



Using Graph Neural Networks in Local Search for Edge-Based Relaxations of the Maximum Clique Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Rupert Ettrich, BA BSc

Matrikelnummer 01129393

an der Fakultät für Informatik

der Technischen Universität Wien

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

Mitwirkung: Projektass. Marc Huber, MSc

Wien, 7. Dezember 2022

Rupert Ettrich

Günther Raidl

Using Graph Neural Networks in Local Search for Relaxations of the Maximum Clique Problem

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Rupert Ettrich, BA BSc

Registration Number 01129393

to the Faculty of Informatics

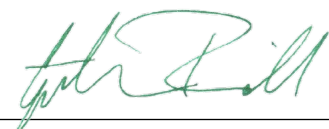
at the TU Wien

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

Assistance: Projektass. Marc Huber, MSc

Vienna, 7th December, 2022


Rupert Ettrich


Günther Raidl

Erklärung zur Verfassung der Arbeit

Rupert Ettrich, BA BSc

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, 7. Dezember 2022



Rupert Ettrich

Danksagung

Zuerst möchte mich bei meinen Eltern bedanken, die es mir ermöglicht haben, mich an einem zweiten Studium zu versuchen. Ohne euch hätte ich vielleicht nie meine Leidenschaft für die Informatik entdeckt. Danke für eure langjährige Unterstützung.

Mein großer Dank gebührt außerdem Günther Raidl und Marc Huber, deren großartige Betreuung im Rahmen dieser Diplomarbeit weit über das hinaus gegangen ist, was ich mir erwartet habe. In unseren zahlreichen Meetings habt ihr mir den Weg gewiesen, wenn ich nicht weiter wusste, und stets wertvolle Ideen, Einwände und Feedback beigesteuert.

Danke an die Algorithms and Complexity Group der TU Wien, die mir die notwendige Infrastruktur und Rechenleistung zur Verfügung gestellt hat, um die Experimente in dieser Arbeit durchzuführen.

Ebenfalls möchte ich mich bei Elina Rönning bedanken, die einige interessante Ideen beigesteuert hat.

Abschließend möchte ich mich bei meiner Freundin Rabea bedanken. Du hast mir die Kraft und Motivation gegeben weiterzumachen, wenn es mir schwer gefallen ist.

Acknowledgements

First, I would like to thank my parents, who made it possible for me to go for another education. Without you, I maybe would never have discovered my passion for computer science. Thank you for your support throughout all these years.

Special thanks go to my supervisors Günther Raidl and Marc Huber, whose support in this thesis was nothing but outstanding. In countless meetings, you lead my way when I did not know how to continue, and continually contributed valuable ideas, corrections, and feedback to this thesis.

Furthermore, I would like to thank the Algorithms and Complexity Group at TU Wien, who provided me with the necessary infrastructure and computation power to do the computational experiments in this thesis.

I would also like to thank Elina Rönnerberg, who contributed some interesting ideas to this thesis.

Finally, I want to thank my girlfriend Rabea. You gave me the strength and motivation to continue when I struggled.

Kurzfassung

Diese Arbeit untersucht den Einsatz von *Graph Neural Networks* (GNNs) in einer Metaheuristik für kantenbasierte Relaxationen des Maximum Clique Problems (MCP), dem Maximum Quasi-Clique Problem (MQCP) und dem Maximum Defective Clique Problem (MDCP). Das Ziel des MCPs ist es, eine Knotenmenge maximaler Kardinalität zu finden, sodass alle Knoten miteinander durch eine Kante verbunden sind. Das Problem ist ein fundamentales NP-schweres kombinatorisches Optimierungsproblem mit zahlreichen Anwendungen, z.B. in der Bioinformatik oder der Analyse von sozialen Netzwerken. In manchen Anwendungen ist es notwendig, dass nicht nach vollständigen, sondern sehr dichten Subgraphen gesucht wird, weswegen das MQCP und das MDCP eingeführt wurden.

Im Rahmen dieser Arbeit orientieren wir uns an den führenden heuristischen Methoden für die genannten Probleme und entwickeln eine Metaheuristik, LSBM, die auf lokaler Suche basiert und ein GNN verwendet, um die Nachbarschaft des derzeitigen Lösungskandidaten auf Knoten einzuschränken, die am ehesten zu einer Verbesserung der Lösung führen. In diesem Zusammenhang präsentieren wir auch einen Trainingsalgorithmus LSBM-T, der verwendet wird, um das GNN zu trainieren, eine Nachbarschaftssuche zu imitieren. Wir untersuchen unterschiedliche Ansätze, die Knoten im GNN zu initialisieren und verwenden eine Encoder-Decoder Architektur für das GNN, die auf dem *Attention*-Mechanismus aufbaut.

Die Ergebnisse unserer Evaluierung zeigen, dass unser Ansatz auf Benchmark-Instanzen der Größe $|V| \leq 300$ mit den führenden Methoden, die algorithmisch komplexer aufgebaut sind, in Bezug auf Lösungsqualität mithalten kann, auf größeren Instanzen jedoch nicht. Darüber hinaus zeigen wir, dass unser Trainingsalgorithmus LSBM-T das GNN erfolgreich trainiert, und der Einsatz des GNNs in LSBM zu deutlich besseren Ergebnissen führt als eine andere häufig verwendete Knotenbewertungsfunktion.

Abstract

This thesis investigates the utilization of Graph Neural Networks (GNNs) in a local search-based metaheuristic for edge-based relaxations of the Maximum Clique Problem (MCP), namely, the Maximum Quasi-Clique Problem (MQCP) and the Maximum Defective Clique Problem (MDCP). The MCP is the problem of finding fully connected subsets of vertices of maximum size in a graph. It is a well-studied, fundamental NP-hard combinatorial optimization problem and has numerous applications in, e.g., bioinformatics and social network analysis. Some real-world applications however require relaxations of the clique model. Thus, researchers have come up with the MQCP and MDCP as a theoretical model for such applications.

Due to the computational complexity of the considered problems, real-world instances are often infeasible to solve in practice using exact methods. Therefore, we develop a heuristic solution approach that is centered around local search, like most state-of-the-art algorithms for the MQCP and MDCP. We study and review the relevant literature and build upon the leading (meta)heuristic algorithms for the MQCP.

The first main contribution of this thesis is a local search-based metaheuristic named LSBM that can utilize a GNN as a scoring function for restricting the neighborhoods during the local search procedure to the most promising vertices. Secondly, we propose a training algorithm LSBM-T that trains the GNN-based scoring function offline on randomly generated representative instances in an imitation learning setting. The expert strategy being imitated is an exhaustive search of a user-defined neighborhood structure. The GNN is thus trained to predict, which vertices in a neighborhood will most likely lead to an improved solution. We investigate different centrality-based and learning-based feature initialization methods, as the input graphs of the considered problems are non-attributed. Furthermore, we present an attention-based Encoder-Decoder GNN architecture built upon similar applications of GNNs in combinatorial optimization.

The results on benchmark instances from the literature indicate that our approach can match the solution quality of the algorithmically more complex state-of-the-art methods for most instances with $|V| \leq 300$, but is not competitive yet for larger instances. However, it can clearly be seen that our training algorithm LSBM-T successfully trains the GNN-based scoring function to effectively guide the search, and LSBM with the trained GNN-based scoring function yields substantially better solutions than LSBM with a so far most often used scoring function.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 Motivation	1
1.2 Methodological Approach	2
1.3 Outline of the Thesis	3
2 Considered Problems and Definitions	5
2.1 Notation	5
2.2 The Maximum Clique Problem	6
2.3 The Maximum Quasi-Clique Problem	8
2.4 The Maximum Defective Clique Problem	8
3 Related Work	11
3.1 Graph Neural Networks	11
3.2 Machine Learning in Combinatorial Optimization	14
3.3 Node Representation Learning	15
3.4 The Maximum Quasi-Clique Problem	17
3.5 The Maximum Defective Clique Problem	21
4 A Local Search Based Metaheuristic for Edge-Based Relaxations of the Maximum Clique Problem	23
4.1 Algorithm Structure	24
4.2 Lower Bound Heuristic	26
4.3 Construction Heuristic	28
4.4 Neighborhood Structure and Local Search	29
4.5 Adaption to the Maximum Defective Clique Problem	31
5 A GNN-based Metaheuristic	35
5.1 Motivation and Overview	35
	xv

5.2	Training the GNN	38
5.3	Look-Ahead Search	41
5.4	GNN Architecture	44
5.5	Performance	49
6	Computational Experiments & Evaluation	53
6.1	Lower Bound Heuristic	55
6.2	Evaluation of Training Parameters	56
6.3	Evaluation of Search Parameters	67
6.4	Results on Benchmark Instances	69
7	Conclusions & Future Work	75
	List of Figures	79
	List of Tables	81
	List of Algorithms	83
	Bibliography	85


Introduction

This thesis investigates the utilization of Graph Neural Networks (GNNs) in the context of a local search-based metaheuristic for solving edge-based relaxations of a well-known combinatorial optimization problem (COP), the Maximum Clique Problem (MCP). The motivation of this work is presented in Section 1.1, as well as the relevance and applications of the considered problems. A brief overview of the contributions of this work is provided in Section 1.2. Lastly, we outline the structure of this thesis in Section 1.3.

1.1 Motivation

In many COPs, problem instances exhibit clearly defined internal structures that can be expressed as graphs. Here, a graph is a tuple $G = (V, E)$, where V is the set of vertices and the set of edges $E \subseteq V \times V$ defines the relationships among vertices. While there are other methods to deal with inputs of variable size (Fully Convolutional Networks, Recurrent Neural Networks), GNNs are Neural Networks tailored specifically to learn from structured input in the form of graphs, making them a valuable tool for Machine Learning (ML) tasks on data with graph-like structure.

In recent years, GNNs have gained popularity in their application in the context of COPs. However, current end-to-end ML approaches are in most cases not competitive to state-of-the-art (meta-)heuristic solution approaches, and their application is limited to small instances, where effective exact algorithms are available. Nonetheless, GNNs show promise in their use in COPs, and there have been many successful applications over the last years, e.g. [ORRH22], where a Large Neighborhood Search is enhanced by a GNN that guides a destroy-operator, or [DHM21], where a GNN is used to find maximal independent sets by imitating a time-expensive Monte-Carlo Tree Search, producing solutions that reach a solution quality of 99.5% while being three orders of magnitude faster.



The main motivation of this thesis is to further study the application of GNNs in the context of metaheuristics for COPs defined on graphs. We address the problems of current end-to-end approaches by using a GNN only as a component of a metaheuristic search procedure that shall provide additional heuristic guidance. More specifically, we consider relaxations of a well-studied COP, the MCP.

The MCP is the problem of finding a fully connected subgraph – a *clique* – of maximum size in a given graph. It is a fundamental problem in computer science, as its decision variant is one of Karp’s 21 NP-complete problems [Kar72]. The MCP has several practical applications, e.g., in bioinformatics [MAY10] and social network analysis [PYB13]. However, for some real-world applications that require identifying dense subgraphs, the MCP is too strict a model. This leads to the introduction of several clique relaxations such as – among others – the Maximum Quasi-Clique Problem (MQCP) (introduced in [ARS02], Definition 2.3.1), and the Maximum k -defective Clique Problem (MDCP) (introduced in [YPTG06], Definition 2.4.1).

As all of these problems are NP-hard optimization problems, it is practically often infeasible to obtain exact solutions for large instances. However, many real-world applications often require solutions for large graphs. Therefore, efficient heuristic methods are needed that produce high-quality solutions in an acceptable amount of time. While the MCP has been studied extensively over the last decades, heuristic methods for the MQCP and the MDCP are less abundant. It is therefore another motivation of this thesis to enrich the arsenal of heuristic methods for these relaxations of the MCP and to provide a foundation for future research including the application of GNNs in the context of MCP relaxations.

1.2 Methodological Approach

The goal of this thesis is to build upon well-established metaheuristic approaches for MCP relaxations and evaluate, how and where GNNs can be utilized in the context of such algorithms to provide additional information. Studying the relevant literature (e.g., [DHB19], [ZBW20], [CCP⁺21], [PWWW21]), we found that the most effective metaheuristic approaches for MCP relaxations are based on metaheuristics centered around local search, where neighboring solutions can be reached by swapping a vertex inside the current candidate solution with a vertex outside the candidate solution. In order to evaluate efficiently which vertices seem promising for swapping, a scoring function is used that assigns scores to the vertices in the graph in each iteration of the local search. Using these scores, the neighborhood can be restricted to only promising vertices. Building upon these approaches, we aim at utilizing a GNN to extract structural information from input graphs that can be used to enhance such a scoring function. One of the main contributions of this thesis is therefore the algorithmic design of a local search-based metaheuristic named LSBM, which can use a GNN-based scoring function.

As the considered MCP relaxations are defined on non-attributed graphs, one of the goals of this thesis is to evaluate, which feature initialization methods can be used to effectively

extract information from input graphs. Thus, we investigate several centrality-based and learning-based methods within the context of our proposed algorithm that can be found in the literature.

Furthermore, we propose a new algorithm that is used to generate training data in order to train the GNN to make high-quality predictions. A training sample consists of the input graph, a candidate solution, and target values obtained by a look-ahead search that searches a neighborhood structure for a neighboring solution with a higher objective value. Using this data, the GNN is trained to predict, which vertices are likely to be in a swap that leads to an improved solution. In order to compute target labels for the vertices in the input graph, we present different methods to search neighborhood structures relative to the candidate solution in a training sample.

Finally, we conduct computational experiments to evaluate our approach on graphs of different sizes and densities and discuss future work that can be done to improve our method.

1.3 Outline of the Thesis

Firstly, we define the notation of graph-related terms used throughout this thesis and show formal definitions of the considered problems in Chapter 2. Afterward, we review and discuss related work and methodology, namely GNNs, node representation learning, the use of ML in combinatorial optimization, and the state-of-the-art for both exact and heuristic solution approaches for the considered MCP relaxations in Chapter 3. Then, in Chapter 4 we propose our algorithm LSBM and its components in detail. The main contribution of our work is contained in Chapter 5, as we present, how a GNN can be incorporated into the previously presented LSBM and how it can be trained using our training algorithm LSBM-T. Thereafter, we show the results of computational experiments and the evaluation of our proposed methods in Chapter 6. Finally, we summarize our findings, offer concluding remarks, and outline promising future work in Chapter 7.

Considered Problems and Definitions

In this Chapter, we provide all necessary definitions and notations used throughout this thesis. We list commonly used graph theoretic notations and definitions in Section 2.1. Afterward, Sections 2.2 – 2.4 show definitions for the problems considered in this thesis, starting with the Maximum Clique Problem, from which all considered problems are derived. Additionally, we list notable properties of the considered problems that are relevant for developing well-performing algorithms for these problems.

2.1 Notation

Throughout this thesis, we use standard graph theory notation. We consider $G = (V, E)$ to be an undirected, simple graph with vertex set V and edge set $E \subseteq \{\{u, v\} \mid u, v \in V\}$. We use the terms vertex, vertices when denoting elements of V in a graph G , and we use the terms node, nodes when denoting elements of a search graph or search tree corresponding to a search algorithm, e.g. in beam search, or nodes in a GNN. This distinction helps to keep formulations precise when for instance an algorithm that uses an internal graph structure is applied to a graph input, or to distinguish between the vertices of a graph and the corresponding nodes of a GNN. If not stated otherwise, $n = |V|$ denotes the number of vertices in G , whereas $m = |E|$ denotes the number of edges. For a graph G we also use $V(G), E(G)$ to denote its vertex set and edge set, respectively.

A graph G is *simple* if it contains no self-loops (e.g., $\{v, v\} \in E$) or multiple edges between two vertices. In this thesis, we only consider unweighted, undirected, simple graphs, as all the considered problems are defined on such graphs. A graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ iff $V' \subseteq V$, $E' \subseteq E$, and $\{u, v\} \in E' : u, v \in V'$. Let $S \subseteq V$ be a set of vertices in G . The subgraph induced by S in G , denoted as $G[S]$, is a graph

with vertex set S and edge set $E(S) = \{\{u, v\} \mid \{u, v\} \in E, u, v \in S\}$. The *diameter* of a graph G is the length of the longest shortest path between any two vertices in G .

A *clique* is a set of vertices $S \subseteq V$ such that $G[S]$ is fully connected, i.e., $|E(S)| = \frac{|S| \cdot (|S|-1)}{2}$. A fully connected graph on n vertices is denoted as K_n . A clique of size k is denoted as a k -clique, and this convention is also applied to clique relaxation models: A γ -quasi clique of size k is denoted as a k - γ -quasi clique, an s -defective clique of size k is denoted as a k - s -defective clique, etc.

The density $\text{dens}(G)$ of a graph $G = (V, E)$ is defined as the ratio of the number of edges $|E|$ to the number of edges in a fully-connected graph with $|V|$ vertices $\binom{|V|}{2}$. The density of a graph is therefore a rational number between zero and one.

The open neighborhood of a vertex v in a graph $G = (V, E)$, denoted as $N_G(v) = \{w \mid v \in V, \{v, w\} \in E\}$ is the set of vertices that are adjacent to v , whereas the closed neighborhood of v , denoted as $N_G[v] = N_G(v) \cup \{v\}$ is the set of vertices that are adjacent to v , and v itself. If the graph G is clear from context, we use $N(v)$ or $N[v]$ to denote the open and closed neighborhoods of v , respectively. The neighborhood of a set of vertices is defined similarly: For $S \subseteq V$, let $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$ be the open neighborhood of S in G , and let $N_G[S] = N_G(S) \cup S$ be the closed neighborhood of S in G .

The degree of a vertex v in a simple graph $G = (V, E)$ is defined as $\text{deg}(v) = |N_G(v)|$, which is the number of adjacent vertices for a vertex $v \in V$. Generalizing the concept of vertex degrees, for $S \subseteq V$, let $d_S(v) = |\{w \mid \{v, w\} \in E, w \in S\}|$ be the number of vertices in S that are adjacent to $v \in V$. As can be seen, $\text{deg}(v) = d_V(v)$ for all $v \in V$.

The *EgoNet* of distance d of a vertex in v in a graph G is defined as follows: Given a vertex $v \in V$, let $N(v, d)$ be the set of vertices reachable from v by a path of length at most d . The induced subgraph $G[N(v, d)]$ is the d -hop-EgoNet of vertex v . Note that for $d = 1$, $N(v, d)$ is the closed neighborhood of v in G .

2.2 The Maximum Clique Problem

The MCP is a fundamental problem in graph theory and computer science. Its decision variant was one of the first problems that were shown to be NP-Complete [Kar72], from which NP-hardness can be derived for the maximization variant. The MCP is a well-studied problem and has many real-world applications, e.g., in network analysis [For09], [PDFV05], bioinformatics [DKR⁺13], [BW06], circuit design [LMA89], and telecommunication [DSAA14]. Figure 2.1 depicts a maximum clique in a graph.

Definition 2.2.1 (Maximum Clique Problem) *Given a graph $G = (V, E)$, the MCP is the problem of finding a clique of maximum size in G .*

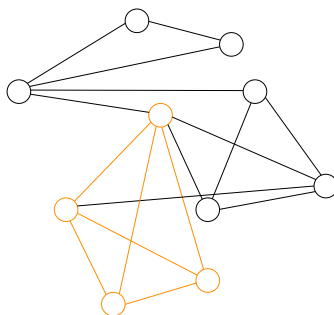


Figure 2.1: A maximum clique of size four.

2.2.1 Relaxations of the MCP

For some real-world applications, the MCP is too strict a model, as some applications require identifying large, dense subgraphs, but not necessarily fully connected subgraphs. Furthermore, it might be the case that acquiring real-world data is an error-prone process, thus requiring relaxations of the clique model tailored to the specific application. For these reasons, several relaxations of the maximum clique problem have been introduced: The Maximum γ -Quasi-Clique Problem [ARS02] and the Maximum s -Defective Clique Problem [YPTG06] are density-based and edge-based relaxations, respectively, where a solution S is a set of vertices with a given minimum density γ , or has at most a given number of s edges missing in $G[S]$. The Maximum s -plex Problem is a degree-based relaxation [SF78], where each vertex in a solution S is required to have at least $|S| - s$ neighbors in S , and the Maximum s -club Problem [Mok79] is a path-based relaxation, where the induced subgraph $G[S]$ must have a diameter of at most s . Other relaxations include s -blocks, s -bundles and s -cores [GIFC15], among others.

2.2.2 Properties of the MCP

In graph theory, a property P of a graph G is *hereditary* if P also holds for all induced subgraphs of G [PVBB13]. It is easy to see that the property of a graph being a clique is hereditary, as each subset of a clique is a clique itself. Heredity is an important concept in the context of graph theory, as it allows the development of algorithms that exploit the structure of solutions exhibited by heredity, e.g. [TBBB13] and [GIP18].

There exist negative results about the hardness of approximating the maximum clique in a graph in polynomial time: In [Hås99] and [Zuc06] the authors show that there exists no fully polynomial time approximation scheme for the MCP, unless $P = NP$, meaning that for a real number $\varepsilon > 0$ there exists no polynomial time algorithm that approximates the maximum clique in a graph with a factor of at least $O(n^{1-\varepsilon})$, where n is the number of vertices in the input graph. More importantly, these hardness results are trivially also valid for relaxations of the MCP. Considering the hardness of approximability of the MCP and its relaxations, it is evident that heuristic approaches are of great importance when tackling large real-world instances that cannot be solved optimally in practice.

2.3 The Maximum Quasi-Clique Problem

The MQCP is a density-based relaxation of the MCP, which was first introduced by Abello et al. [ARS02]. Pattillo et al. show that for any γ with $0 < \gamma < 1$, the problem is NP-complete in its decision variant [PVBB13].

