] Demane Rodney, Alan Soper, and Chris Walshaw. The application of multilevel refinement to the vehicle routing problem. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 212–219. IEEE, 2007.

[TSRB71] Constantine Toregas, Ralph Swain, Charles ReVelle, and Lawrence Bergman. The location of emergency service facilities. *Operations Research*, 19(6):1363–1373, 1971.

[VAdPF^{+}21] Alan Demétrius Baria Valejo, Paulo Eduardo Althoff, Thiago de Paulo Faleiros, Maria Lígia Chuerubim, Jianglong Yan, Weiguang Liu, and Liang Zhao. Coarsening algorithm via semi-synchronous label propagation for bipartite networks. In *Intelligent Systems: 10th Brazilian Conference, BRACIS 2021, Virtual Event, November 29–December 3, 2021, Proceedings, Part I*, pages 437–452. Springer, 2021.

[VdOdSNZ21] Alan Demétrius Baria Valejo, Wellington de Oliveira dos Santos, Murilo Coelho Naldi, and Liang Zhao. A review and comparative analysis of coarsening algorithms on bipartite networks. *The European Physical Journal Special Topics*, 230(14-15):2801–2811, 2021.

[VdOGFdAL18] Alan Valejo, Maria Cristina Ferreira de Oliveira, PR Geraldo Filho, and Alneu de Andrade Lopes. Multilevel approach for combinatorial optimization in bipartite network. *Knowledge-Based Systems*, 151:45–61, 2018.

[VFdOdAL20] Alan Valejo, Thiago Faleiros, Maria Cristina Ferreira de Oliveira, and Alneu de Andrade Lopes. A coarsening method for bipartite networks via weight-constrained label propagation. *Knowledge-Based Systems*, 195:105678, 2020.

[VFF^{+}20] Alan Valejo, Vinícius Ferreira, Renato Fabbri, Maria Cristina Ferreira de Oliveira, and Alneu de Andrade Lopes. A critical survey of the multilevel method in complex networks. *ACM Computing Surveys (CSUR)*, 53(2):1–35, 2020.

[Wal02] Chris Walshaw. A multilevel approach to the travelling salesman problem. *Operations Research*, 50(5):862–877, 2002.

[Wal08] Chris Walshaw. *Hybrid metaheuristics: an emerging approach to optimization*, chapter Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance, pages 261–289. Springer, 2008.

[Wan08] Ying-Wei Wang. Locating battery exchange stations to serve tourism transport: A note. *Transportation Research Part D: Transport and Environment*, 13(3):193–197, 2008.

[Wil92] Frank Wilcoxon. *Breakthroughs in Statistics: Methodology and Distribution*, chapter Individual Comparisons by Ranking Methods, pages 196–202. Springer, 1992.

[XYD$^+$09] Ming Xie, Wenjun Yin, Jin Dong, Jinyan Shao, and Lili Zhao. A marginal increment assignment algorithm for maximal coverage location problem. In *2009 IEEE/INFORMS International Conference on Service Operations, Logistics and Informatics*, pages 651–656. IEEE, 2009.

[YCW$^+$14] Jia Yu, Yun Chen, Jianping Wu, Rui Liu, Hui Xu, Dongjing Yao, and Jing Fu. Particle swarm optimization based spatial location allocation of urban parks—a case study in baoshan district, shanghai, china. In *2014 The Third International Conference on Agro-Geoinformatics*, pages 1–6. IEEE, 2014.

[YK16] Huairen Ye and Hyun Kim. Locating healthcare facilities using a network-based covering location problem. *GeoJournal*, 81(6):875–890, 2016.

[YM12] Ping Yin and Lan Mu. Modular capacitated maximal covering location problem for the optimal siting of emergency vehicles. *Applied Geography*, 34:247–254, 2012.