Definition 2.3.1 (Maximum Quasi-Clique Problem) *Given a graph $G = (V, E)$ and $\gamma \in (0, 1]$, the MQCP is the problem of finding a subset of vertices $S \subseteq V$ of maximum size such that the induced subgraph $G[S]$ has an edge density of at least γ , or, in other words, $G[S]$ contains at least $\gamma \binom{|S|}{2}$ edges.*

2.3.1 Properties of the MQCP

In contrast to cliques, the property of being a quasi-clique is not hereditary. Algorithms that build upon heredity can therefore not easily be adapted to the MQCP. However, as noted in [PVBB13], quasi-cliques display a property that the authors call *quasi-heredity*. A property P of a graph $G = (V, E)$ is *quasi-hereditary*, if there exists some $v \in V$ such that $G[V \setminus \{v\}]$ also has property P . Furthermore, the authors show that quasi-cliques are quasi-hereditary. Therefore, for a γ -quasi-clique of size k there exists a series of γ -quasi cliques of size $1, 2, \dots, k$ such that each γ -quasi clique is a strict subset of the next one. Most of the leading heuristic algorithms for the MQCP (e.g., [DHB19], [ZBW20], [CCP+21]) build upon this important property by searching for a sequence of γ -quasi cliques of increasing size in order to approximate the maximum γ -quasi clique in a graph.

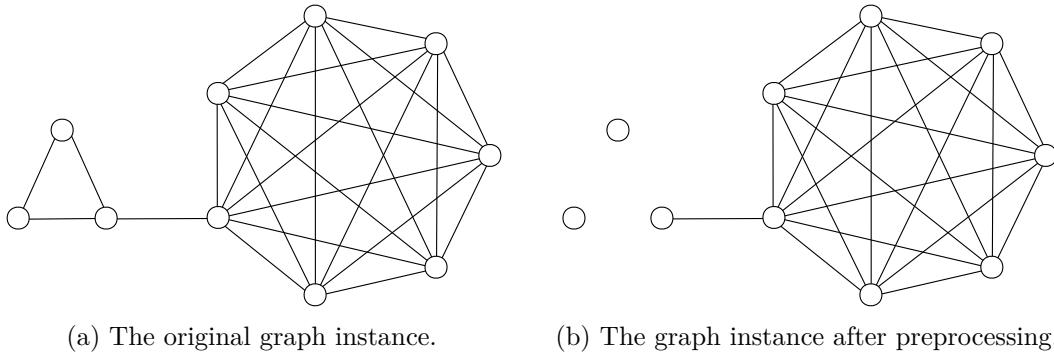
2.3.2 Preprocessing of MQCP Instances

In [ARS02], the authors introduce the notion of γk -peelable vertices in the context of preprocessing. A vertex v is γk -peelable if v and all its neighbors have degrees smaller than γk . If a lower bound k for the size of a maximum γ -quasi clique is known, the authors propose to remove all edges incident to γk -peelable vertices from G in preprocessing. This preprocessing rule might be used to speed up the search process by sparsifying the graph, but we note that this preprocessing rule does not necessarily preserve the existence of optimal solutions. We prove this claim by giving a counterexample in Figure 2.2, where the optimal solution cannot be obtained after applying said preprocessing rule.

2.4 The Maximum Defective Clique Problem

The MDCP was first introduced in [YPTG06] in the context of predicting protein-protein interactions.

Definition 2.4.1 (Maximum s -defective-Clique Problem) *Given a graph $G = (V, E)$ and integer s , the Maximum s -defective Clique Problem (MDCP) is the problem of finding a subset of vertices $S \subseteq V$ of maximum size such that the induced subgraph $G[S]$ contains at least $\binom{|S|}{2} - s$ edges.*



(a) The original graph instance.

(b) The graph instance after preprocessing.

Figure 2.2: Consider the graph G given in Subfigure 2.2a. The considered MQCP instance is defined by G consisting of a K_3 and a K_7 connected by an edge and $\gamma = 0.5$. Since G has a density $\text{dens}(G) = \frac{25}{45} \approx 0.56$, the maximum 0.5-quasi clique in G is V . Assume a lower bound of $k = 6$ is known, therefore the two vertices on the left are $k\gamma$ -peelable and preprocessing would remove all edges incident to these vertices, as can be seen in Subfigure 2.2b. Therefore, the density of this preprocessed graph would be reduced to $\frac{22}{45} \approx 0.49$ and the optimal solution V would not be preserved.

2.4.1 Properties of the MDCP

Although the MQCP and the MDCP are similar relaxations of the MCP, there are some key differences between the problems. It can easily be seen that the property of being an s -defective clique is hereditary: If S is an s -defective clique with at most s edges missing in $G[S]$, the removal of a vertex $v \in S$ cannot “introduce” any new missing edges. However, for a fixed size k , the two problems become irreducible: Finding a k - s -defective clique is equivalent to finding a k - γ -quasi clique with $\gamma = \frac{\binom{k}{2} - s}{\binom{k}{2}}$, and finding a k - γ -quasi clique is equivalent to finding a k - s -defective clique, where $s = \binom{k}{2} - \lceil \gamma \binom{k}{2} \rceil$. When approximating the maximum s -defective clique in a graph G by a series of s -defective cliques of increasing size, the problem can therefore be reduced to finding a series of γ -quasi cliques and vice-versa.

Related Work

Utilizing ML techniques in combinatorial optimization has been a research field of growing interest in recent years, and especially the application of GNNs in the context of combinatorial optimization is spreading in popularity. In this Chapter, we provide a concise review of methods and work related to the topics of this thesis, namely ML methods and GNNs in combinatorial optimization, and also work related to the considered problems, the MCP, the MQCP, and the MDCP. Section 3.1 provides an introduction to the necessary concepts related to GNNs that are applied in this work. Afterward, we review the application of ML techniques that are relevant or related to this work in Section 3.2. We present methods to generate node features in feature-less graphs, which we apply in our algorithm in Chapter 5, in Section 3.3. Furthermore, we provide an overview of existing heuristic and exact solution approaches for the MQCP in Section 3.4 and for the MDCP in Section 3.5.

3.1 Graph Neural Networks

Problems that are defined on graphs, e.g. the Traveling Salesperson Problem (TSP), pose new challenges in utilizing machine learning effectively: Utilized methods should be able to effectively capture and exploit the structure of input graphs while taking into account that graphs do not have a unique representation and no fixed order of vertices, i.e., they should be *order invariant*. Furthermore, graphs of all sizes should be considered as input, therefore utilized methods need to be *scale invariant*. Ideally, the results of the training should generalize on unseen instances that might also be of a different size than the instances seen during training. In order to address these challenges, researchers have come up with machine learning architectures known as GNNs.

The authors of [SS97] were the first to research the application of neural networks on graphs. Since then, many researchers have advanced the field by developing new

architectures and methods. A comprehensive overview of the taxonomy of GNNs and a review of applications and methods can be found in [WPC⁺19] and [ZCH⁺20].

In [WPC⁺19], GNNs are classified into four main categories: Recurrent GNNs, Convolutional GNNs (ConvGNNs), Graph autoencoders, and Spatial-temporal graph neural networks. In the following, we conform to the authors' definitions of GNN-related concepts. ConvGNNs can further be categorized as spectral-based and spatial-based GNNs. Spectral-based GNNs are based on signal processing theory, and a graph is in this context thus seen as a signal which is transformed to a spectral domain by a graph Fourier transform. The convolution operation is then conducted in the spectral domain, and afterward, the graph signal is transformed back using the inverse graph Fourier transform. In contrast, spatial-based ConvGNNs apply convolution operations directly on the input graph based on its topology. In the context of our work, spatial-based ConvGNNs are the most relevant.

The Message Passing Neural Network (MPNN) [GSR⁺17] is a framework of spatial-based ConvGNNs, which captures several different concrete realizations due to its generality. Here, information corresponding to vertex v in a graph is represented by a feature vector x_v , and convolution operations are defined to aggregate information of the neighboring vertices by passing messages over the edges of the graph. A convolutional layer then aggregates the features of neighboring vertices for all vertices in the graph. The general idea of this convolution operation is sketched in Figure 3.1, where the updated value for vertex v_0 is computed by aggregating the messages passed from its neighbors. MPNN uses a fixed number of layers to compute the final node embeddings. More precisely, to update the representation $h_v^{(k)}$ of a node v in layer k , the following message passing function is applied:

$$h_v^{(k)} = U_k(h_v^{(k-1)}, \sum_{u \in N_G(v)} M_k(h_v^{(k-1)}, h_u^{(k-1)}, x_{vu}^e))$$

Here, $U_k(\cdot), M_k(\cdot)$ are functions with learnable parameters, $h_v^{(0)} = x_v$, and x_{vu}^e is the feature vector of the (directed) edge from v to u . Note that edge features are however not required, as also our work in this thesis utilizes spatial-based ConvGNNs on graphs that only contain information about vertices, but not about edges. The functions U_k, M_k can be realized by any learnable function and thus cover many concrete realizations of the concept, for example, M_k can be a Multi-Layer Perceptron (MLP) that takes the concatenation of $(h_v^{(k-1)}, h_u^{(k-1)}, x_{vu}^e)$ as inputs to compute its message, and U_k is often realized by a ReLU activation function, followed by a batch-normalization [IS15] layer with learnable parameters.

In recent years, researchers have developed attention-based spatial ConvGNNs (e.g., [VCC⁺18], [ZSX⁺18], [BAY21]) in order to translate the popular attention mechanism, which was originally introduced in the context of sequence-based tasks ([BCB15], [VSP⁺17]), to graph-based tasks. In contrast to the basic spatial-based convolution operator discussed before, attention-based convolution operators assign different weights

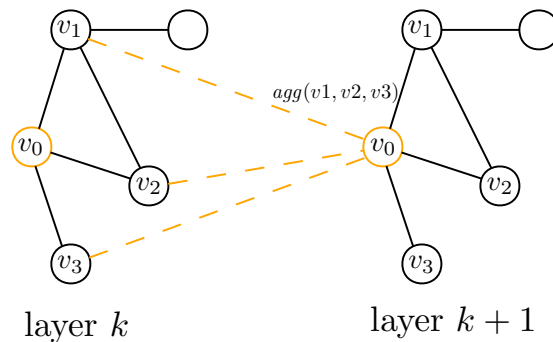


Figure 3.1: Visual representation of the convolution operation for a vertex in a graph. Information of adjacent vertices is aggregated to compute the updated representation.

for different neighbors in an attempt to filter noise and focus the “attention” only on the most relevant information. Most prominently, the graph attention network (GAT) proposed in [VCC⁺18] computes the state of a node v by the following update function:

$$h_v^{(k)} = \rho\left(\sum_{u \in N_G[v]} \alpha_{vu} W^{(k)} h_u^{(k-1)}\right),$$

$$\alpha_{vu} = \frac{\exp(\text{LeakyReLU}(a^T [W^{(k)} h_v^{(k-1)} \parallel W^{(k)} h_u^{(k-1)}]))}{\sum_{w \in N_G[v]} \exp(\text{LeakyReLU}(a^T [W^{(k)} h_v^{(k-1)} \parallel W^{(k)} h_w^{(k-1)}]))},$$

where W is a weight matrix of learnable parameters, a is the weight vector of a single-layer perceptron, and ρ is a non-linear function. In [BAY21], the authors present another variant named GATv2, in which they slightly change the update function of a layer by changing the attention coefficients:

$$\alpha_{vu} = \frac{1}{z_v} \exp(a^T \text{LeakyReLU}(W^{(k)} h_v^{(k-1)} \parallel W h_u^{(k-1)})),$$

where z_v is the normalization factor for node v . According to the authors, this simple modification makes a significant difference, and they validate their claim by showing that there exist problems, where GATv2 greatly outperforms the previous GAT layers. Another important property of the GAT architecture is the utilization of multi-head attention proposed in [VSP⁺17]. Here, a fixed number K of multiple independent attention heads are applied and then their outputs are either concatenated or the element-wise mean is computed.

GNNs can be used for different tasks such as node-level tasks, e.g. the classification of nodes in a graph, edge-level tasks, e.g. edge classification or link prediction, or graph-level tasks, e.g. graph classification, both in supervised and unsupervised learning settings.

3.2 Machine Learning in Combinatorial Optimization

In many exact and heuristic algorithms for COPs, ML techniques are a determining factor in the effectiveness of said algorithms. There are many ways ML can be applied: It can be used in an end-to-end fashion to directly produce a solution, to replace or enhance key components of existing algorithms, e.g., by obtaining quick approximations for otherwise computationally heavy parts of an algorithm, to provide additional information to heuristics or metaheuristic, or to make decisions within an algorithm, e.g. variable selection in branch-and-bound algorithms. A driving motivation in utilizing ML in COPs thus is to (partially) automate the process of hand-crafting heuristic algorithm components.

Most of the current ML-based approaches are based on supervised learning, as stated in [CCK⁺21]. This often requires labeled training data in the form of optimal solutions to computationally hard problems, which is a great restriction. The two most common approaches to tackle this challenge are to either limit the training data to only incorporate labeled data of small instances that can be solved optimally within reasonable time or to use high-quality approximations, and thus decrease the solution quality. The former approach additionally poses the challenge to develop methods that translate well to unseen and possibly larger real-world instances that are relevant in practice. A prominent example of this application is [LCK18], where a GNN is trained to predict whether a vertex in the graph is in an optimal solution. A tree search algorithm is then executed that uses the information provided by the GNN. The approach generalizes well to instances that are much larger than those seen during training, and the authors apply it to multiple different COPs.

Other approaches are based on unsupervised reinforcement learning and do not require labeled training data. In [DKZ⁺17], the authors propose an end-to-end reinforcement learning framework, S2V-DQN, which generalizes well to a wide range of problems. Furthermore, recent advances in this field have also been made by researchers of combinatorial games. Especially AlphaGo Zero is a prominent example, which was used to train models that excel in the board games Chess and Go via repeated self-play. AlphaGo Zero was later adapted in [AXSS19] and applied to several combinatorial optimization problems, including Minimum Vertex Cover and Maximum Cut. In [KvHW19] and [JCRL21], the authors present end-to-end ML approaches by utilizing GNNs for routing problems including the Euclidean TSP. While their work shows promise, the proposed methods are only available for relatively small graphs, where exact approaches are available. In [HLMP21], the authors present a guided local search approach, where a GNN is used to provide additional information to enhance the metaheuristic search, leading to improved results when applied to the TSP. Again, only relatively small instances are considered. We refer to [CCK⁺21] for a comprehensive overview of GNNs in the context of combinatorial optimization for both heuristic and exact algorithms.

In the context of the MCP, several ML-based approaches have been made. In [RGR⁺22], the authors build upon an exact branch-and-bound algorithm for the MCP, that uses

an empirically determined algorithm parameter that greatly affects the algorithm’s performance. The authors thus enhance the algorithm by training a GNN model to predict suitable parameter settings for each input graph. In [SLE21], ML is applied to reduce MCP instances by training models to predict, whether a vertex is in an optimal solution or not. The input graph is then reduced, and an exact algorithm is applied to the reduced instance. Furthermore, in [JDHB22], the authors propose a metaheuristic breakout local search for the Maximum k -plex Problem (a degree-based MCP relaxation) that uses reinforcement learning in the diversification stage of the algorithm to predict good parameter settings for the diversification. Finally, a Pointer-Network based deep learning algorithm for the MCP is presented in [GY20]. To the best of our knowledge, there are no ML-based or GNN-based approaches to the MQCP or the MDCP.

3.3 Node Representation Learning

As with other ML methods, the choice of input features can affect the performance greatly when utilizing GNNs. Some problems like the 2D Euclidean TSP are defined on graphs, where each vertex of the input graph is embedded in a 2D plane. It is therefore natural to use the 2D coordinates of each vertex as input features in any GNN-based task related to this problem. However, many problems and tasks utilizing GNNs are defined on inherently featureless, or non-attributed input graphs. To tackle this challenge, researchers have come up with two different approaches regarding feature initialization. The first approach is to use centrality-based input features that can be derived directly from the graph, e.g. the degree of a vertex in the graph. Secondly, researchers have come up with learning-based approaches that try to extract input features directly from the graph structure. In [DHD⁺19] the authors investigate both centrality-based and learning-based feature initialization methods and evaluate the quality of the produced feature embeddings in a node-level classification task on several well-established benchmark sets. The results of the authors’ experiments show that learning-based approaches outperform centrality-based feature initialization methods.

One of the most popular approaches in node representation learning is based on ideas from natural language processing (NLP) and word representation learning. In [MCCD13], the authors propose their method word2vec, which was developed to embed words into an embedding space such that words with a similar meaning are mapped closer together. While in other NLP tasks it is common to predict the next word in a sequence from a context, word2vec turns the problem on its head and tries to predict the context from a word, where the context is defined as the words that appear at a certain distance before and after a word. In [PAS14], the authors present DeepWalk, which translates this idea to node representation learning by sampling random walks of a fixed length from a graph and by considering the vertices of the graph as a vocabulary and the random walks as sentences over this vocabulary. Here, a random walk starts at some vertex, and at each step of the random walk, a random neighbor that can be reached from the current vertex is chosen as the successor, and all neighbors of a node are equally likely to be chosen. Multiple random walks are sampled, starting from each vertex of the graph. The node

embeddings are initialized randomly. Each random walk is then used as a sentence to update the embeddings using the SkipGram language model, as in word2vec. This is done by maximizing for each vertex appearing in a random walk the probability of predicting the vertices that appear at a fixed distance before and after said vertex. The authors use stochastic gradient descent to maximize the said probability and show the success of their proposed method by evaluating the learned representations in a multi-label classification task.

Node2Vec [GL16] builds upon DeepWalk and introduces another method to sample random walks from graphs, which the authors call second-order random walks. Here, two hyperparameters p, q are introduced to provide additional control over the exploration of the graph during random walks. Assuming a random walk just traversed an edge (t, v) , a successor is then chosen among the neighbors of v by assigning each neighbor x of v an unnormalized transition probability $\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$, defined as

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases},$$

where w_{vx} is the edge weight of edge (v, x) and d_{tx} is the distance of the shortest path between t and x (a small example is depicted in Figure 3.2). The next vertex in the random walk is chosen by selecting a vertex from the neighborhood of v by weighted sampling according to the normalized transition probabilities. The parameter p then controls the probability of returning to the most recently visited vertex, whereas the parameter q controls the probability of either staying close to a vertex t or exploring regions of the graph at a greater distance from t . The authors show that different parameterizations can be used to capture either homophily or structural equivalence in the produced embeddings.

While Node2Vec can capture some structural similarities between vertices in the input graph, it only does so if vertices are at a certain maximum distance from each other, as the random walks traverse the edges of the original graph. Struc2Vec [FRS17] is a method for learning node representations from structural identity alone, not considering the locality. This is done by creating a multi-layer graph, where each layer is composed of a complete graph containing the vertices of the original graph, but edges between the vertices are weighted by their structural similarity. For a vertex v , let $R_k(v)$ be the set of vertices at distance k from v , but cannot be reached by a path of length $< k$ from v . The authors define the structural similarity of two vertices u, v in layer k in the multi-layer graph as the similarity between the ordered degree sequences of the vertex sets $R_k(u), R_k(v)$. Let A, B be the ordered degree sequences of the vertex sets $R_k(u), R_k(v)$. The authors define the distance between two elements $a \in A, b \in B$ in the sequences as

$$d(a, b) = \frac{\max(a, b)}{\min(a, b)} - 1.$$

The similarity between two degree sequences is then computed by dynamic time warping, which is a method used to find the optimal alignment between two sequences that

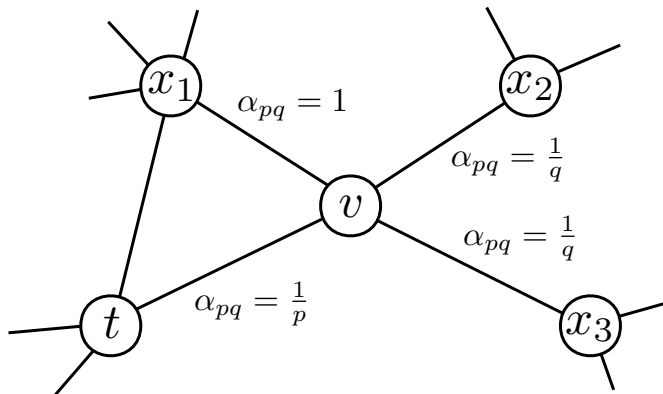


Figure 3.2: Illustration of the definition of unnormalized transition probabilities for a random walk that just traversed the edge (t, v) and is computing the transition probabilities for its successor. Figure recreated from [GL16].

minimizes the proposed distance function. The edge weights of layer k in the multi-layer graph are therefore set to be the result of this computation. Random walks are then sampled from the constructed multi-layer graph. Before each step, the walk can decide to go one layer up or down the multi-layer graph. A neighbor on the current layer is then chosen by weighted sampling, considering the edge weights of the current layer. The authors show that the proposed method of sampling random walks from a multi-layer graph can capture structural similarities despite vertices being far apart in the input graph.

Other notable approaches in node representation learning include HOPE [OCP⁺16], which was developed specifically to preserve properties of directed graphs and DeepGL [RZA17], which is a deep learning framework for node representation learning that is designed such that learned embeddings transfer well across different networks. Beyond the scope of node representation learning much research has also been done in other areas of network representation learning, e.g. edge representation learning and graph representation learning. In our work, we chose to consider and evaluate primarily Node2Vec and Struc2Vec, as these methods are simple to implement and seem the most suited to our application, as they are able to capture local and structural features for non-attributed graphs.

3.4 The Maximum Quasi-Clique Problem

Since the introduction of the problem in [ARS02], researchers have proposed several exact and heuristic algorithms for the MQCP. Especially in recent years, there has been an increasing interest in the problem. In the following subsections, we review and discuss exact and heuristic solution approaches for the MQCP.

3.4.1 Exact Approaches

Pattillo et al. [PVBB13] propose a Mixed Integer Linear Programming formulation with a number of variables that is quadratic in the size of the vertex set V . It is derived from the following quadratic formulation, where $x_i \in \{0, 1\}$ are binary decision variables representing the solution set $S \subseteq V$, and a_{ij} represent elements of the adjacency matrix with a_{ij} equal to one if $\{i, j\} \in E$ and zero otherwise:

$$\max \sum_{i=1}^n x_i \tag{3.1}$$

$$\text{s.t. } \sum_{i=1}^n \sum_{j=i+1}^n a_{ij} x_i x_j \geq \gamma \sum_{i=1}^n \sum_{j=i+1}^n x_i x_j \tag{3.2}$$

The authors linearize the above quadratic formulation by introducing variables w_{ij} defined as $w_{ij} = x_i x_j$ in their MILP-formulation named F1:

$$\max \sum_{i=1}^n x_i \tag{3.3}$$

$$\text{s.t. } \sum_{i=1}^n \sum_{j=i+1}^n (\gamma - a_{ij}) w_{ij} \leq 0 \tag{3.4}$$

$$w_{ij} \leq x_i \quad \forall \{i, j\} \in E \tag{3.5}$$

$$w_{ij} \leq x_j \quad \forall \{i, j\} \in E \tag{3.6}$$

$$w_{ij} \geq x_i + x_j - 1 \quad \forall \{i, j\} \in E \tag{3.7}$$

$$w_{ij} \geq 0 \quad \forall \{i, j\} \in E \tag{3.8}$$

$$x_i \in \{0, 1\} \quad i \in V \tag{3.9}$$

Furthermore, the authors propose a second MILP-formulation F2 that has a number of variables that is linear in n , but their computational experiments show that the above formulation performs better in practice on most of the evaluated instances.

In [VPBP16], the authors propose two additional MILP-formulation. In one of their proposed MILP-models which is named F3, they utilize an upper and lower bound to provide a tighter formulation than the ones from [PVBB13]. As a lower bound ω^ℓ , the size of any known γ -quasi clique can be used, whereas the upper bound is defined as $\omega^u = \lfloor \frac{1}{2} + \frac{1}{2} \sqrt{1 + 8 \frac{|E|}{\gamma}} \rfloor$. The proposed formulation F3 is then defined as:

$$\max \sum_{i \in V} x_i \quad (3.10)$$

$$\text{s.t.} \quad \sum_{(i,j) \in E} y_{ij} \geq \gamma \sum_{k=\omega^\ell}^{\omega^u} \frac{k(k-1)}{2} z_k \quad (3.11)$$

$$y_{ij} \leq x_i \quad \forall (i,j) \in E \quad (3.12)$$

$$y_{ij} \leq x_j \quad \forall (i,j) \in E \quad (3.13)$$

$$\sum_{i \in V} x_i = \sum_{k=\omega^\ell}^{\omega^u} k z_k \quad (3.14)$$

$$\sum_{k=\omega^\ell}^{\omega^u} z_k = 1 \quad (3.15)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (3.16)$$

$$y_{ij} \geq 0 \quad \forall i, j \in V, i < j \quad (3.17)$$

$$z_k \geq 0 \quad \forall k \in \{\omega^\ell, \dots, \omega^u\} \quad (3.18)$$

Compared to the formulation F1 in [PVBB13], the authors introduce a variable z_k , that is used to determine the size of the solution S , i.e. $z_k = 1$ iff $|S| = k$. Clearly, providing tight upper and lower bounds ω^ℓ, ω^u has a great effect on the performance of this model. The authors evaluate their formulations and show both theoretically and empirically that they outperform the previously mentioned formulations F1 and F2, and that the formulations are especially effective on sparse graphs. Dense graphs with $|V| > 100$ however are infeasible to solve in practice using any of the proposed formulations.

In [RR19], the authors propose a branch-and-bound algorithm that can outperform the previously mentioned MILP-formulations on dense graphs of sizes of up to 100 vertices. Furthermore, they propose a new method to calculate upper bounds for the size of a maximum-quasi clique, which provides much tighter upper bounds than previously known methods. A MILP-formulation with an exponential number of variables is proposed in [MPR21]. The authors also present a branch-and-price algorithm that can be used for column generation in the context of their presented formulation.

Despite the advances in the research of exact methods regarding the MQCP, the problem proves to be extremely computationally challenging. In practice, only graphs that are sufficiently small or sparse are feasible to solve using any of the discussed exact methods.

3.4.2 Heuristic Approaches

As already discussed, large and dense MQCP instances are infeasible to solve in practice due to the computational complexity of the problem. Researchers have therefore come up with several fast heuristic solution approaches that yield in practice often good, or

even near-to-optimal solutions that can be used on large real-world instances. Early heuristic work on the MQCP dates back to 2002 [ARS02], where the authors are the first to formally define the problem and propose a greedy randomized adaptive search procedure (GRASP) as a first heuristic solution approach. A GRASP iteration typically consists of two phases - an initial candidate solution is given by a greedy randomized construction heuristic, followed by improvements through local search. In [BHB08], the authors present two stochastic algorithms based on the local search paradigm. A distributed algorithm for mining quasi-cliques in large graphs based on the MapReduce programming model is presented in [KS11], and in [OPR13] the authors present a restart iterative greedy algorithm named RIG.

In recent years, several heuristic solution approaches have been presented. In [PPRR15], [PRRP18], and [PRRR21] the authors present variations of a genetic algorithm for the MQCP. The authors' latest work on the MQCP, [PRRR21], shows the best performance out of the three variations. It is based on the hybridization of their previously proposed genetic algorithm and a local search strategy.

A multi-start tabu search algorithm is presented in [DHB19]. It builds upon an adaptive multi-start tabu search algorithm that was initially proposed for the MCP. The authors offer an analysis of the considered neighborhood structures used during the tabu search and adapt the intensification and diversification mechanisms of this algorithm to be suitable for the MQCP. The results of the computational experiments by the authors show that the performance of their tabu search algorithm is highly competitive to the state-of-the-art and performs well on established benchmark instances.

Furthermore, in [ZBW20], an opposition-based memetic algorithm for the MQCP is presented. The authors combine the ideas of using a genetic algorithm to build a population of candidate solutions with a tabu search that is used to improve these solutions. Their main contribution is the incorporation of an opposition-based construction heuristic. Every time an offspring, which corresponds to a candidate solution S , is created by means of a recombination operator, another offspring \bar{S} is created that does not share any vertices with S . Both opposing individuals S, \bar{S} are then improved by a tabu search procedure, and the individual that leads to a better solution through tabu search is then added to the population. The authors compare the performance of their algorithm to RIG [OPR13], and three variations of BRKGA [PPRR15],[PRRP18], [PRRR21] and show that their algorithm outperforms these approaches.

In [CCP⁺21], another local search-based algorithm is presented. The authors make two main contributions to existing local search-based approaches: Firstly, they identify weaknesses in the scoring function used to determine, which vertices should be used during iterations of the local search procedure. Their computational experiments show that in most iterations, there are several vertices with the same score, thus they provide a secondary scoring function that serves as a tiebreaker. The value of this secondary scoring function is proportional to the number of iterations that a vertex was not moved during the local search, thus it strongly encourages exploration of the search space. Secondly, they replace the commonly used tabu list by configuration checking,

which is another short-term memory type that can be used to prevent cycling through the search space. The authors present their own variation of configuration checking, which they call BoundedCC. In general, configuration checking is a short-term memory mechanism that blocks a vertex v from being considered in local search until one of its (unblocked) neighbors is moved. BoundedCC refines this strategy by keeping track of the values $confChange(v)$, $threshold(v)$, and it is parameterized by a user-defined value $ub_threshold \in \mathbb{N}$. A vertex is blocked, if $confChange(v) < threshold(v)$ holds. Three update rules are defined as follows:

- *BoundedCC-InitialRule*. Initially, for all $v \in V$ set $confChange(v) = 1$, $threshold(v) = 1$.
- *BoundedCC-AddRule*. When v is added into the candidate solution, for all $u \in N_G(v)$, $confChange(u)$ is increased by one and $threshold(v)$ is increased by one as well. If $threshold(v) > ub_threshold$, $threshold(v)$ will be reset to 1.
- *BoundedCC-RemoveRule*. When v is removed from the candidate solution, set $confChange(v) = 0$.

The authors compare their algorithm to those presented in [PRRR21], [DHB19], and [ZBW20] and show that their algorithm outperforms similar algorithms in terms of solution quality and runtime.

Finally, the most recent heuristic solution approach for the MQCP at the time of writing this thesis is presented in [PWWW21]. Here, the authors propose a hybrid artificial bee colony algorithm, which incorporates an opposition-based construction phase and then adapts the artificial bee colony framework to the MQCP. The algorithm again uses tabu search to improve initially constructed candidate solutions. The authors report new best results on several instances that were not included in the evaluation of [CCP⁺21]. A comparison is made only with RIG [OPR13], and [PPRR15], [PRRP18], and [PRRR21], which are outperformed on other instances by [CCP⁺21] and [ZBW20].

As we have extensively studied the literature regarding heuristic solution approaches, we found that most state-of-the-art methods are based on or utilize metaheuristics based on the local search paradigm. Therefore, we let previous work on the MQCP lead our way and plan to build our work upon the local search paradigm as well.

3.5 The Maximum Defective Clique Problem

Compared to the MQCP, only a few solution approaches have been proposed for the MDCP. The first algorithm is proposed in [YPTG06], along with the first formal definition of the problem. The authors develop their approach for the use case of link prediction for protein-protein interactions. Even before that, the problem has been considered in the context of transportation planning [SST02], where MILP formulations of the problem

are analyzed and discussed. Note that, similar to the MQCP, there is a straight-forward MILP-formulation as presented in [SCS06]:

$$\max \sum_{i \in V} x_i \tag{3.19}$$

$$\text{s.t. } \sum_{\{i,j\} \in \bar{E}} z_{ij} \leq k \tag{3.20}$$

$$z_{ij} \geq x_i + x_j - 1 \quad \forall \{i, j\} \in \bar{E} \tag{3.21}$$

$$z_{ij} \geq 0 \quad \forall \{i, j\} \in \bar{E} \tag{3.22}$$

$$x_i \in \{0, 1\} \quad \forall i \in V \tag{3.23}$$

Here, \bar{E} denotes the set $\{\{i, j\} \mid \{i, j\} \notin E\}$, i.e. the set of edges that are not in E .

In [TBBB13], the authors analyze several MCP relaxations from the perspective of hereditary properties in graphs. They show that the property of being a defective clique is a non-trivial, interesting property that is hereditary on induced subgraphs, and that the NP-hardness of the problem follows from this observation. Furthermore, the authors propose an exact algorithm based on Russian Doll Search (RDS) that can be used to solve MCP relaxations that are hereditary. In [GIP18], the RDS algorithm is revisited and improved by new pre-processing procedures. More recently, exact solution approaches are proposed in [CZHX21] and [GXLY22] specifically to exploit massive sparse graphs, and a heuristic first-order optimization method is presented in [BRZ22].

When studying the literature related to the MDCP, we notice that there are only a few heuristic approaches. Due to the close relation of the MQCP and the MDCP we focus our approach to build upon existing heuristic algorithms for the MQCP and show that it can be adapted to the MDCP easily.

A Local Search Based Metaheuristic for Edge-Based Relaxations of the Maximum Clique Problem

In this chapter we present a local search based metaheuristic, named LSBM, for relaxations of the MCP in detail, which is the algorithmic foundation of the GNN-based algorithm presented in Chapter 5. At first, we introduce the general structure of LSBM without using GNNs in Section 4.1, followed by a detailed inspection of the components of our algorithm. We propose a lower bound heuristic based on beam search (BS) in Section 4.2 that is used to efficiently obtain a feasible initial solution of high quality. Next, we discuss the construction heuristic used within LSBM in Section 4.3. This construction heuristic does not necessarily yield a feasible solution but is used to explore new regions of the search space not visited yet, as it will be used as a starting point for the local search procedure. The main move operator used in the local search procedure is defined in Section 4.4. Moreover, we define and analyze the corresponding neighborhood structures. Thereafter, we show in detail the local search procedure of LSBM that searches said neighborhood structures and uses a scoring function to evaluate the vertices in the graph to restrict the neighborhood to the most promising moves. We introduce such a scoring function, namely the function d_S , that denotes for a vertex in the input graph the number of neighbors in the current candidate solution S . This scoring function is commonly used in state-of-the-art heuristic approaches for the MQCP, but we also discuss that any other scoring function can be used within LSBM.

Note that we present our algorithm in the context of the MQCP. As the MQCP and the MDCP are closely related, the adaption to the MDCP is straight-forward and the

necessary steps to adapt LSBM to the MDCP are shown at the end of the Chapter in Section 4.5.

4.1 Algorithm Structure

As noted by Friden et al. [FHdW89], a solution for the MCP can be obtained by finding a series of k -cliques for increasing values of k . Based on this observation, Wu et al. [WH13] develop an adaptive multi-start tabu search algorithm for the MCP that in each iteration searches for a clique of size k using a tabu search procedure. If such a clique can be found, k is increased and the algorithm is restarted. The largest clique found this way is returned as the solution. Djeddi et al. [DHB19] show that this idea can be extended to the MQCP by following the same principle: The maximum γ -quasi clique in a graph is approximated by finding a series of k - γ -quasi cliques, where the tabu search is restarted with increased k each time a k - γ -quasi clique is found. This idea is applied successfully also in other state-of-the-art algorithms for the MQCP, namely, [ZBW20] and [CCP⁺21], which is why we want to base the general structure of our algorithm on this idea as well.

The pseudocode of the general structure of our local search algorithm is shown in Algorithm 4.1. *InitialSolution*, which is a heuristic algorithm based on BS, constructs a feasible initial solution. Its main purpose is to quickly obtain a reasonable lower bound that can be used as a starting value for k . The procedure is described in detail in Section 4.2.

In Line 3 a long-term memory *Freq* is initialized, where *Freq*[i] stores how many times each vertex is operated on for $i = 1, \dots, n$. This memory is kept when the loop in Lines 4–12 is repeated, but no feasible solution could be found. In *ConstructionHeuristic*, *Freq* is used to produce a candidate solution that favors vertices that have not been considered yet. This way, new regions of the search space are explored. The idea of using this long-term memory *Freq* in the construction heuristic to cover a larger area of the search space comes from [CCP⁺21]. We change their construction heuristic slightly by combining it with a GRASP-like construction step as described in Section 4.3.

Once a candidate solution is obtained by a construction heuristic, we try to improve it by the local search procedure in Line 6, which is described in detail in Section 4.4 along with the definition of the neighborhood structures. During this local search procedure, we use the scoring function f_S in each iteration to restrict the neighborhood of a candidate solution S to the most promising moves in order to increase its objective value. If the resulting solution S is feasible, a simple search is performed in *Extend* to check if the solution can be extended to a feasible solution of greater size as shown in Algorithm 4.2. Furthermore, k is increased, *Freq* is reset, and the best-so-far found solution S^* is updated. If the solution obtained by the local search procedure is not feasible, the while-loop is repeated until the stopping criterion is met. We define two user-defined parameters as the stopping criterion: a time limit τ and a maximum number of restarts ξ . Whenever one of the two stopping criteria is fulfilled, the execution of the algorithm is terminated.

Algorithm 4.1: General structure of LSBM

Input: Graph G , Target Density γ
Output: Approximate maximum γ -quasi clique in G , scoring function f_S

- 1 $S^* \leftarrow \text{InitialSolution}(G, \gamma)$
- 2 $k \leftarrow |S^*|$
- 3 $\text{Freq} \leftarrow [0, \dots, 0]$ // n -element Array, initialized with 0
- 4 **while** stop condition not met **do**
- 5 $S \leftarrow \text{ConstructionHeuristic}(G, \text{Freq}, k)$
- 6 $S \leftarrow \text{LocalSearch}(G, \text{Freq}, \gamma, S, f_S)$
- 7 **if** S is feasible **then**
- 8 $S \leftarrow \text{Extend}(G, S, \gamma)$
- 9 $S^* \leftarrow S$
- 10 $k \leftarrow |S| + 1$
- 11 $\text{Freq} \leftarrow [0, \dots, 0]$
- 12 **end**
- 13 **end**
- 14 **return** S^*

Algorithm 4.2: Extend a feasible solution

Input: Graph G , Feasible solution S , Target Density γ
Output: Feasible solution S' of size at least $|S|$

- 1 $S' \leftarrow S$
- 2 **while** true **do**
- 3 **for** $u \in V \setminus S'$ **do**
- 4 **if** $S' \cup \{u\}$ is a γ -Quasi Clique **then**
- 5 $S' \leftarrow S' \cup \{u\}$
- 6 **continue** while-loop in line 2
- 7 **end**
- 8 **end**
- 9 **break** out of while-loop
- 10 **end**
- 11 **return** S'

4.2 Lower Bound Heuristic

In this Section, we propose a BS-based heuristic that returns a feasible solution that can be used as a lower bound for the size of a maximum γ -quasi clique in the input graph. This lower bound is used to obtain a high starting value for k in Line 2 of Algorithm 4.1. The pseudocode of the beam search is shown in Algorithm 4.3. In the following, note the textual distinction between *nodes* in the search tree and *vertices* in the input graph which is made for the sake of clarity.

In BS, a node v of the search tree is in general evaluated by a function $f(v) = g(v) + h(v)$, where $g(v)$ is the cost of the solution corresponding to a node, and $h(v)$ is a heuristically determined value given by a guidance function, which predicts the cost-to-go to complete the partial solution corresponding to a node. We use a simple data structure for nodes of the search tree where a node v has fields $v.S$ containing a feasible γ -quasi clique that this node represents, $v.n_e$, which contains the number of edges in $G[v.S]$, $v.d_S$, which is a vector containing the d_S -values for vertices in G in order to allow efficient computation of $v.n_e$, and $v.h$, which is the heuristically determined value of the node according to the used guidance function. Additionally, we define the cost of a node v as $g(v) = |v.S|$, which is the size of the feasible solution that this node represents. The root node r of the BS tree is then defined by an empty candidate set $r.S$ with $r.g = 0, r.h = 0, r.d_S = [0 \dots 0], r.n_e = 0$.

The main loop in Lines 3–17 expands all nodes in the current beam \mathcal{B} into feasible successor nodes. The maximum size of the beam is defined by the parameter *beta*, the so-called *beam-width*. Since a node v represents a set of vertices $v.S$ which is a feasible γ -quasi clique, successor nodes are generated by checking for each vertex $u \in V \setminus v.S$ whether $v.S \cup \{u\}$ is still a feasible γ -quasi clique. These successor nodes are then added to a set C , containing all expanded nodes of the next level of the search tree. Using the information stored in a node v of the beam search tree, it can be checked in $O(1)$ whether any vertex $u \in V \setminus v.S$ in the graph can be used to extend the solution $v.S$ into a feasible γ -quasi clique. Creating a successor node however takes $O(n)$ time, as the d_S -values for the created successor node have to be updated. In order to keep the computational effort low, we introduce a parameter ε that controls the maximum number of successor nodes for each node in the search tree. Thus, if a node has more than ε feasible successors, we add only the first ε successor nodes to C , as can be seen in Line 7 of the algorithm.

To avoid symmetries, i.e., nodes u, v representing the same solutions such that $u.S = v.S$, we use a hash map for each level of the search tree containing all solution vertex sets that were already added for expansion. Before adding a successor node v to C , a check is performed to make sure that a node representing $v.S$ is not already contained in C .

Note that due to this expansion into successor nodes all the solutions corresponding to the nodes on a level of the search tree have the same cardinality, and thus the same g -value. Therefore, g -values can be ignored in the evaluation, and all nodes in C are evaluated only by a guidance function h . The β nodes with the highest h -values then form a new beam while the remaining nodes are discarded. Furthermore, since all the

solutions corresponding to nodes on a level of the search tree have the same cardinality, a random node is selected to update the best-so-far solution S^* on each level. Finally, when the beam becomes empty, the maximum cardinality solution is returned.

The heuristic guidance function h can have a great impact on the quality of the obtained solutions, but also the runtime of the beam search. Therefore, we propose three guidance functions, *Greedy Construction*, *Feasible Neighbors*, and *Number of Edges*, which we want to consider in the context of this beam search:

- *Greedy Construction*. We use the result of a simple greedy construction as the value of the heuristic function h : Given a node v with candidate set $S = v.S$, greedily add the vertex $v = \arg \max_{u \in V \setminus S} d_S(u)$ if $S \cup \{v\}$ is still a feasible γ -quasi clique, where $d_S(u) = |\{v \mid v \in S, \{u, v\} \in E\}|$. Ties are broken randomly. This greedy extension is repeated until the corresponding solution is maximal, i.e., no single vertex can extend the cardinality of the solution while maintaining feasibility. The size of the solution obtained this way is then the return value of the guidance function $h(v)$. This greedy guidance function has a time complexity of $O(|S^*|n)$ per iteration, where $|S^*|$ is the size of the best found solution obtained by the greedy construction and n is the number of vertices in G . As $|S^*|$ is bound by n , the time complexity of *Greedy Construction* is in $O(n^2)$ per node of the search tree.
- *Feasible Neighbors*. Consider the following function defined for a node v in the search tree $h_{v.S}: V \setminus v.S \rightarrow \mathbb{N}$:

$$h_{v.S}(w) = \begin{cases} 0 & \text{if } w \text{ cannot extend } v.S \text{ into a feasible solution} \\ v.d_S(w) - \delta & \text{if } w \text{ can extend } v.S \end{cases},$$

where $\delta = \lceil \gamma \cdot \frac{|v.S| \cdot (|v.S| + 1)}{2} \rceil - v.n_e$, i.e. the number of edges that need to be added when extending $v.S$ into a feasible solution of size $|v.S| + 1$. The heuristic value $h(v)$ for a node v in the search tree is then computed as the sum $\sum_{w \in V \setminus v.S} h_{v.S}(w)$. For each vertex that can be used to extend $v.S$ - which we call a *feasible neighbor* in this context - we count the surplus of edges that are added to $v.S$ when extending $v.S$ by this vertex. As we have access to $v.n_e$, we know that each vertex that adds at least $\frac{|v.S| \cdot (|v.S| + 1)}{2} - v.n_e$ edges to the solution can be used to extend $v.S$. As the number of edges gained by adding a vertex is determined by the values in $v.d_S$, this computation can be done in time $O(n)$ per node of the search tree. Furthermore, consider two nodes u, v on the same level of the search tree, such that $u.n_e = v.n_e$, both nodes have the same number of feasible neighbors, but the d_S -values for the feasible neighbors of $u.S$ are higher than the d_S -values of the feasible neighbors of $v.S$. Intuitively, u is more promising than v , as adding one of the feasible neighbors adds more edges to the corresponding solution, which is also reflected in the computation of this heuristic guidance function.

- *Number of Edges*. We set $h(v) = v.n_e$, which means this heuristic guidance function has a time complexity of $O(1)$. This allows for efficient computation at the cost of not obtaining much additional information.

4. A LOCAL SEARCH BASED METAHEURISTIC FOR EDGE-BASED RELAXATIONS OF THE MAXIMUM CLIQUE PROBLEM

These heuristic guidance functions differ greatly in terms of time complexity per node and information obtained to guide the search. As the main purpose of the lower bound heuristic proposed in this Section is to quickly obtain a good approximation, we will prioritize fast computation over solution quality. Nonetheless, for small instances, it could still be useful to consider more expensive guidance functions.

Algorithm 4.3: Lower Bound heuristic based on Beam Search

Input: Graph G , Target density γ , guidance function h , beam width β , expansion control parameter ε
Output: A feasible γ -quasi clique S

```

1  $r.S, r.g, r.h, r.n_e \leftarrow \emptyset, 0, 0, 0$  // Root node  $r$ 
2  $S^* \leftarrow \emptyset$  // Best obtained solution
3  $\mathcal{B} \leftarrow \{root\}$  // Beam  $\mathcal{B}$ 
4 while  $\mathcal{B}$  not empty do
5    $C \leftarrow \emptyset$  // Set of successors (children) of nodes in  $\mathcal{B}$ 
6   for  $v \in \mathcal{B}$  do
7     Expand  $v.S$  into up to  $\varepsilon$  successor nodes that can be obtained by adding a
      vertex from  $V \setminus v.S$ 
8      $C \leftarrow$  add feasible successor nodes obtained from expanding  $v$ 
9   end
10  if  $C$  not empty then
11     $S^* \leftarrow v.S$  for some arbitrary node in  $C$ 
12  end
13  Evaluate nodes in  $C$  by guidance function  $h$ 
14   $\mathcal{B} \leftarrow$  Choose  $\beta$  nodes with highest  $h$ -values as new beam
15 end
16 return  $S^*$ 

```

4.3 Construction Heuristic

The construction heuristic in Line 5 in Algorithm 4.1 aims to return a set of nodes S of fixed size $|S| = k$ that is promising to lead to new local optima when used as a starting point for the local search procedure. The candidate solution returned by this construction heuristic is not necessarily feasible. The construction heuristic includes a parameter $b \in [0, 1]$ which controls the balance between choosing nodes that have not been explored yet and nodes that increase the number of edges among vertices in the current candidate solution. This approach of balancing exploration and objective value is inspired by [CCP⁺21], where the authors use a similar strategy during construction. We combine the authors' idea with a GRASP-like construction step. The procedure is shown in Algorithm 4.4. At first, a starting node is selected by choosing a node with the lowest *Freq*-value, breaking ties randomly. In each iteration of the main while loop in line 3-17, with probability b a node with the lowest *Freq*-value in $N_G(S)$ is added to

the current candidate solution S . If $N_G(S)$ is empty, an arbitrary node with the lowest $Freq$ value from $V \setminus S$ is selected instead. With probability $1 - b$ we perform an iteration of a GRASP-like construction step: A restricted candidate list RCL is built by adding all nodes from $V \setminus S$ that fulfill the condition $d_S(u) \geq d_{\max} - \alpha(d_{\max} - d_{\min})$, where $d_S(u)$ denotes the number of neighbors in S for a node $u \in V \setminus S$, and d_{\max}, d_{\min} are the maximum and minimum values $d_S(u)$, respectively, over all nodes $u \in V \setminus S$. Here, the parameter α controls the balance between greediness and randomness in a GRASP-like manner: $\alpha = 0$ corresponds to a greedy strategy, whereas $\alpha = 1$ corresponds to a purely random strategy.

Algorithm 4.4: Construction Heuristic with focus on exploration

Input: Graph G , n -element array $Freq$, Target size k , GRASP parameter α ,
Exploration parameter b

Output: Candidate set S

```

1  $u \leftarrow$  node in  $G$  with lowest  $Freq$ -value, break ties randomly
2  $S \leftarrow \{u\}$ 
3 while  $|S| < k$  do
4   if  $rand() < b$  then
5     if  $N_G(S)$  not empty then
6        $u \leftarrow$  Pick random neighbor in  $N_G(S)$  with lowest  $Freq$ -value
7     else
8        $u \leftarrow$  Pick random node in  $V \setminus S$  with lowest  $Freq$ -value
9     end
10  else
11     $d_{\min} = \min_{u \in V \setminus S} d_S(u)$ 
12     $d_{\max} = \max_{u \in V \setminus S} d_S(u)$ 
13     $RCL \leftarrow \{u \mid u \in V \setminus S, d_S(u) \geq d_{\max} - \alpha(d_{\max} - d_{\min})\}$ 
14     $u \leftarrow$  Pick random node from  $RCL$ 
15  end
16   $S \leftarrow S \cup \{u\}$ 
17 end
18 return  $S$ 

```

4.4 Neighborhood Structure and Local Search

The neighborhood of a candidate solution S is defined by a move operator that swaps a node $u \in S$ with a node $v \in V \setminus S$. The set of neighboring candidate solutions of S with $|S| = k$ that can be obtained by a single swap move is therefore defined as $\Omega_1 = \{(S \setminus \{u\}) \cup \{v\} \mid u \in S, v \in V \setminus S\}$ and has size $|\Omega_1| = k \cdot (n - k)$, which is in $O(n^2)$. Similarly, we define Ω_d for $d = 1, 2, 3, \dots$ as the set of neighboring solutions that can be obtained by swapping d nodes from S with d nodes from $V \setminus S$, obtaining a neighborhood of size $|\Omega_d| = \binom{k}{d} \cdot \binom{n-k}{d}$, which is in $O(n^{2d})$.

4. A LOCAL SEARCH BASED METAHEURISTIC FOR EDGE-BASED RELAXATIONS OF THE MAXIMUM CLIQUE PROBLEM

In general, this definition of a neighborhood structure can be applied to many MCP relaxations. The quality of a candidate solution then depends on the considered problem. However, for the MQCP and the MDCP, the definition of objective value coincides: a candidate solution S is of higher quality than a neighboring solution S' if $G[S]$ contains more edges than $G[S']$. Thus, we define the objective function $f: 2^V \rightarrow \mathbb{N}$ for the local search for MQCP and MDCP as the function that maps a subset of vertices $S \subseteq V$ to the number of edges in the induced subgraph $G[S]$. Therefore, S is of higher quality than S' if and only if $f(S) > f(S')$.

Similar algorithms that use local search to improve a candidate solution (e.g., [CCP⁺21], [DHB19], [ZBW20]) only search the neighborhood Ω_1 for improving solutions. As the size of this neighborhood is already quadratic in n , the authors of the mentioned algorithms restrict the neighborhood to promising moves. A scoring function is used to find promising vertices in $V \setminus S$ and in S that can be used for swaps. This scoring function is in most cases defined as $d_S(u) = |\{v \mid v \in S, \{u, v\} \in E\}|$. This way, the change in the objective function value, i.e., the *gain*, when swapping a node $u \in S$ with a node $v \in V \setminus S$ can efficiently be calculated by $\Delta_{uv} = d_S(v) - d_S(u) - e_{uv}$, where $e_{uv} = 1$ if $\{u, v\} \in E$ and $e_{uv} = 0$ otherwise. The scoring function d_S is thus used to restrict the neighborhood Ω_1 . Let $d_{\min} = \min_{u \in S} d_S(u)$ and $d_{\max} = \max_{u \in V \setminus S} d_S(u)$. Furthermore, let

$$\begin{aligned} X &= \{u \mid u \in S, \quad d_S(u) \leq d_{\min} + 1\}, \\ Y &= \{v \mid v \in V \setminus S, d_S(v) \geq d_{\max} - 1\}. \end{aligned}$$

The neighborhood Ω_1 of a candidate solution S is then restricted to only those neighboring solutions that can be obtained by swapping a node $u \in X$ with a node $v \in Y$, as all node swaps that maximize Δ_{uv} are contained in this restricted neighborhood.

In LSBM, we want to make use of this scoring function as well. The pseudocode for our local search procedure is shown in Algorithm 4.5. Note that a scoring function f_S is passed as an input parameter to the local search procedure. Within the context of LSBM, any scoring function can be used to restrict the candidate sets X, Y , and we plan to replace this scoring function with a GNN-based scoring function in Chapter 5. For this Chapter, however, we want to focus on the well-established scoring function d_S , i.e., f_S is instantiated by d_S .

In each iteration of the main while-loop in Algorithm 4.5, we first define the aspiration value a , which is the number of edges that have to be added to the current candidate solution S such that $f(S) > f(S^*)$ holds. Then, the restricted candidate sets X and Y are created from the sets S and $V \setminus S$, respectively by using the scoring function f_S . The sets X, Y are created as described before, with an additional restriction: To prevent cycling through the same candidate solutions over and over again, a short-term memory mechanism is applied which prevents unblocked vertices from being chosen for a swap if they do not fulfill an aspiration criterion. Two main methods can be found in the literature, which we want to evaluate in our algorithm:

- In [DHB19] and [ZBW20], a tabu list is used to prevent cycling. Each time a vertex is added to or removed from the solution, it is added to the tabu list for a fixed number of iterations. While a vertex is in the tabu list, it is blocked and cannot be used in swaps. In [DHB19], the authors use two tabu lists: One for moving vertices out of the candidate solution, and one for adding vertices to the candidate solution. In [ZBW20], a single tabu list is used for both operations.
- Configuration Checking is used by the authors of [CCP⁺21]. In its simplest form, once a vertex v is moved out of the candidate Solution, it is blocked from being moved again until at least one neighbor of v is moved. The authors refine this strategy by using a threshold for each node v that starts with $threshold(v) = 1$ for all $v \in V$. Each time a vertex v is moved into the candidate solution, $threshold(v)$ is increased by 1. When v is moved out of the candidate solution, it is blocked from being moved again, until $threshold(v)$ of its neighbors have been moved. Additionally, if $threshold(v) > ub_threshold$, where $ub_threshold$ is a user-defined upper bound, then $threshold(v)$ is reset to 1.

In Line 8, the neighborhood defined by X, Y is searched. The subroutine is shown in Algorithm 4.6. The neighborhood is searched exhaustively for the swap that maximizes the gain of edges Δ_{uv} . Blocked vertices are not considered for potential swaps unless the respective gain of edges is greater than the aspiration value a . If no improving swap was found, a random unblocked move is chosen $u \in X, v \in Y$. Afterward, the chosen vertices u, v are swapped and $Freq$, the short-term memory, and the d_S values are updated correspondingly. Furthermore, we define as a stopping criterion: If the best solution S^* does not improve for a fixed number of iterations η , the procedure is stopped and S^* is returned. Additionally, if a given time limit is reached, the procedure is stopped as well.

Summarizing the previous sections, we present a flowchart visualization of LSBM in Figure 4.1.

4.5 Adaption to the Maximum Defective Clique Problem

The MQCP and the MDCP can be approached similarly in the context of a local search-based algorithm, as during the local search procedure in both cases we try to find neighboring solutions that maximize the number of edges in the subgraph induced by the current candidate solution. The objective value of a candidate solution as defined in Section 4.4 is equivalent in both problems. Thus, we only have to change the feasibility checks for feasible MQCs in Line 7 of Algorithm 4.1, Line 4 of Algorithm 4.2, and Line 19 in Algorithm 4.5, as well as the expansion into feasible successors in the lower bound heuristic in Line 7 of Algorithm 4.3 to feasibility checks for MDCs. The proposed construction heuristic can be used for the MDCP without any change.

4. A LOCAL SEARCH BASED METAHEURISTIC FOR EDGE-BASED RELAXATIONS OF THE
 MAXIMUM CLIQUE PROBLEM

Algorithm 4.5: Local Search Procedure

Input: Graph G , Frequency list $Freq$, Target density γ , Candidate solution S ,
 scoring function f_S
Output: Best found solution S^*

```

1  $k \leftarrow |S|$ 
2  $S^* \leftarrow S$ 
3  $d_S \leftarrow$  compute  $d_S$ -values for vertices in  $G$ 
4 while stopping criterion not met do
5      $a \leftarrow f(S^*) - f(S)$  // aspiration value
6      $X, Y \leftarrow$  restrict candidate sets  $X \subseteq S, Y \subseteq V \setminus S$  using  $f_S$ 
7      $B \leftarrow$  blocked vertices according to short-term memory
8      $u, v, \Delta_{uv} \leftarrow SearchNeighborhood(G, d_S, X, Y, B, a)$ 
9     if  $\Delta_{uv} < 0$  then
10         // No improving solution found
11          $u, v \leftarrow$  choose arbitrary (unblocked)  $u \in X, v \in Y$ 
12     end
13      $S \leftarrow$  solution obtained by swapping  $u \in X, v \in Y$ 
14     Update  $Freq[u], Freq[v]$  with swap  $u, v$ 
15     Update short-term memory with swap  $u, v$ 
16     Update  $d_S$ -values
17     if  $f(S) > f(S^*)$  then
18          $S^* \leftarrow S$ 
19     end
20     if  $S^*$  is feasible then
21         return  $S^*$ 
22     end
23 return  $S^*$ 

```

Algorithm 4.6: Search restricted neighborhood

Input: Graph G , Restricted candidate sets X, Y . Blocked vertices B , aspiration value a

Output: Best found swap $u \in X, v \in Y$ and corresponding gain Δ_{uv}

```

1  $u_{best}, v_{best}, \Delta_{best} \leftarrow 0, 0, -\infty$ 
2 for  $u \in X, v \in Y$  do
3    $\Delta_{uv} \leftarrow d_S[v] - d_S[u] - e_{uv}$ 
4   if  $\Delta_{uv} \leq a \wedge (u \in B \vee v \in B)$  then
5     | continue
6   end
7   if  $\Delta_{uv} > \Delta_{best}$  then
8     |  $u_{best}, v_{best}, \Delta_{best} \leftarrow u, v, \Delta_{uv}$ 
9   end
10 end
11 return  $u_{best}, v_{best}, \Delta_{best}$ 
    
```

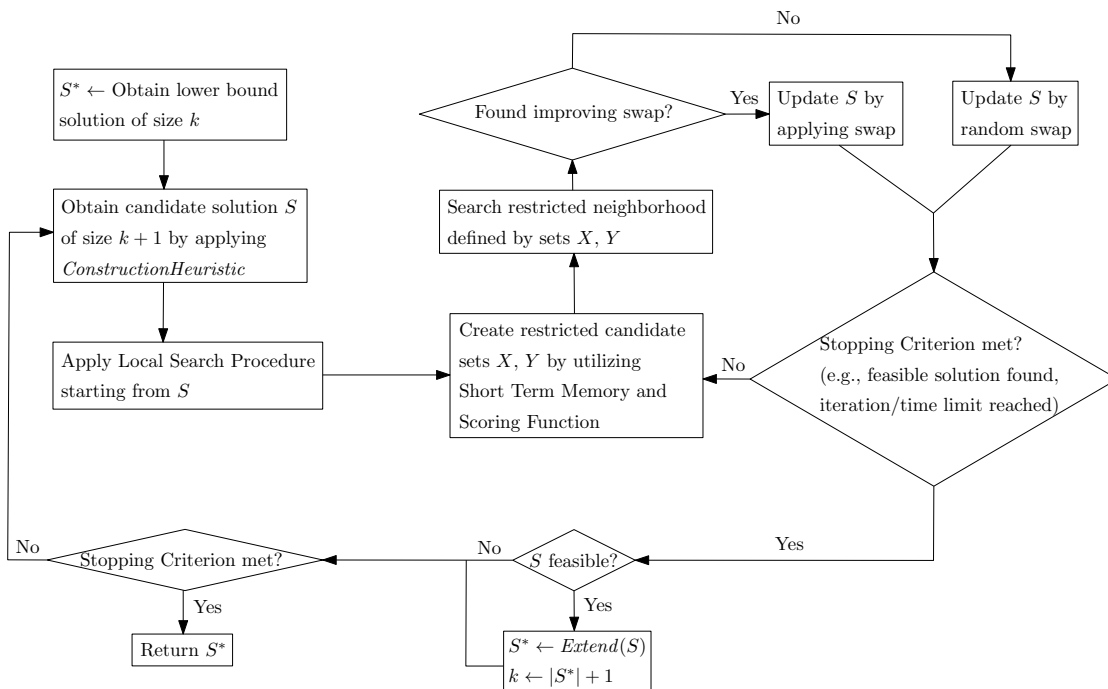


Figure 4.1: Flowchart of the structure of the Local Search Based Metaheuristic.

A GNN-based Metaheuristic

In this Chapter, we extend our algorithm LSBM presented in Chapter 4 by utilizing a GNN as a learnable scoring function to enhance the algorithm. Section 5.1 discusses the motivation for using a GNN within LSBM and points out weaknesses of current approaches that we attempt to overcome. Moreover, we show the general structure of our training algorithm LSBM-T and an overview of our approach. In the following Sections, we present in detail the components of LSBM-T. The definition of training samples is given in Section 5.2, and we discuss, how training data is generated. Next, in Section 5.3, we present our idea of using a look-ahead search that takes into account the current candidate solution and returns the best neighboring solution with respect to a user-defined neighborhood structure in order to obtain target values for training the GNN. Our proposed GNN architecture is shown in Section 5.4. Finally, we analyze the effect on the runtime of LSBM when using the GNN-based scoring function in contrast to using only d_S as a scoring function in Section 5.5.

Note that all the methods presented in this Chapter can be used both for the MQCP and the MDCP. Only LSBM itself has to be adapted as discussed previously in Section 4.5, but no changes have to be made in training data generation and during the look-ahead search, as the definition of the objective value of a candidate solution coincides for both problems.

5.1 Motivation and Overview

In Chapter 4 we presented a swap-based metaheuristic that evaluates possible swaps relative to a candidate solution S by their respective gain Δ_{uv} for $u \in S, v \in V \setminus S$ which is similar to current state-of-the-art heuristic solution approaches. Here, the function d_S can be seen as a scoring function over the vertices in V . Vertices with higher scores in $V \setminus S$ and vertices with low scores in S are promising candidates for swapping, as they maximize the expected gain of a swap Δ_{uv} . Most local search-based algorithms on the

MQCP rely on this scoring function, as it is efficiently computable, and swaps with a positive Δ_{uv} directly increase the objective value, which is the number of edges in the subgraph induced by the candidate solution. However, using d_S as a scoring function to evaluate the quality of swaps has two major drawbacks:

- In each iteration of the local search procedure, potential swaps are determined greedily: Only swaps that maximize the gain Δ_{uv} are considered. As discussed in Chapter 4, this corresponds to only searching the neighborhood structure Ω_1 . It is possible, however, that one or more swaps with low gain later lead to an improved solution – for example, a local optimum S with respect to the neighborhood structure Ω_1 is not necessarily a local optimum with respect to the neighborhood structure Ω_5 , and in order to obtain an improving solution from S with respect to Ω_5 , some swaps might have a negative gain when applied consecutively.
- As shown in [CCP⁺21], in most iterations there are many possible swaps with the same gain, but not all of them lead to the same local optimum. To decide between swaps with equal gain the authors propose to use a more fine-grained secondary scoring function that serves as a tiebreaker.

To address these drawbacks, we propose to incorporate a GNN into the local search algorithm as a possible improvement over the function d_S as a scoring function for swaps: In each major iteration of the local search algorithm, all vertices are evaluated by a GNN, assigning scores to vertices. The GNN can therefore be seen as a scoring function g_S , where $g_S(v)$ denotes the score of vertex v relative to candidate solution S . As with d_S , the scores returned by the GNN induce a ranking over the possible swaps: High scores for vertices in $V \setminus S$ and low scores for vertices in S shall indicate that these vertices are more likely to lead to an improved solution when considered for swaps.

The use of a GNN in the context of LSBM in this approach is two-fold:

- Similar to the use of d_S , the scoring function g_S can be used to restrict the neighborhood to only promising neighboring candidate solutions that can be obtained by only considering swaps involving the k' vertices with the highest scores in $V \setminus S$ with the k' vertices with the lowest scores in S .
- The GNN can be trained to determine scores not only based on the neighborhood Ω_1 but also to look ahead more than a single swap. This way, it can be used to obtain higher quality solutions as it is not caught in the local optima of the neighborhood structure Ω_1 .

We plan to achieve the desired behavior of g_S by training the GNN using principles from imitation learning: g_S is trained offline on representative problem instances to imitate an expert strategy that searches a user-defined neighborhood structure (e.g., $\Omega_1 \cup \Omega_2$) with respect to a candidate solution S and computes the neighboring candidate

solutions that maximize the gain of edges. From these neighboring solutions, binary target labels are computed for all vertices in the input graph, where vertices outside S are given a target value of one if they lead to one of the best neighboring solutions, and zero otherwise, whereas vertices in S are given a target value of one if they appear in every neighboring solution, and zero otherwise. The GNN has a final layer with a single sigmoid output per vertex and is trained to predict the target values computed using this expert strategy, which can be seen as a vertex-level binary classification task. The concrete details regarding our training algorithm are presented in Sections 5.2 and 5.3.

The concrete architecture of the GNN is inspired by [KvHW19], where the authors use an Encoder-Decoder-based architecture with a more complex attention-based encoder that generates node embeddings only once per graph, and a light-weight decoder that uses the node embeddings as inputs and is called multiple times to construct a solution in an end-to-end ML approach for the TSP. In our approach, evaluating the whole graph by a ConvGNN every time the candidate solution is updated by a swap is too expensive, as the convolution operation has a time complexity of $\Theta(n^2)$ for dense graphs. This motivates the choice of the described architecture. The node embeddings are generated only once per graph by the encoder, which is an attention-based spatial ConvGNN. The node embeddings and the current candidate solution S are then used to compute a context embedding. These embeddings are concatenated and used as inputs for the light-weight decoder, which is an MLP that evaluates the vertices of the graph with a time complexity of $O(n)$ and outputs scores for the vertices by the use of a final sigmoid layer with a single output per vertex. Note that g_S is not necessarily independent of d_S : In our approach, the d_S -values of vertices are used as features for the vertices in the graph when constructing the context embedding. Thus, the function d_S is also taken into consideration, but its information is enhanced by structural information obtained from the graph to overcome its drawbacks. The architecture of the Encoder-Decoder-based GNN is presented in detail in Section 5.4.

The attentive reader might have noticed that the approach described above is only sensible if the vertices in the input graph are represented by relevant initial feature vectors. As the input graphs are non-attributed, these node features cannot come directly from the problem input, unlike in [KvHW19], where the node inputs for solving the 2D Euclidean TSP are the 2D coordinates of the vertices embedded in the unit square. To overcome this issue, we investigate several feature initialization methods that extract information from the graph to compute initial node features. These methods can be categorized as centrality-based methods, where information is computed directly from properties of the vertices in the graph (e.g., degree-based initialization), and learning-based methods, where vertices in the graph are mapped into a feature space by a learnable function that tries to capture similarities between vertices. Using such feature initialization methods in node-level vertex classification tasks on non-attributed graphs has proven to be successful, as reported in [DHD⁺19]. We refer to Section 5.4 for more information about the choice of node features evaluated within our approach.

Even when employing the proposed Encoder-Decoder-based architecture, the use of a

GNN-based scoring function within our approach is computationally expensive compared to the scoring function d_S : Whereas d_S can be updated efficiently after a swap, incrementally updating the scores is generally not possible with a GNN, as the whole graph has to be re-evaluated. Therefore, obtaining scores is less efficient using g_S as a scoring function. In Section 5.5 we analyze from a theoretical perspective, how the use of g_S affects the performance of LSBM compared to the use of d_S .

5.2 Training the GNN

The general approach of the training algorithm LSBM-T is structured as depicted in Algorithm 5.1. A visualization of the algorithm structure is presented in the flowchart in Figure 5.1. At the start, the GNN-based scoring function g_S is initialized randomly. Then, in each iteration of the training loop a random representative instance is generated on the fly and LSBM is started with a fixed time limit τ , a maximum number of iterations without improvement η until the local search procedure is restarted, a maximum number of restarts ξ if no feasible solution is found, and - most importantly - with the scoring function g_S that is used to restrict the neighborhood during the local search procedure instead of the scoring function d_S . The size of the restricted neighborhood during this local search is defined by input parameter k' , which sets the maximum cardinality for the restricted candidate sets X, Y .

We adapt LSBM slightly to return a swap history H in Line 5 besides the best found solution S^* . This swap history is a simple array-like data structure that contains an entry for each call to the local search procedure during the execution of LSBM. An entry consists of the candidate solution that is generated by the construction heuristic at the start of the local search procedure and a chronological list of pairs of vertices in the input graph that correspond to the vertices swapped during the local search procedure. Thus, the number of encountered candidate solutions, denoted as $|H|$, their order of appearance, and all encountered candidate solutions themselves can be obtained from H . After the execution of LSBM has finished, n_s encountered candidate solutions are reconstructed from the entries in H to generate training samples. This is done by sampling n_s numbers without replacement from $\{1, \dots, |H|\}$ and reconstructing the respective candidate solutions in their order of appearance in H . Note that in general n_s is much smaller than $|H|$, as thousands of candidate solutions can be encountered during the execution of LSBM. Only a small subset \mathcal{S} of these is chosen in Line 6 as training samples, as the generation of these samples can be computationally expensive. A look-ahead search is applied to compute the best neighboring solutions for each candidate solution in \mathcal{S} , returning a vector t of target values for the vertices in G . From G, \mathcal{S}, t , a training sample is created and added to the replay buffer R . For a detailed description of how target values for these samples are computed, we refer to Subsection 5.2.1 and Section 5.3, where we propose and discuss different methods to identify and compute improving neighboring solutions. At the end of each iteration of the training loop, the GNN is trained with data from the replay buffer, if it has reached at least half of its

maximum fill capacity ρ . This process is repeated for z iterations until the GNN is sufficiently trained.

Using the NN during training data generation is common practice in reinforcement learning, where the NN is often improved by “self-play”. We justify this algorithmic design choice by arguing about the trajectory of the search, i.e., the set of candidate solutions encountered during execution of the local search-based metaheuristic with or without the GNN: using the GNN-based scoring function g_S during training data generation will generate samples that would appear naturally during the execution of LSBM at test time. In contrast, using only d_S as a scoring function during training data generation, the trajectory of the search might differ greatly compared to the trajectory when using g_S - especially for an initially untrained GNN, which might decrease the stability of training.

Algorithm 5.1: Training algorithm LSBM-T

Input: Target density γ , Number of iterations z , replay buffer size ρ , size of restricted neighborhood k' , number of samples generated per iteration n_s

Output: Trained GNN-based scoring function g_S

```

1 Initialize untrained GNN-based scoring function  $g_S$ 
2 Initialize Replay Buffer  $R$ 
3 for  $z$  iterations do
4    $G \leftarrow$  create representative graph instance
5    $S^*, H \leftarrow$   $LSBM(G, \gamma, g_S, k')$ 
6    $\mathcal{S} \leftarrow$  reconstruct  $n_s$  randomly sampled candidate solutions from  $H$ 
7   for  $S \in \mathcal{S}$  do
8      $t \leftarrow$   $LookaheadSearch(G, S)$ 
9     Add  $(G, S, t)$  to  $R$ 
10  end
11  if  $|R| > \frac{\rho}{2}$  then
12    Train  $g_S$  with data from  $R$ 
13  end
14 end
15 return  $g_S$ 

```

5.2.1 Obtaining Target Values

As already mentioned, a training sample consists of a graph $G = (V, E)$, a candidate solution S , and a vector of target values t containing target values for each vertex in G . Training is done in an imitation learning setting: In order to obtain target values for the vertices in V , we apply a look-ahead search to the candidate solution S as described in Section 5.3, and the GNN is then trained to imitate this look-ahead search.

The computation of target vector t is shown in Algorithm 5.2. It is a subroutine called at the end of the look-ahead search. The look-ahead search first computes the set

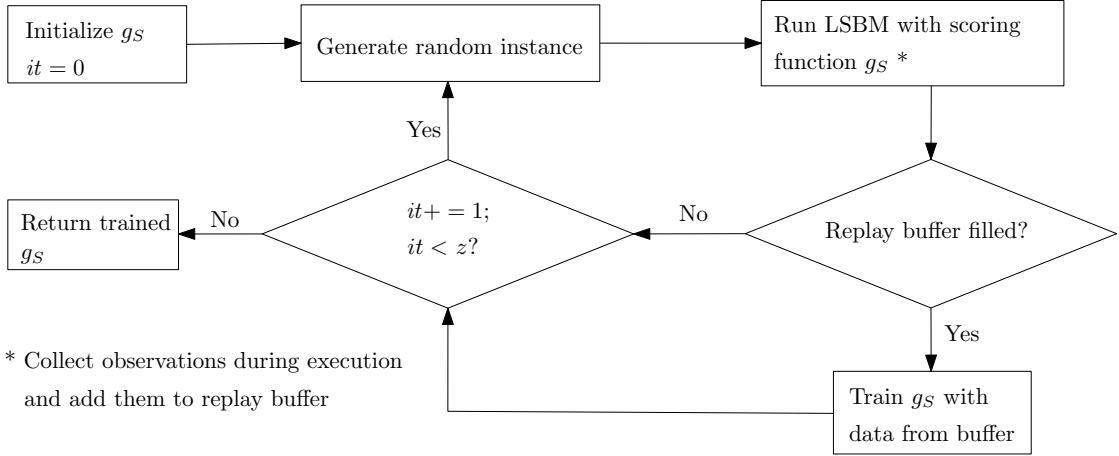


Figure 5.1: Flowchart of the structure of the training algorithm.

$\mathcal{T} = \{(X_1, Y_1), \dots, (X_j, Y_j)\}$ of vertices $X_i \in S, Y_i \in V \setminus S, 1 \leq i \leq j$ that need to be removed from and added to the current candidate solution S , respectively, in order to obtain a neighboring solution with respect to S that maximizes the objective value. This set \mathcal{T} is passed as an input argument in Algorithm 5.2 to compute the target vector t . Note that there might be multiple such neighboring solutions with equal objective value, or none if S is a local optimum. The target vector t is then initialized by setting $t[u] = 1$ for $u \in S$, and $t[v] = 0$ for $v \in V \setminus S$. Then, for every pair of vertex sets $(X_i, Y_i) \in \mathcal{T}$ we set $t[u] = 0$ for $u \in X_i$ and $t[v] = 1$ for $v \in Y_i$. The resulting target vector t is then returned. We demonstrate this computation by a small example in Figure 5.2.

Algorithm 5.2: Computation of target values

Input: Graph G , Candidate solution S , Set \mathcal{T}

Output: Vector of target values t

```

1  $t \leftarrow [0 \dots 0]$  //  $n$ -element vector
2 for  $u \in S$  do
3   |  $t[u] = 1$ 
4 end
5 for  $(X_i, Y_i) \in \mathcal{T}$  do
6   | for  $u \in X_i$  do
7     |  $t[u] = 0$ 
8   | end
9   | for  $v \in Y_i$  do
10  | |  $t[v] = 1$ 
11  | end
12 end
13 return  $t$ 
  
```

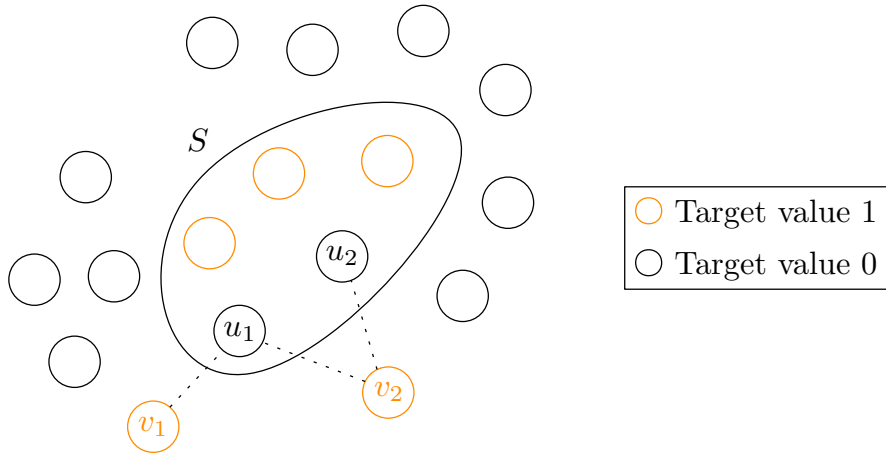


Figure 5.2: Target vector t obtained from a graph G with candidate solution S . Edges are omitted for the sake of clarity. The three best neighboring solutions of S w.r.t. the Ω_1 neighborhood structure can be obtained by applying one of the swaps $(\{u_1\}, \{v_1\})$, $(\{u_1\}, \{v_2\})$, $(\{u_2\}, \{v_2\})$ (marked by dotted lines). Thus, the vertices v_1, v_2 and the vertices in $S \setminus \{u_1, u_2\}$ are labeled with 1, whereas all remaining vertices are labeled with 0.

As the target vector t contains a label of either zero or one for each vertex in G , this method of training can be seen as a vertex-level binary classification task. To train the GNN we propose to use the binary cross entropy¹ as a loss function, which is suitable for binary classification tasks:

$$\text{binarycrossentropy}(\hat{y}, t) = \frac{1}{n} \sum_{i=1}^n (-t[i] \cdot \log(\hat{y}[i] + \epsilon) - (1 - t[i]) \cdot \log(1 - \hat{y}[i] + \epsilon))$$

Here, \hat{y} denotes a vector of size $|V|$ that contains predictions given by the GNN-based scoring function g_S , t denotes the target vector computed as described above, and ϵ denotes a small positive number that is included to avoid infinity.

5.3 Look-Ahead Search

In order to obtain target values for training the GNN, we need to be able to identify the best neighboring solutions in a given neighborhood structure relative to the S . More specifically, we want to consider the neighborhood structure(s) $\bigcup_{i=1}^d \Omega_i$ for different values of d . In this section, we propose different methods in order to search these neighborhood structures.

¹More precisely, we use Flux.jl's `logitbinarycrossentropy` function, which is mathematically equivalent, but more numerically stable

5.3.1 A simple approach: Searching Ω_1

As a first approach, we can define the look-ahead search as simply searching the neighborhood structure Ω_1 w.r.t. a candidate solution S , which can be done in time $O(n^2)$ by inspecting all possible swaps. Note that the optimization described in 4.4 using the restriction of possible swaps to vertices from $X \subseteq S, Y \subseteq V \setminus S$ can be applied in order to speed up the search. When g_S is trained successfully to imitate this look-ahead search, it will serve the same purpose as the function d_S in Chapter 4, as d_S is used to efficiently search the neighborhood structure Ω_1 .

5.3.2 Beyond Ω_1 : Exact Approaches

In order to search the larger neighborhoods $\bigcup_{i=1}^d \Omega_i$ for $d > 1$, we distinguish between exact approaches and heuristic approaches. In the remainder of this section, let $X_i \subseteq S, Y_i \subseteq V \setminus S$ be two sets of vertices of equal cardinality and let $S' = (S \setminus X_i) \cup Y_i$ be the neighboring solution of S obtained by removing the vertices in X_i from S and adding the vertices in Y_i to this set of vertices. Let $\text{gain}_S(X_i, Y_i) = |E(G[S'])| - |E(G[S])|$ denote the gain of edges when comparing S' and S .

Enumerating all possible swaps

As already stated in Chapter 4, the neighborhood structure Ω_d contains $O(|V|^{2d})$ neighboring solutions and can be searched in time $O(|V|^{2d} \cdot T_d)$ by enumerating all possible swaps, where T_d is the time needed for determining the objective value of a neighboring solution in Ω_d . The $\text{gain}_S(X_i, Y_i)$ of a swap of nodes $X_i = \{u_1, \dots, u_d\}, Y_i = \{v_1, \dots, v_d\}$ is calculated using d_S the following way:

$$\text{gain}_S(X_i, Y_i) = \sum_{u \in X_i} d_S(u) - \sum_{v \in Y_i} d_S(v) + |E(G[X_i])| + |E(G[Y_i])| - \sum_{u \in X_i} \sum_{v \in Y_i} e_{uv}, \quad (5.1)$$

where $e_{uv} = 1$ if $\{u, v\} \in E$, and $e_{uv} = 0$ otherwise. The above equation is a generalization of the calculation of Δ_{uv} : When adding the vertices in Y_i to the candidate solution S , we add $\sum_{u \in Y_i} d_S(u)$ edges between vertices in Y_i and S , and $|E(G[Y_i])|$ edges between vertices in Y_i , from which we have to subtract $\sum_{u \in X_i} \sum_{v \in Y_i} e_{uv}$ edges between X_i and Y_i , as the vertices from X_i are removed from S . When removing the vertices in X_i from S , we remove $\sum_{u \in X_i} d_S(u)$ edges, but the edges in $G[X_i]$ are counted twice, which is why we need to add the term $|E(G[X_i])|$. Note that T_d is in $O(d^2)$, as the number of summands in Equation 5.1 depends on d .

We propose the following optimizations to speed up the search:

- Let $X_j = \{u_1, \dots, u_d\} \subseteq S, Y_j = \{v_1, \dots, v_d\} \subseteq V \setminus S$ such that $d_S(u_i) \leq u$ for $1 \leq i \leq d, u \in S \setminus X_j, d_S(v) \geq v$ for $1 \leq i \leq d, v \in V \setminus (S \cup Y_j)$, so X_j contains the d vertices in S with the lowest d_S -values, and Y_j contains the d vertices in $V \setminus S$ with the highest d_S -values. Then, for any sets of vertices $X_i = \{u_1, \dots, u_d\} \subseteq S, Y_i = \{v_1, \dots, v_d\} \subseteq V \setminus S$ it must hold that $\text{gain}_S(X_i, Y_i) \leq$

$\sum_{v \in Y_j} d_S(v) - \sum_{u \in X_j} d_S(u) + 2 \cdot \binom{d}{2}$. If the upper bound on the right side of the inequality is less than or equal to zero, then S is a local optimum with respect to Ω_d , and it is not necessary to examine any of the neighboring solutions at all.

- Similarly, we can derive a lower bound: Let X_j, Y_j be as defined above. Then, $\text{gain}_S(X_j, Y_j)$ is clearly a lower bound for the maximum gain among all neighboring solutions of S . When calculating $\text{gain}_S(X_i, Y_i)$ for arbitrary sets $X_i = \{u_1, \dots, u_d\} \subseteq S, Y_i = \{v_1, \dots, v_d\} \subseteq V \setminus S$, we can therefore check, whether $\sum_{u \in Y_i} d_S(u) - \sum_{v \in X_i} d_S(v) + 2 \cdot \binom{d}{2} < \text{gain}_S(X_j, Y_j)$. If this is the case, we can avoid the computation of $\text{gain}_S(X_i, Y_i)$. This reduces the time T_d from $O(d^2)$ to $O(d)$ if the above condition is fulfilled.

In practice, preliminary experiments have shown that searching the neighborhood exhaustively by enumeration of all possible swaps as described above is computationally infeasible for $d \geq 3$ even for small graphs ($|V| \leq 250$), even when applying the described optimizations.

MILP Approach

As an alternative approach, we present the following MILP-formulation which we adapted from the formulations in [PVBB13] and [VPBP16].

The formulation is based on F1 from [PVBB13]:

$$\max \sum_{i=1}^n \sum_{j=i+1}^n a_{ij} w_{ij} \quad (5.2)$$

$$\text{s.t.} \quad \sum_{i \in V} x_i = k \quad (5.3)$$

$$\sum_{i \in S} x_i \geq k - d \quad (5.4)$$

$$w_{ij} \leq x_i \quad \forall \{i, j\} \in E \quad (5.5)$$

$$w_{ij} \leq x_j \quad \forall \{i, j\} \in E \quad (5.6)$$

$$w_{ij} \geq x_i + x_j - 1 \quad \forall \{i, j\} \in E \quad (5.7)$$

$$w_{ij} \geq 0 \quad \forall \{i, j\} \in E \quad (5.8)$$

$$x_i \in \{0, 1\} \quad i \in V \quad (5.9)$$

We change the objective function to determine the neighboring solution S' of S that can be obtained after at most d swaps and maximizes the number of edges in $G[S']$. The equality in 5.3 ensures that the neighboring solution contains $|S| = k$ vertices, and the inequality in 5.4 ensures that $|S' \setminus S| \leq d$.

Furthermore, note that this method has similarities to the idea from formulation F3 in [VPBP16] presented in Section 3.4, where the variables $z_{\omega^\ell}, \dots, z_{\omega^u}$ are introduced that

“guess” the size of the solution, which must be somewhere between the lower and upper bounds ω^l, ω^u . The formulation above can therefore be seen as fixing the variable $z_k = 1$ while setting $z_j = 0$ for $j \neq k$. Theoretically, this should speed up the search significantly, as the search space is reduced greatly by fixing the size k of the solution.

In practice, we obtain similar results as for the exhaustive search of $\bigcup_{i=1}^d \Omega_i$: Even for relatively small graphs ($|V| \leq 250$), this method becomes infeasible for $d \geq 3$.

5.3.3 Beyond Ω_1 : Heuristic Approaches

In order to make training feasible on larger instances and for greater d , we propose heuristic approaches that approximate the best neighboring solution, trading solution quality for efficiency.

Restricting the neighborhood

Recall that during the execution of LSBM we restrict the cardinality of the sets of vertices $X \subseteq S, Y \subseteq V \setminus S$ to the k' vertices with the lowest and highest g_S -values, respectively. Similarly, when searching $\bigcup_{i=1}^d \Omega_i$, we can restrict the cardinality of the sets of vertices considered for swaps X, Y to contain up to k' vertices with the lowest and highest g_S -values each, and then exhaustively enumerate all possible neighboring solutions that can be obtained by removing up to d vertices in X from S and adding up to d vertices from Y . Intuitively, it makes sense to use a value $k^* > k'$ for the size of the restricted neighborhood during the look-ahead search, as this means a greater part of the neighborhood structure $\bigcup_{i=1}^d \Omega_i$ is searched and therefore the approximation is more likely to be precise. Furthermore, by using a fixed size neighborhood determined by k^* , the computation scales better to larger graphs.

For making this training process more stable, we introduce a user-defined parameter $\omega \in \mathbb{N}$ to the look-ahead search. This parameter dictates the number of iterations that the scoring function d_S is used to restrict the neighborhood during the look-ahead search instead of the scoring function g_S , as the latter is initially untrained and thus unlikely to restrict the neighborhood to the most promising vertices. After the first ω iterations of the training loop in LSBM-T were completed, the look-ahead search then uses the scoring function g_S to restrict the neighborhood.

Furthermore, we note that the optimizations we proposed to exhaustively search the neighborhood $\bigcup_{i=1}^d \Omega_i$ can also be applied to this neighborhood restricted by a GNN-based scoring function.

5.4 GNN Architecture

In this Section, we present the key components of the GNN architecture used in our algorithm and justify our choices in designing said architecture. The convolution operation of attention-based spatial ConvGNNs has a time complexity of $\Theta(n^2)$ for dense graphs

(i.e., each vertex in the input graph has $O(n)$ neighbors), which means it is too expensive to re-evaluate the graph by the GNN-based scoring function after every swap of vertices. Thus, we follow the work presented in [KvHW19], where an Encoder-Decoder-based GNN architecture is used in an end-to-end ML approach for the 2D Euclidean TSP. The main motivation for utilizing the Encoder-Decoder paradigm lies in its gain in efficiency: The costly attention-based encoder is only evaluated once per graph to compute node embeddings. Every time the context, i.e., the candidate solution S , changes during the local search, only the decoder is applied to re-evaluate the vertices in the input graph. In contrast, the lightweight decoder only has a time complexity of $O(n)$.

The whole process of evaluating a graph $G = (V, E)$ and a candidate solution S is presented through a small example in Figure 5.3. Initially, feature vectors x_1, \dots, x_n are computed for all vertices in $V = \{v_1, \dots, v_n\}$, producing a feature matrix of dimension $dim_{in} \times |V|$, where dim_{in} is the size of a feature vector. This feature matrix is used as input for the GNN-based encoder, which consists of l_e graph convolutional layers and outputs a node embedding matrix of dimension $dim_{out} \times |V|$. From these node embeddings, a context embedding is computed by taking into account the current candidate solution S , and aggregating corresponding node embeddings by an aggregation function agg . The context embedding and input features derived from the d_S -values for the vertices in V are then concatenated to the node embeddings. The resulting matrix is used as input for the decoder, which is realized as an MLP with a final layer that maps the values of the previous layer to a scalar output score by applying a sigmoid activation function.

In the remaining Section, we will discuss the used node features, the more specific structure of the encoder and decoder, and the context embedding in detail.

5.4.1 Node Features

While the feature initialization method is not a part of the NN itself, it plays an important role in the overall process of computing the node embeddings. Since the node embeddings are only computed once per instance, the computed node features must represent the vertices in a meaningful way by extracting similarities and differences between vertices and their surrounding neighborhoods. More precisely, high-quality features should map vertices with similar neighborhoods closer together in the feature space, whereas vertices that do not share any structural similarities should be further apart. Moreover, the chosen feature initialization method should ideally generalize well on unseen graphs, as the GNN used in our algorithm is trained offline on representative problem instances that differ from the instances encountered during execution.

In [DHD⁺19] the authors analyze the effect of different node feature initialization methods in vertex-level node classification tasks. They distinguish between centrality-based approaches (e.g., degree-based initialization, number of triangles that a vertex participates in, etc.), and learning-based approaches (e.g., DeepWalk [PAS14], where vertices are mapped closer together if the probability that they occur in a random walk together is higher). The authors conclude that centrality-based approaches are generally not

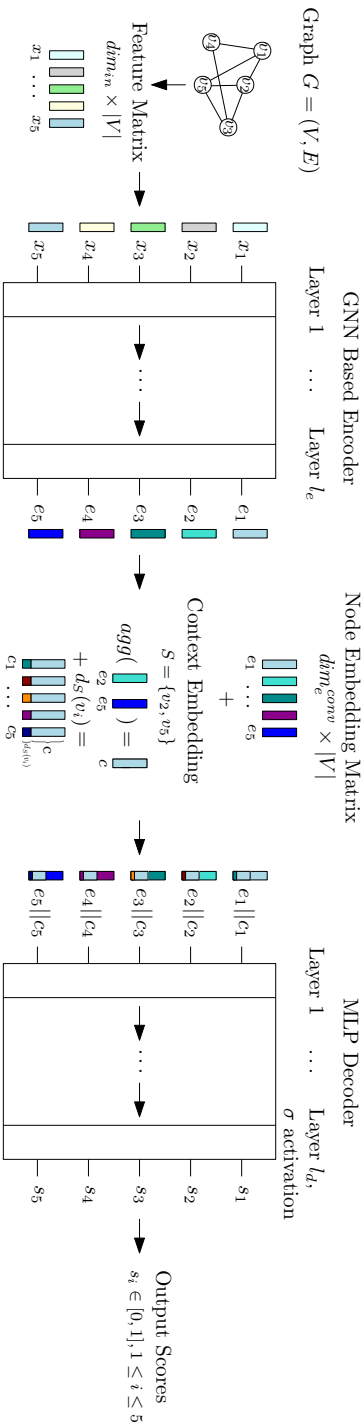


Figure 5.3: Evaluation of a graph $G = (V, E)$ with candidate solution $S = \{v_2, v_5\}$ by the Encoder-Decoder-based architecture.

as suitable as learning-based approaches by evaluating different feature initialization approaches in a node classification setting.

We investigate both the application of centrality-based and learning-based feature initialization methods within the context of LSBM. More precisely, the following methods are considered:

- *Degree*. Each vertex is simply initialized by its degree, normalized by $\frac{1}{n}$.
- *EgoNet*. For a vertex v we derive a feature vector of size three from its EgoNet. It contains the number of vertices in the EgoNet of v , the number of edges in the subgraph induced by the EgoNet of v , and the number of edges that have exactly one endpoint in the EgoNet of v . Note that to avoid the vanishing gradient problem, the first value of the feature vector generated for each vertex is normalized by $\frac{1}{n}$, whereas the second and third values are normalized by $\frac{1}{\binom{n}{2}}$. Furthermore, we consider the 1-hop-EgoNet and the 2-hop-EgoNet as separate node features.
- *Node2Vec*. Input features for the vertices of the input graph are learned using the learning-based method Node2Vec [GL16]. We consider this method, as the authors show that the produced embeddings can capture the similarities of vertices that have similar neighborhoods.
- *Struc2Vec*. Struc2Vec [FRS17] is used to learn the node features. This method was proposed by the authors to capture structural similarities between vertices regardless of the length of the shortest path between the vertices.

The concrete parameterizations of the learning-based feature initialization method are given in Subsection 6.2.2, which shows the results of our evaluation regarding node features within the context of our algorithm. Furthermore, we evaluate each of the proposed node features on their own, and combinations of them, i.e., we concatenate the output of two different feature initialization methods.

5.4.2 Encoder

In general, any ConvGNN model can be used as an encoder. Recent applications of GNNs in the context of COPs (e.g., [KvHW19], [JCRL21], [HLMP21]) show that attention-based GNNs seem especially promising, which is why we want to build our proposed GNN architecture upon an attention-based encoder as well. The architecture of our encoder builds upon the work presented in [KvHW19], in which the authors use attention-based GNNs to solve the Euclidean 2D TSP. We use their encoder architecture in this proposed ConvGNN model and slightly adapt it to our purpose:

- Layer 1: We apply a linear transformation $A_1x_i + b_1$ with learnable parameters A_1, b_1 for each feature vector $x_i, 1 \leq i \leq n$.

- Layers 2, ..., l_e : Following the ConvGNN model used by [KvHW19], each attention layer consists of two sub-layers: A multi-head attention (MHA) layer and a dense, fully connected feed-forward (FF) layer. Each sub-layer also applies a skip-connection and batch-normalization. The following sub-layer definitions are given by the authors:

$$\begin{aligned}\hat{h}_i &= \text{BN}^l(h_i^{(l-1)} + \text{MHA}_i^l(h_1^{(l-1)}, \dots, h_n^{(l-1)})) \\ h_i^{(l)} &= \text{BN}^l(\hat{h}_i + \text{FF}^l(\hat{h}_i))\end{aligned}$$

In [KvHW19], the authors use as the MHA layer an attention mechanism that is based on the transformer architecture in [VSP⁺17]. We replace this layer with the newer GATv2 layer from [BAY21], which was developed as an improved version of the Graph Attention Network [VCC⁺18]. Here, a GATv2 attention layer is defined by the update function

$$h_i^{(l)} = \sum_{j \in N_G[i]} \alpha_{ij} W_1^{(l)} h_j^{(l-1)}.$$

The attention coefficients α_{ij} are defined as

$$\alpha_{ij} = \frac{1}{z_i} \exp(a^T \text{LeakyReLU}(W_2^{(l)} h_i^{(l-1)} + W_1 h_j^{(l-1)})),$$

where z_i is a normalization factor and a, W_1, W_2 are learned layers.

In our concrete implementation, we denote by dim_{in} the size of the node embeddings, by $\text{dim}_e^{\text{conv}}$ the size of all MHA layers, and by dim_e^{FF} the size of all FF layers in the encoder.

5.4.3 Context

Within LSBM, the context is determined by the current candidate solution S . The context embedding is computed by aggregating the columns of the node embedding matrix that correspond to the vertices in S . This is inspired by graph representation learning, where the node embeddings are aggregated into a graph embedding by the application of an order invariant aggregation function $\text{agg} \in \{\text{sum}, \text{mean}, \text{max}\}$. Most commonly, however, *mean* is used to compute the graph embedding, e.g., in [KvHW19]. We follow a similar approach but only consider the columns of the node embedding matrix that correspond to the vertices in S , as we are only concerned about the representation of the candidate solution. The main motivation behind this approach is to compress a context of linear size into a constant-size context embedding c . However, we note that this definition of a context is one of the current weaknesses of our approach – by taking the mean of the node embeddings of the vertices in S the information of the candidate solution is highly compressed. Future work will be dedicated to developing methods that generate a stronger context embedding that preserves more relevant information about the candidate solution.

Furthermore, we derive features from the function d_S . From the d_S -values for the vertices in the input graph, we compute the mean and standard deviation. Then, for each vertex

in the graph, we concatenate a vector containing its d_S -value and the mean and standard deviation of all d_S -values to the context embedding c , creating feature vectors c_i for $v_i, 1 \leq i \leq |V|$. These vectors are then concatenated to the node embeddings, creating the final decoder inputs $e_i \parallel c_i$ for $v_i, 1 \leq i \leq |V|$.

5.4.4 Decoder

The decoder is a classification model that maps the inputs obtained from the node embeddings and the context embedding to scalar values representing scores for the vertices in V . We suggest using a simple MLP architecture. The input dimension of the decoder is determined by its inputs, which are computed as previously described by concatenating the context embeddings to the node embeddings. The first layer of the decoder then simply applies a learnable linear transformation, followed by a batch normalization. Furthermore, it consists of l_d layers, with layers $2, \dots, l_d - 1$ being hidden feed-forward layers of dimension dim_d with a ReLU activation function, each followed by a batch normalization. The final layer l_d of the decoder has a single dimension, mapping the results of the previous layer to scalar values between zero and one with a sigmoid activation function. The concrete number of layers and their dimensions are shown in Chapter 6, where we evaluate suitable values for our approach.

5.5 Performance

Since the main difference between the algorithms presented in Chapter 4 and 5 lies in the scoring function used to evaluate vertices during the swap-based local search procedure, we want to analyze the effect of using the scoring function g_S to restrict the neighborhoods instead of the scoring function d_S . Note that d_S has to be used in LSBM, even when another scoring function is used to restrict the neighborhoods, as using the d_S -values is the most efficient way to keep track of the objective value of candidate solutions. Thus, we know that using the scoring function g_S can never be faster than using d_S , alone, but we want to analyze, where possible bottlenecks can be found and how to keep the additional computational effort as low as possible. In order to do so, we identify the three main operations during the local search procedure that are influenced by the choice of scoring function: Initialization, restricting neighborhoods, and updating the scoring function.

5.5.1 Initializing the scoring function d_S

At the start of the local search procedure, a construction heuristic is used to produce a new candidate solution S , which serves as a starting point. The scoring function d_S must then be initialized, as it is dependent on S . The initialization procedure can be seen in Algorithm 5.3. It starts by initializing an array of zeros of size n . For each vertex $u \in S$, increase the value $d_S[v]$ by one for each neighbor v of u . This procedure has a worst-time-complexity of $O(n^2)$ time, as the size of S is in $O(n)$, and each vertex in S can have $O(n)$ neighbors.

Algorithm 5.3: Initialize scoring function d_S

Input: Graph $G = (V, E)$, candidate solution S
Output: Initialized scoring function d_S

```
1  $d_S \leftarrow [0, \dots, 0]$  // Array of size  $n$ 
2 for  $u \in S$  do
3   | for  $v \in N_G(u)$  do
4   |   |  $d_S[v] \leftarrow d_S[v] + 1$ 
5   | end
6 end
7 return  $d_S$ 
```

5.5.2 Initializing the scoring function g_S

When using the scoring function g_S , initializing the scoring function is done by computing node embeddings for each vertex in G . This is done by first computing node features and then computing node embeddings by passing the node features through a GNN-based encoder. The time complexity of this initialization depends on the two parts. Firstly, the chosen node features have to be computed once per graph. This can be as simple as obtaining the node degrees, but can also be much more complex when for example learning-based node features are used. Thus, let T_f be the node feature-specific time taken for initialization. Secondly, the convolutional operations defined by the layers of the GNN itself will take $O(n^2)$ time. The argument is similar as with d_S , as each vertex needs to be evaluated by taking into account all its neighboring vertices. Note that this initialization process has to be done only once per graph, thus it has a combined runtime of $O(T_f + n^2)$. In subsequent restarts of the local search procedure, the node embeddings are already precomputed, and only the linear time decoder has to be applied.

Since initialization is only done once per graph or once per restart of the local search procedure, the runtime should not have a great impact on the performance of the algorithm. Therefore, especially with a GNN-based scoring function, this time should be used by utilizing an encoder of high enough complexity to produce high-quality node embeddings.

5.5.3 Restricting the neighborhood using d_S

When using the scoring function d_S , the neighborhood Ω_1 is searched implicitly, as only swaps that maximize the gain are considered. The restricted candidate sets $X \subseteq S, Y \subseteq V \setminus S$ are generated in time $O(n)$, as only the minimum and maximum scores for vertices in S and $V \setminus S$, respectively, have to be determined. As a reminder, the restricted candidate sets are defined as

$$X = \{u \mid u \in S, \quad d_S(u) \leq d_{\min} + 1\},$$
$$Y = \{v \mid v \in V \setminus S, d_S(v) \geq d_{\max} - 1\},$$

with $d_{\min} = \min_{u \in S} d_S(u)$, $d_{\max} = \max_{v \in V \setminus S}$. If the swap $u \in S, v \in V \setminus S$ maximizes the gain of the candidate solution, this method of restricting the neighborhood guarantees that $u \in X, v \in Y$. While this method produces restricted neighborhoods of variable size, these neighborhoods are usually small and thus can be searched efficiently in practice. In general, however, the size of the sets X, Y can be in $O(|V|)$ in the worst case.

5.5.4 Restricting the neighborhood using g_S

Using the scoring function g_S , we do not have a guarantee that the swap that maximizes the gain lies among the lowest-scoring vertices in S and the highest-scoring vertices in $V \setminus S$. Moreover, we might not even want to consider such a swap, as the scoring function was trained to search greater neighborhoods than Ω_1 , and the swap that maximizes the gain does not lead to the best neighboring solution. We therefore propose to use restricted candidate sets X, Y of a fixed size k' , where X consists of the k lowest-scoring vertices in S , and Y consists of the k' highest-scoring vertices in $V \setminus S$. This guarantees that searching the neighborhood takes a fixed amount of time. Obtaining the k' highest or lowest scoring vertices can be done in time $O(k' \cdot n)$, which is linear for fixed k' , but is still more expensive than generating a restricted neighborhood for the scoring function d_S . As the restricted neighborhood is searched in every iteration of the local search procedure, its size can therefore also have a great impact on the runtime. It is therefore essential to determine a neighborhood size that strikes a good balance between efficiency and the quality of neighboring solutions found in the neighborhood.

5.5.5 Updating the scoring function d_S

After each swap of vertices $u \in S, v \in V \setminus S$ the scoring function needs to be updated, as it is dependent on the current candidate solution S . For d_S , this update can be efficiently in $O(|V|)$ time by decrementing $d_S[x]$ for $x \in N_G(u)$ and incrementing $d_S[y]$ for $y \in N_G(v)$. All other vertices remain unaffected by a single swap and their corresponding d_S -values do not need to be updated.

5.5.6 Updating the scoring function g_S

For the scoring function g_S , we can generally not update the scores incrementally, as the context embedding changes, which is part of the input of all vertices. Therefore, all vertices need to be re-evaluated by the decoder. This still takes time linear in the number of vertices, but, depending on the complexity of the decoder, it can take considerably more time than updating d_S after a swap. As this operation is done in every iteration of the local search procedure, it has a great impact on the performance of the algorithm. Therefore, the complexity of the decoder should be chosen such that it is as small and efficient as possible while still providing high-quality predictions.

Initialization		Restricting the neighborhood		Update	
d_S	g_S	d_S	g_S	d_S	g_S
$O(n^2)$	$O(T_f + n^2)$	$O(n)$	$O(k' \cdot n)$	$O(n)$	$O(n)$

Table 5.1: Worst-case time complexity of the main operations used during LSBM when using the scoring functions d_S and g_S .

5.5.7 Conclusion

When analyzing the performance aspect of the proposed scoring functions, it seems clear that a GNN-based scoring function can in practice not be as fast and efficient as the scoring function d_S alone. However, we want to address this weakness of our approach by keeping the NN – especially the decoder – as small and efficient as possible, by determining a neighborhood size that balances efficiency and solution quality, and by training the NN to provide better guidance through the search space than d_S , leading to solutions of improved quality. Finally, in Table 5.1 we summarize the discussed worst-case time complexity of each of the three main operations.

Computational Experiments & Evaluation

We conduct several computational experiments to evaluate the components and parameter settings of LSBM and LSBM-T. An overview of parameters for LSBM and LSBM-T is given in Tables 6.1 and 6.2. The evaluation of the BS-based lower bound heuristic is shown in Section 6.1, where we compare the runtimes and quality of obtained solutions of the proposed guidance functions with several parameter configurations. Next, we use Section 6.2 to analyze in detail how the different parameter settings of LSBM-T affect the performance of the trained GNNs on graphs of different sizes and densities. Most importantly, we show in Subsection 6.2.1 that LSBM-T is able to successfully train the GNNs to imitate the look-ahead search and that the trained models effectively guide the local search during LSBM. Then, we investigate different parameter settings of LSBM-T. Our focus lies on evaluating the different feature initialization methods and combinations thereof, and depth values for the look-ahead search parameter d . Furthermore, we investigate how our approach generalizes to bigger, unseen instances. Next, in Section 6.3 we discuss the settings used for the execution of LSBM, and how different LSBM-specific parameters affect runtime and solution quality. Finally, the performance of our approach is evaluated on well-established benchmark instances, and we compare the results to the state-of-the-art in Section 6.4.

The computational experiments are focused on evaluating results obtained on dense graphs, i.e., the number of neighbors of each vertex is in $O(n)$. These instances seem especially challenging, as exact algorithms are often infeasible to use in practice on dense graphs. We consider both random graphs and selected benchmark instances within the scope of our evaluation. Moreover, we primarily focus on the evaluation of MQCP instances, as we expect our findings to apply to the MDCP as well.

Parameter	Description
\mathcal{M}	Short Term Memory type (TabuList, ConfigurationChecking)
α	GRASP control variable of construction heuristic, controls randomness
b	Exploration parameter b for construction heuristic
β	Beam width of beam search used as lower bound heuristic
ε	Maximum number of successor nodes for a node in lower bound heuristic
τ	Cutoff time for execution of algorithm
η	Local search is restarted after η iterations without improvement
ξ	Execution is terminated after ξ restarts without improvement
k'	Maximum size of sets X, Y in restricted neighborhood during local search

Table 6.1: LSBM parameters

A Note on Random Graphs

All random graphs are generated using the Erdős-Rényi random graph model [ER59], also referred to as *uniform* random graphs, where, given $n \in \mathbb{N}, m \in \mathbb{N}, m \leq \binom{n}{2}$, a random graph is sampled with uniform probability from all graphs with n vertices and m edges. Note that instead of using the parameter m directly, we generate graphs by specifying a density $0 \leq \text{dens}(G) \leq 1$ from which m can be easily obtained as $\lceil \text{dens}(G) \binom{n}{2} \rceil = m$. We use this random graph model to generate graph instances in all our experiments due to its generality and ease of implementation. However, we note that there are other methods to generate random graphs (e.g., [ACL01]), and any method can be used within our approach. Especially if some structural properties of the graphs encountered at test time are known, it can lead to improved performance if these structural properties are also present in the generated random instances during training.

We generally sample $n, \text{dens}(G)$ from distributions \mathcal{V}, \mathcal{D} to make the trained GNNs more robust towards small variations in size and density of the input graphs. The instances generated are thus defined by a tuple, e.g., $(\mathcal{V} \sim \mathcal{N}(200, 10), \mathcal{D} \sim \mathcal{U}(0.45, 0.55))$ denotes that instances are generated by sampling the number of vertices n from a normal distribution \mathcal{N} with $\mu = 200, \sigma^2 = 10$ and the density of the graph is sampled from a uniform distribution \mathcal{U} with $a = 0.45, b = 0.55$.

Experiment Setup

We implemented LSBM and LSBM-T in Julia 1.8.3 using Flux ¹ and GraphNeuralNetworks ²[Loc21] for the implementation of the GNN, and Word2Vec ³ for the implementation of Node2Vec and Struc2Vec. All computational experiments in this chapter are executed on an Intel Xeon E5-2640 processor with 2.40 GHz. For training the NNs, we use a memory limit of 32 GB, whereas, for all other experiments a memory limit of 20 GB is used.

¹<https://github.com/FluxML/Flux.jl>

²<https://github.com/CarloLucibello/GraphNeuralNetworks.jl>

³<https://github.com/JuliaText/Word2Vec.jl>

Parameter	Description
d	Depth of look-ahead search
k^*	Maximum size of sets X, Y in restricted neighborhood in look-ahead search
dim_e^{conv}	Dimension of Convolutional layers of encoder
dim_e^{FF}	Dimensions of Feed Forward layers of encoder
l_e	Number of layers in encoder
dim_d	Dimensions of Feed Forward layers of decoder
l_d	Number of layers in decoder
ρ	Maximum replay buffer capacity
z	Number of iterations of training loop
n_f	List of node features extracted from the input graph (Degree, EgoNet, Node2Vec, Struc2Vec)

Table 6.2: LSBM-T parameters

Benchmark Instances

The DIMACS benchmark set is a heterogenous collection of artificially created graphs of different sizes and densities. It is a standard benchmark set for the MCP and related problems and it is used to evaluate the performance of state-of-the-art MQCP (e.g., [PWWW21], [CCP⁺21]) and MDCP (e.g., [CZHX21]) algorithms. Similarly, the BHOSLIB benchmark set is a collection of large graphs of high density which was specifically created for performing benchmark tests for graph problems. We evaluate the performance of our algorithm on a subset of these benchmark instances.

6.1 Lower Bound Heuristic

The BS-based lower bound heuristic proposed in Section 4.2 is a heuristic method that produces a feasible solution on its own. We presented three different heuristic guidance functions: *Greedy Completion*, which has a complexity of $O(|S^*|n)$ per node of the beam search tree, where $|S^*|$ is the size of the best found solution, *Feasible Neighbors*, which has a complexity of $O(n)$ per node, and *Number of Edges*, which returns in $O(1)$ the density of the candidate solution represented by a node in the search tree and thus corresponds to a simple generalized greedy construction in the context of Beam Search.

Our proposed lower bound heuristic is controlled by two parameters, namely the beam width $\beta \in \mathbb{N}$ and the expansion control variable $\varepsilon \in \mathbb{N}$, which is the maximum number of successor nodes for each node in the search tree. Clearly, these two parameters control the breadth of the search tree and thus can be used to balance runtime and solution quality. As the lower bound heuristic is used to find a feasible solution quickly in the context of LSBM, we prioritize runtime over solution quality, as the main part of the computation within LSBM is done during local search. Nonetheless, we evaluate different parameter settings for the different heuristic functions to obtain a good starting solution even with a low computational effort.

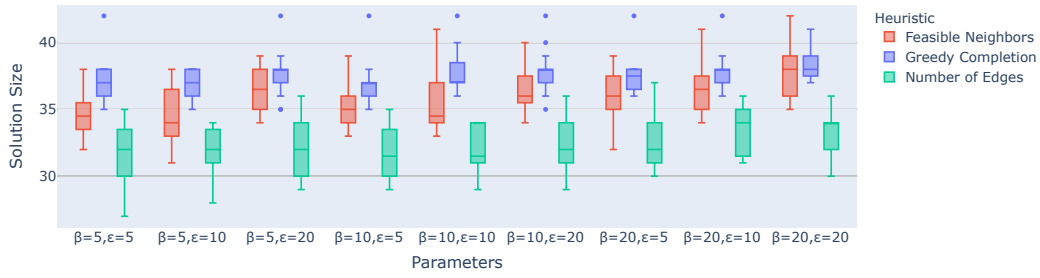
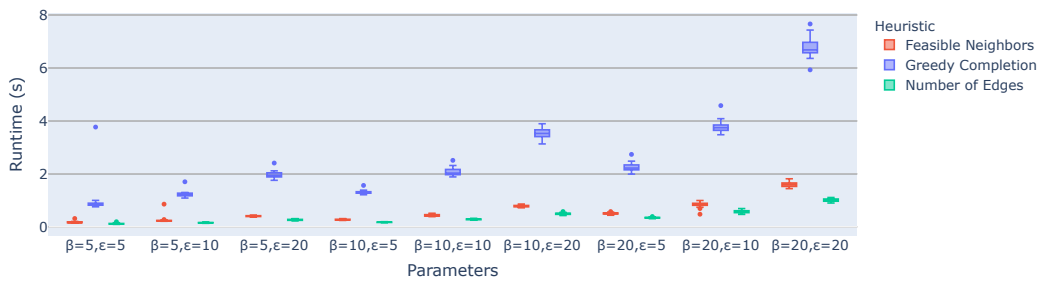
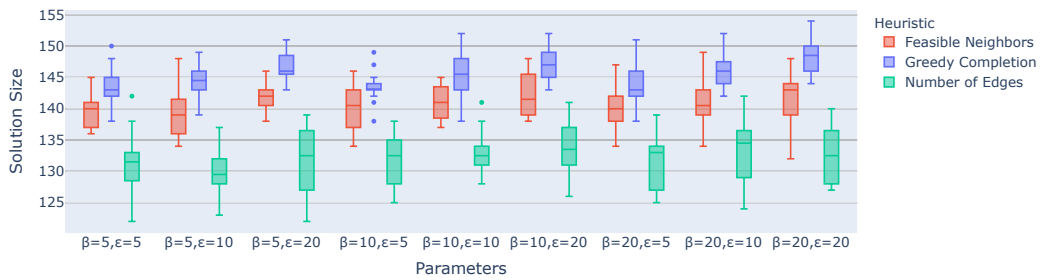
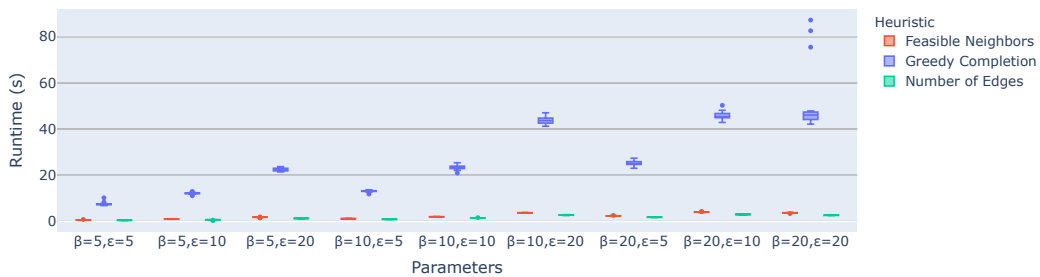
In Figures 6.1 and 6.2 we show the obtained solution qualities and runtimes of the three heuristics on randomly generated instances of different sizes and densities. We generated 20 uniform random graphs for each combination of $|V| \in \{500, 1000\}$, $\text{dens}(G) \in \{0.75, 0.9\}$ and ran our beam search with MQCP parameter $\gamma = 0.95$. As to be expected, *Greedy Completion* outperforms the other heuristics in terms of solution quality, but due to its time complexity, it should only be used when smaller solutions are to be expected, as the time complexity per node of the search tree is dependent on the solution length when using this heuristic evaluation function. In comparison, the solution quality obtained when using *Feasible Neighbors* is slightly lower, but especially when larger solutions are expected, the speedup is significant. We can also see that the difference in runtime between *Feasible Neighbors* and *Number of Edges* is marginal, while the former produces better results in all tested hyperparameter settings. We conclude that *Greedy Completion* works well on instances where smaller solutions are expected, but on large instances with high densities and low γ the much more efficient *Feasible Neighbors* heuristic should be used. Thus, we set $\beta = 10, \varepsilon = 10$ in all further experiments and use the heuristic guidance function *Feasible Neighbors*, as we argue that the speedup compared to *Greedy Completion* outweighs the loss in solution quality within the context of LSBM.

6.2 Evaluation of Training Parameters

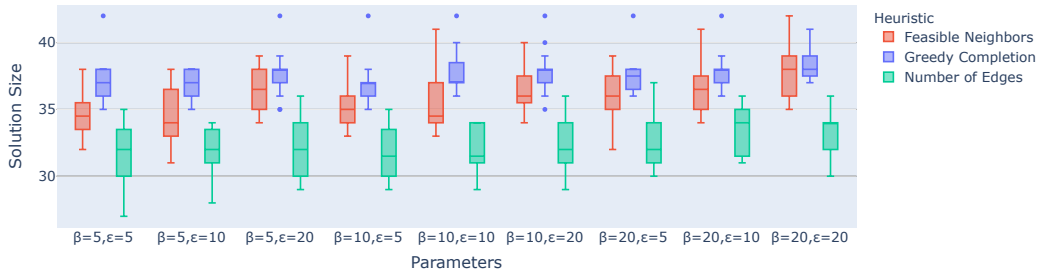
The performance of LSBM when using a GNN-based scoring function depends mainly on how effectively the GNN is trained to make high-quality predictions. First, we show that our training algorithm LSBM-T effectively trains the GNN to imitate the search of a neighborhood, which is a central result of this thesis. Furthermore, we provide evidence from computational experiments, that, after the training has been completed, the GNN-based scoring function g_S is an improvement over the scoring function d_S and guides the local search more effectively. In the remaining Section, we evaluate the impact of different feature initialization methods, different values for the look-ahead search parameter d , and the generalization on unseen, larger graph instances.

For the execution of LSBM within LSBM-T, we use the following settings in this Section unless stated otherwise: The short-term memory mechanism used in all experiments is a tabu list with a tabu tenure of ten, the construction heuristic parameters $\alpha = 0.2, b = 0.3$, the lower bound heuristic parameters $\beta = \varepsilon = 10$, a time limit $\tau = 300.0$, a maximum number of iterations without improvement $\eta = 2000$, and a maximum number of $\xi = 3$ restarts before execution is terminated. During the evaluation of random graphs and benchmark instances we set $\xi = 10$, but keep the remaining parameters unchanged.

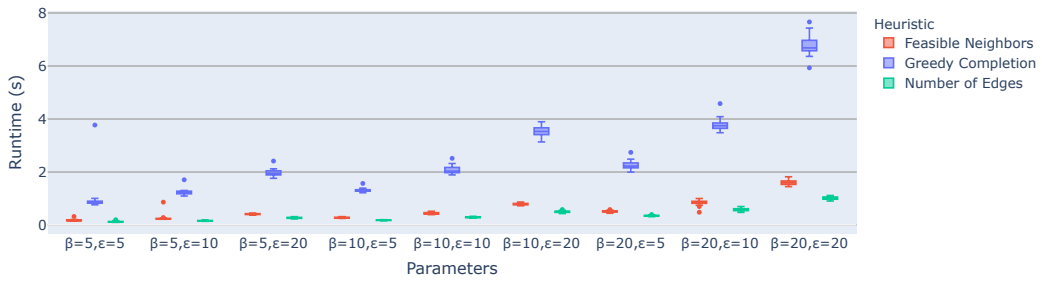
The parameter settings regarding the GNN were obtained by manually fine-tuning our approach over many experiments. We use a GATv2-based encoder as described in Section 5.4 with four attention heads per layer, $l_e = 3$ attention layers of size $\text{dim}_e^{\text{conv}} = 64$ and feed-forward layers of size $\text{dim}_e^{\text{FF}} = 128$, and an MLP-decoder with $l_d = 2$ hidden layers of size $\text{dim}_d = 32$. We did not notice any significant increase in performance using an encoder or decoder with more or higher-dimensional layers, and in terms of efficiency,

(a) Solution sizes for $\text{dens}(G) = 0.75$ (b) Runtimes in seconds for $\text{dens}(G) = 0.75$ (c) Solution sizes for $\text{dens}(G) = 0.9$ (d) Runtimes in seconds for $\text{dens}(G) = 0.9$ Figure 6.1: Beam Search solution sizes and runtimes in seconds for instances with $|V| = 500, \text{dens}(G) \in \{0.75, 0.9\}$

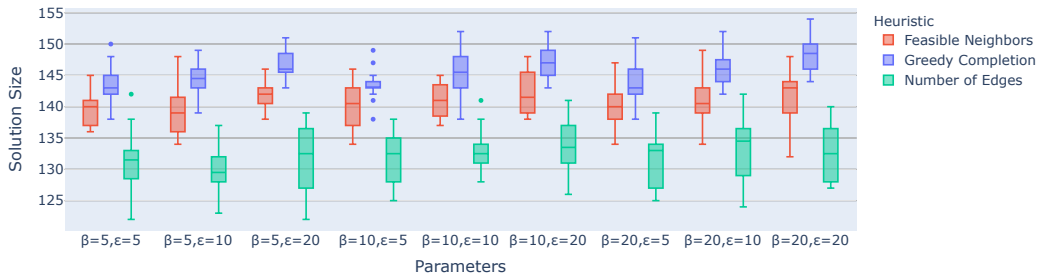
6. COMPUTATIONAL EXPERIMENTS & EVALUATION



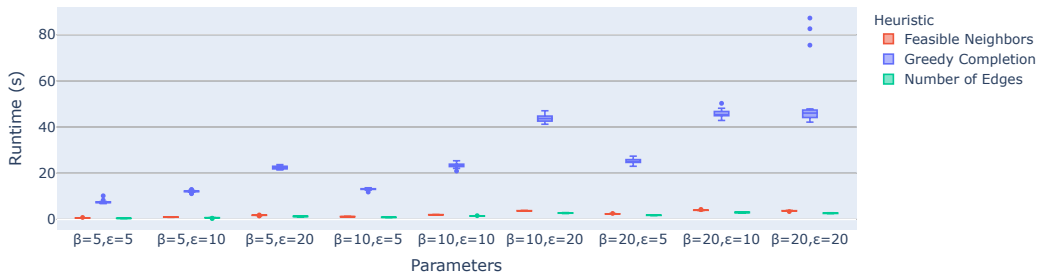
(a) Solution sizes for $\text{dens}(G) = 0.75$



(b) Runtimes in seconds for $\text{dens}(G) = 0.75$



(c) Solution sizes for $\text{dens}(G) = 0.9$



(d) Runtimes in seconds for $\text{dens}(G) = 0.9$

Figure 6.2: BS solution sizes and runtimes in seconds for instances with $|V| = 1000, \text{dens}(G) \in \{0.75, 0.9\}$.

Benchmark Set	Instance	V	E	$\text{dens}(G)$
DIMACS	brock200_2	200	9876	0.496
DIMACS	brock400_1	400	59723	0.748
DIMACS	brock400_2	400	59786	0.749
DIMACS	brock400_3	400	59681	0.748
DIMACS	C250	250	27984	0.899
BHOSLIB	frb30-15-1	450	83198	0.824
BHOSLIB	frb30-15-2	450	83151	0.823
BHOSLIB	frb30-15-3	450	83216	0.824
BHOSLIB	frb30-15-4	450	83194	0.823
BHOSLIB	frb30-15-5	450	83231	0.824

Table 6.3: Benchmark instances used for experiments

it is clearly beneficial to keep the NN as small as possible. By this reasoning, we justify our choice of the presented GNN parameter settings.

The remaining LSBM-T specific parameters are set as follows: The size of the restricted neighborhood is set to $k' = 20$. Furthermore, we run LSBM-T with a replay buffer size $\rho = 1000$ for $z = 300$ iterations and generate $n_s = 50$ training samples per iteration. The GNN is trained in four batches of size eight in each iteration. We use the ADAM optimizer with a learning rate of 0.001 and a momentum of (0.9, 0.999) and train the models using FLux.jl’s logitbinarycrossentropy loss function.

In the following experiments, we evaluate the results of the trained models on uniform random graphs and on selected benchmark instances. The benchmark instances used in this Section are shown in Table 6.3 for reference. In all experiments shown in this Section, we train ten GNNs for each investigated parameter configuration unless stated otherwise.

6.2.1 Effectivity of Training

The most important questions that arise in the context of this work are whether the presented training algorithm LSBM-T successfully trains the GNN to imitate the look-ahead search and how the results obtained using the GNN-based scoring function compare to those obtained using d_S . To answer these questions, we show the results of two test runs of training ten GNNs each using LSBM-T. During each iteration of the training loop, LSBM is executed twice on the random instance generated in this iteration, once with the GNN-based scoring function g_S , and once with d_S , while all other parameters regarding the execution of LSBM are equal in both runs. In the first experiment, we trained ten GNNs on MQCP instances with $|V| = 200, \text{dens}(G) = 0.65, \gamma = 0.999$ using a combination of Node2Vec and Struc2Vec as a feature initialization method, and in the second experimental run ten models are trained on MQCP instances with $|V| = 400, \text{dens}(G) = 0.75, \gamma = 0.999$ and the feature initialization method Struc2Vec is used.

The line plots in Figure 6.3 show the gap between the solution size of the best found solutions $S_{g_S}^*$, $S_{d_S}^*$ obtained by LSBM using g_S and d_S , respectively. The gap is computed as $\frac{|S_{g_S}^*| - |S_{d_S}^*|}{|S_{d_S}^*|}$. Therefore, a positive gap between solution sizes indicates that LSBM was able to find a larger solution using g_S as a scoring function compared to using d_S .

We observe that in both runs during the first 50 iterations the gap is negative, which is to be expected as the GNN is initialized randomly. Note that the training starts after the replay buffer was filled to half its capacity, which is after the completion of ten iterations of the training loop. As the training progresses and the loss decreases, the solution quality of LSBM using g_S increases. This strongly indicates that the GNN is successfully learning to imitate the look-ahead search.

In the first experiment on smaller instances, we notice that after the first 200 iterations of the training loop the gap never reaches a value below zero, which shows that LSBM consistently performs equally well or better using g_S as compared to d_S . The second experiment is done to evaluate the effect on larger MQCP instances with $|V| = 400$, $\text{dens}(G) = 0.75$, $\gamma = 0.999$. We observe a higher variance in the gap, but smoothing the lines shows that the average gap is above zero, which again indicates that the scoring function g_S outperforms d_S .

Furthermore, we perform additional experiments to compare the solution quality of LSBM using g_S and d_S on random instances. We train ten GNNs each for three feature initialization methods on instances of different sizes and densities and evaluate them by the average solution quality obtained on a test set of 20 random instances for each instance group. Figure 6.4 shows the results. The dashed line represents the average solution quality of LSBM using the scoring function d_S on the same test instances. Regarding the execution of LSBM, the same parameter configuration is used when using the trained GNN-based scoring functions or d_S . On both instance sets, we observe that the average solution quality is higher when using the trained GNN-based scoring functions.

In conclusion, the reported observations show strong evidence that LSBM-T successfully trains the GNNs to imitate the look-ahead search and the trained GNN-based scoring functions outperform the scoring function d_S on random instances that are similar to those the GNNs were trained on.

6.2.2 Node Features

In Chapter 5 we proposed four different feature initialization methods: Degree (D), EgoNet of size 1 and 2, (E1, E2), Node2Vec (N2V), and Struc2Vec (S2V). What follows is an evaluation of these feature initialization methods and combinations thereof: D+N2V, D+S2V, E1+N2V, E1+S2V, N2V+S2V, and E1+E2. In total, we thus investigate ten different node features.

We note that N2V and S2V node features are learned using the recommended settings from the respective papers ([GL16], [FRS17]): we sample 20 random walks of length 80

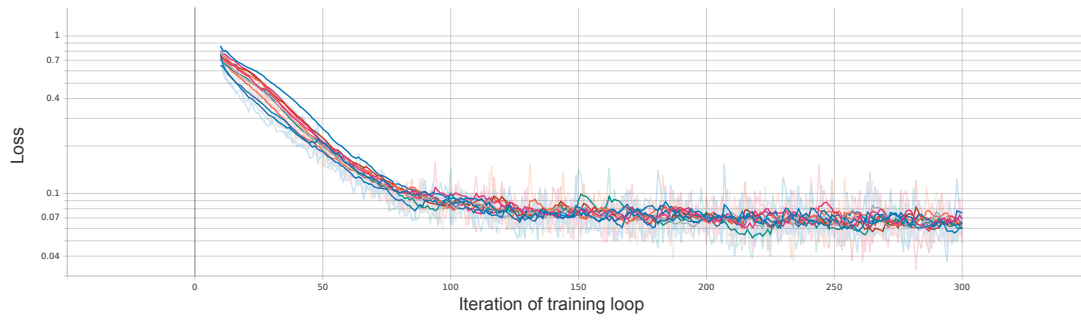
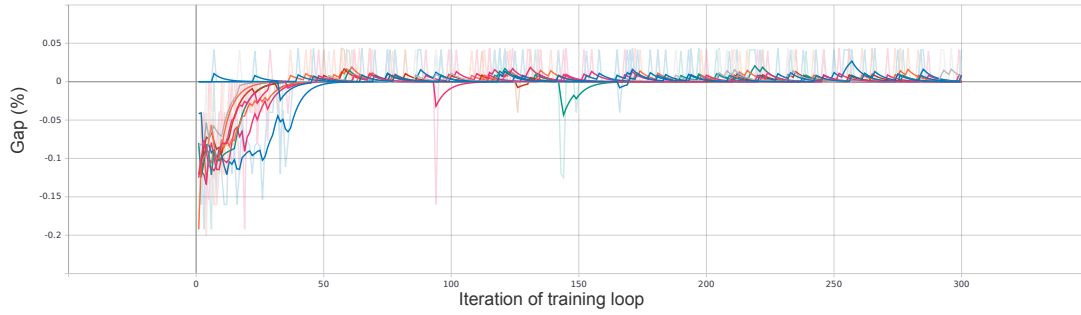
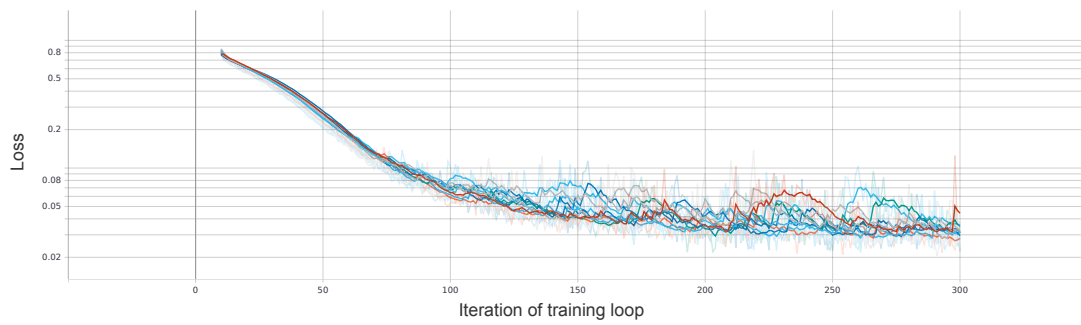
(a) Ten runs with $|V| = 200$, $\text{dens}(G) = 0.65$, $\gamma = 0.999$ (b) Ten runs with $|V| = 400$, $\text{dens}(G) = 0.75$, $\gamma = 0.999$

Figure 6.3: Each line in the plots represents an independent run of the training algorithm LSBM-T. In each iteration, a random instance is generated and LSBM is performed using the scoring functions g_S and d_S , producing the best found solutions $S_{g_S}^*$, $S_{d_S}^*$, respectively. The gaps between these solutions are computed as $\frac{|S_{g_S}^*| - |S_{d_S}^*|}{|S_{d_S}^*|}$.

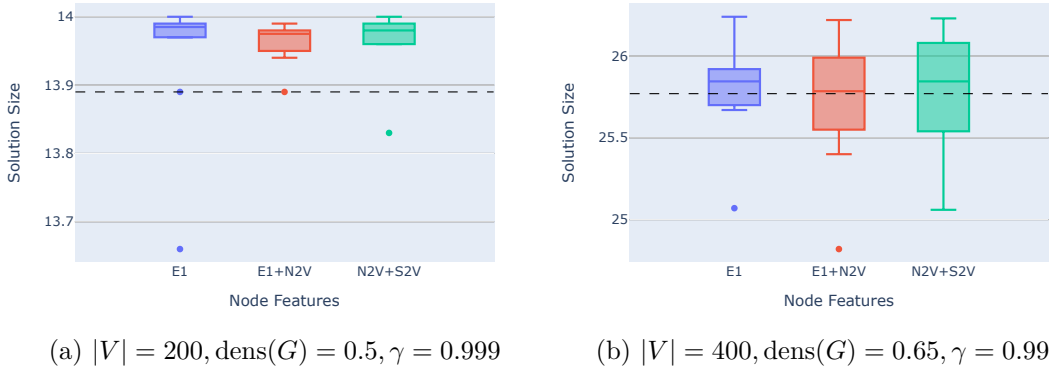


Figure 6.4: Comparison of solution quality of g_S and d_S . The average solution quality on 20 random instances by d_S is marked by the dashed line. LSBM is executed with the same parameters for g_S and d_S .

for each vertex in the input graph, and node contexts are generated with a window size of five to produce node embeddings of size 64. Furthermore, for N2V we use a return parameter $p = 2$ and an in/out-parameter $q = 4$, as preliminary experiments have shown these parameter settings to be suitable to focus on locality-based feature extraction, and for S2V we use a layer transition probability of 0.3, which is the default setting in the authors’ S2V implementation.

In each experimental run, we trained ten models for each node feature on uniform random graphs defined by a specific size and density. Furthermore, in each run, we use 20 unseen graphs drawn from the same distribution and evaluate the trained models by the average solution quality obtained on these test instances. The results are then grouped by the feature initialization method. Additionally, we compare the results obtained on random instances to selected benchmark instances that are similar in size and density to those seen during training. Moreover, all models are trained with a look-ahead depth $d = 1$, as we want to investigate, which node features are suited to imitate a simple look-ahead search before trying higher and thus more computationally expensive values for d .

The first experiment is done to investigate the effect of the choice of feature initialization methods on relatively small graphs with a density of about 0.5: $\mathcal{V} \sim \mathcal{N}(200, 5), \mathcal{D} \sim \mathcal{U}(0.48, 0.52)$. We set the MQCP input parameter $\gamma = 0.8$ during training and evaluation. The results obtained on 20 random graphs and on benchmark instance *brock_200_2*, which has a similar size and density, are shown in Figure 6.5. Moreover, the results on random graphs show that node features E1+E2, E1+S2V performed best, followed by E1, N2V, N2V+S2V. All trained models were able to obtain a solution of size 24 on the benchmark instance, which is the best known solution for this MQCP instance as reported in [PWWW21].

To investigate the performance of the trained models on graphs of even higher density,

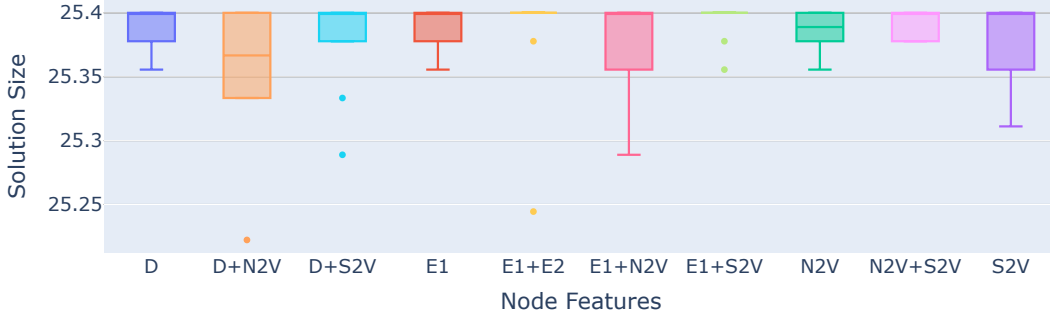
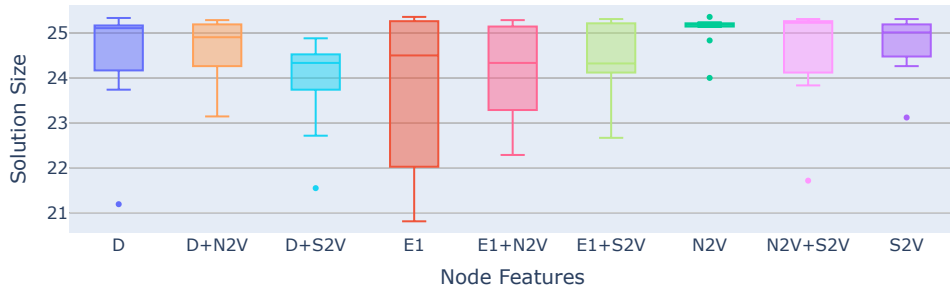


Figure 6.5: Results for uniform random graphs with $\mathcal{V} \sim \mathcal{N}(200, 5)$, $\mathcal{D} \sim \mathcal{U}(0.48, 0.52)$, MQCP parameter $\gamma = 0.8$, grouped by average solution quality per model on unseen test instances.

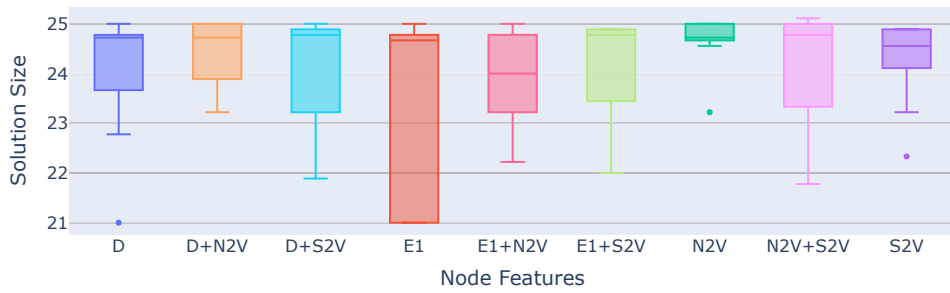
we target benchmark instance groups *brock400*- i , $i \in \{1, 2, 3\}$ with $|V| = 400$, $\text{dens}(G) \approx 0.75$, and *frb30-15*- i , $i \in \{1, \dots, 5\}$ with $|V| = 450$, $\text{dens}(G) \approx 0.823$. Again, we generate 20 random instances of similar size and density for both instance groups and evaluate the trained models by the average results obtained on the random graphs and the benchmark instances. The results are shown in Figure 6.6. Note that we removed the node feature E1+E2, as the 2-hop EgoNet for graphs of this density is already G itself in most cases, and thus no additional information is provided. The results show that the learning-based features N2V, S2V, perform well in terms of solution quality on random instances and on the *brock400* instance group compared to the other feature initialization methods. Interestingly, E1 is among the lowest-performing feature initialization methods for the *brock400* instance group and corresponding random graphs, but among the highest-performing for random graphs generated corresponding to the *frb30-15* instance group. Furthermore, we notice that the solution sizes obtained on the *frb30-15* benchmark instances are considerably smaller than those obtained on random instances, as these instances are generated differently than the uniform random graphs used to train the models. This indicates that using uniform random graphs during training does not generalize well to structurally different graph instances.

We draw two conclusions from the results of the computational experiments presented in this Section. First, all the presented feature initialization methods can effectively be used within the context of our algorithm, as most of the trained models improve significantly throughout the training, and no method clearly outperforms all other methods. Second, the differences between the evaluated feature initialization methods regarding the obtained solution quality seem to be only marginal. We explain this observation by arguing that feature initialization is only the first stage when evaluating the graph, and the main part of the computation is done by the application of the attention-based GNN.

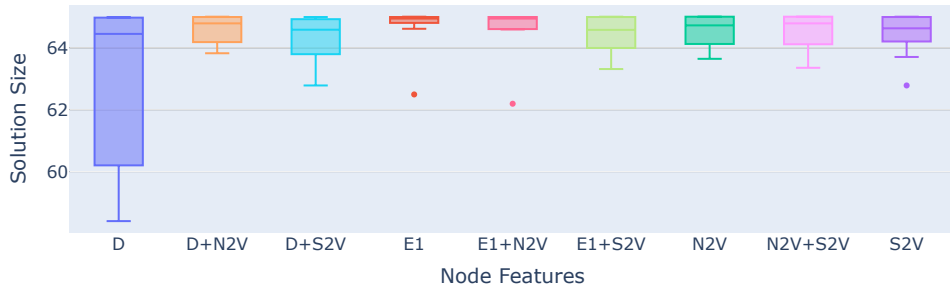
6. COMPUTATIONAL EXPERIMENTS & EVALUATION



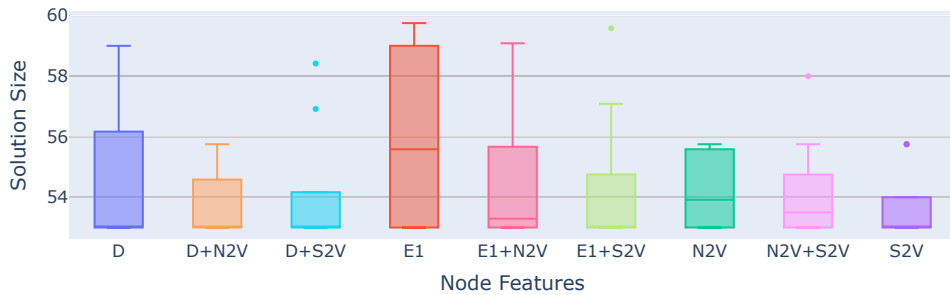
(a) Results on random graphs with $\mathcal{V} \sim \mathcal{N}(400, 10)$, $\text{dens}(G) \sim \mathcal{U}(0.74, 0.76)$, $\gamma = 0.999$.



(b) Results on DIMACS benchmark instances $\text{brock400}_i, i \in \{1, 2, 3\}$, $\gamma = 0.999$.



(c) Results on random graphs with $\mathcal{V} \sim \mathcal{N}(450, 10)$, $\text{dens}(G) \sim \mathcal{U}(0.82, 0.825)$, $\gamma = 0.95$.



(d) Results on benchmark instances $\text{frb30-15-}i, i \in \{1, \dots, 5\}$, $\gamma = 0.95$.

64 Figure 6.6: Impact of feature initialization method on random graphs and selected benchmark instances.

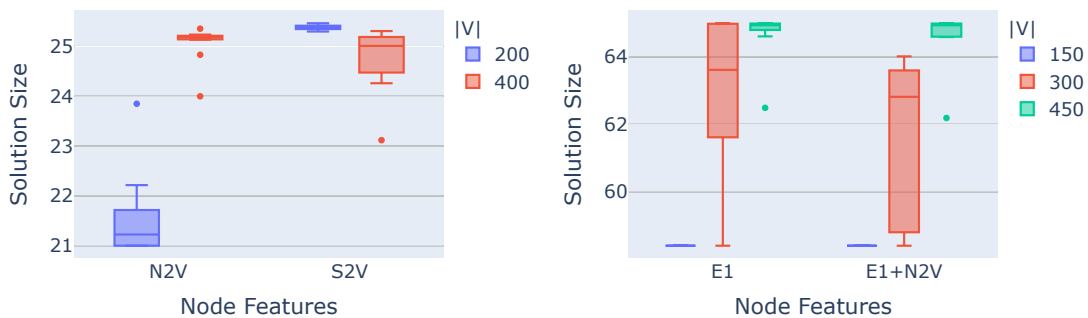
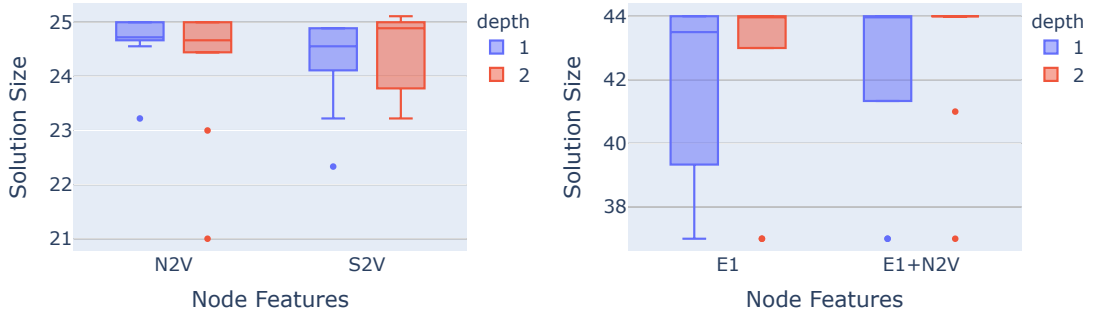


Figure 6.7: Results of training on smaller instances. The left Subfigure shows the effect of training on graphs of size $|V| \in \{200, 400\}$, evaluation on graphs of size 400. The right Subfigure shows training for $|V| \in \{150, 300, 450\}$, evaluation on graphs of size 450. On all instances, we set $\gamma = 0.999$.

6.2.3 Generalization to Larger Instances

Training the GNN-based scoring function is time-consuming, especially when larger graphs are used during training. To keep our approach scalable, we investigate, whether the trained models generalize to larger instances with similar densities to the instances seen during training. Figure 6.7 shows the results of our experiments. We select the two best-performing node features for two instance groups from Subsection 6.2.2, namely $|V| = 400, \text{dens}(G) \approx 0.75$ and $|V| = 450, \text{dens}(G) \approx 0.823$ with $\gamma = 0.999$ and compare, how models that were trained on smaller graphs perform on random instances with these properties. The left Subfigure shows the results for models that were trained on graphs of size $|V| = 200$ in comparison with models trained on graphs of size $|V| = 400$. While N2V did not generalize to bigger instances, S2V even outperformed the best results of models trained on the size of instances seen during evaluation. In the right Subfigure we see that the models with feature initialization methods E1, E1+N2V did not generalize well to larger graphs. The GNNs trained on graphs of size $|V| = 150$ show the worst performance, whereas the best models that were trained on graphs of size $|V| = 300$ were able to match the performance of the models trained on graphs of size $|V| = 450$.

We conclude that, at its current state, our approach does not generalize reliably to instances that are larger than those seen during training by a factor of two or more. Finally, we note that future work shall evaluate curriculum learning strategies as discussed in [LAT20], where the NNs are trained by starting on smaller instances and gradually increasing the instance size. The authors show that these strategies are successful in making the trained models achieve better generalization.

(a) Results for *brock400* instances.(b) Results for instance *C250*.Figure 6.8: Impact of look-ahead depth d on the average solution quality of the trained models ($\gamma = 0.999$).

6.2.4 Look-ahead Search Parameters

To evaluate the impact of the look-ahead depth, we compare models trained with a look-ahead depth $d = 1$, which corresponds to imitating a search of the Ω_1 neighborhood structure, to models trained with a look-ahead depth $d = 2$ and a restricted neighborhood size of $k' = 50$. Due to computational budget limitations, we set the number of generated training samples per iteration $n_s = 25$ to balance the increase in runtime.

We train ten models each for the two best-performing node features for the *brock400* instances, and for instance *C250* with lookahead-depth $d \in \{1, 2\}$. The results are shown in Figure 6.8. The difference in average solution quality of the trained models is only marginal on the *brock400* instances seen in the left Subfigure, but the right Subfigure shows that the models trained using a depth $d = 2$ achieve a better average solution quality on instance *C250*.

The results depicted in the figure are representative of other similar experiments we conducted. We conclude that using a higher look-ahead depth increases in most cases the average solution quality of the trained models, as expected, which justifies the increased computational effort during training.

To compare the impact on the runtime when using a look-ahead depth $d = 2$ compared to $d = 1$, we observe the average runtime in an iteration of the training loop that is used to generate training samples in both parameter settings for the trained models in this Subsection. Whereas 50 samples with look-ahead depth $d = 1$ are generated in less than one second per iteration, the average runtime per iteration to compute 25 training samples using a look-ahead depth $d = 2$ is 24.32 seconds, introducing a noticeable computational overhead throughout $z = 300$ iterations of the training loop.

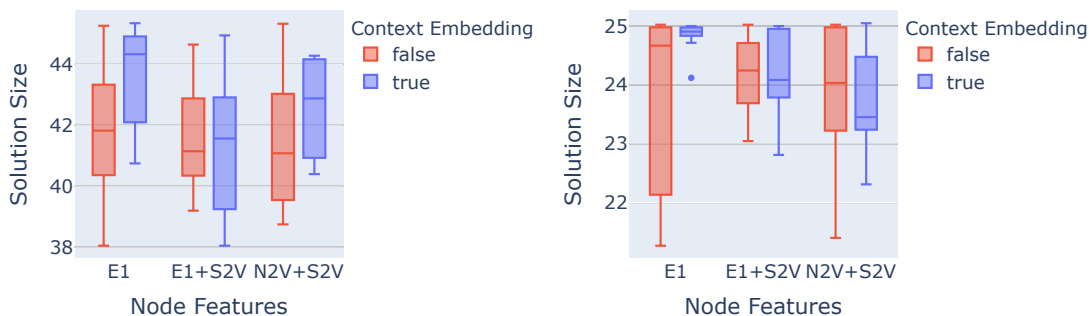


Figure 6.9: Impact of using the context embedding for MQCP instances with $|V| = 250$, $\text{dens} = 0.9$, $\gamma = 0.999$ (left) and $|V| = 400$, $\text{dens} = 0.75$, $\gamma = 0.999$ (right).

6.2.5 Impact of the Context Embedding

As previously noted, we identified the context embedding to be a weakness of our approach, as it compresses the information given by the candidate solution S too much by simply taking the mean of the node embeddings of the vertices in S . Therefore, we conduct additional experiments to investigate, whether the context embedding as defined in Section 5.4 has a positive effect on the performance, or if the same solution quality can be achieved by removing these context embeddings and only appending the features obtained from the d_S -values to the node embeddings as inputs for the decoder.

We perform two test runs by training ten GNNs with and without the context embedding for three different feature initialization methods on randomly generated MQCP instances with $|V| = 250$, $\text{dens} = 0.9$, $\gamma = 0.999$ and $|V| = 400$, $\text{dens} = 0.75$, $\gamma = 0.999$. The results, evaluated on independent random instances of similar size and density, are shown in Figure 6.9. For feature initialization method E1 we notice a clear improvement in solution quality in both test runs if the context embedding is used during the training and the execution of the algorithm. The GNNs using E1+S2V and the context embedding show slightly better performance in terms of solution quality when compared to those that do not use the context embedding. The results for the models trained with N2V+S2V are inconclusive, as those that use the context embedding perform better on the instances with $|V| = 250$, whereas those trained without the context embedding perform better on the instances with $|V| = 400$. We conclude that using the context embedding produces better results in most cases. Nonetheless, using a stronger context can certainly lead further to improvements in our approach.

6.3 Evaluation of Search Parameters

In this Section, we discuss the used parameter settings of LSBM. First, we implemented both considered short-term memory mechanisms, Tabu List and Configuration Checking,

and used the recommended settings presented in [ZBW20] for the tabu tenure, and in [CCP⁺21] for the Configuration Checking specific parameter *ub_threshold*. As we did not notice any significant differences in terms of solution quality between the two short-term memory mechanisms within our algorithm, we decided to use a Tabu List approach, as the tabu values for the vertices can be updated in time $O(1)$ after a swap, whereas the update in Configuration Checking takes $O(n)$ time, as all neighbors of the swapped vertices need to be updated as well.

Regarding the LSBM parameters τ, η, ξ , we decide to use a time limit $\tau = 300.0$, a number of iterations without improvement until the local search procedure is restarted $\eta = 2000$, and a maximum number of restarts $\xi = 10$, as increasing the values beyond this point did only increase the runtime of LSBM, but not the solution quality. The parameter $b = 0.3$ for the construction heuristic is set to the recommended value of the similar construction heuristic presented in [CCP⁺21], and we use the parameter $\alpha = 0.25$ for the GRASP-like construction step. Furthermore, we use the parameter settings $\beta = 10, \varepsilon = 10$ for the lower bound heuristic, as discussed in Section 6.1.

6.3.1 Neighborhood Size

We evaluate the effect on runtime and solution quality of differently sized neighborhoods during the execution of LSBM. We consider the values $k' \in \{10, 20, 30\}$. The GNNs are trained using the exact LSBM-T parameters as stated in Section 6.2 and use a neighborhood size $k' = 20$ during training. Figure 6.10 shows the obtained solution qualities and runtimes for randomly generated instances with $|V| = 378, \text{dens}(G) = 0.92, \gamma = 0.999$. We observe that the solution quality increase when using a larger neighborhood, which is to be expected. Furthermore, the runs with $k' = 10$ have in general a longer runtime than those with $k' = 20$, which is an unintuitive result. This is most likely because fewer swaps are performed on average with $k' = 20$ compared to $k' = 10$, until a feasible solution is found. We conclude that a neighborhood size smaller than $k' = 20$ shall not be used in the context of LSBM.

6.3.2 Sparse evaluation

As discussed in Section 5.5 the use of a GNN-based scoring function introduces a significant computational overhead. During our experiments, we notice that the runtime increases by an average factor between five and ten, depending on the input graph size and density and choice of node features, when using g_S during the execution of LSBM instead of d_S . This is primarily due to the decoder that evaluates the vertices in the graph every time the candidate solution changes. To reduce this overhead, we investigate, how runtime and solution quality are affected if the decoder is only re-applied if the previous vertex swap did not yield a solution with an increased objective value. We call this variation *sparse evaluation*.

We conduct two experiments on MQCP instances of different sizes and densities. We trained ten GNNs for three different feature initialization methods with a look-ahead

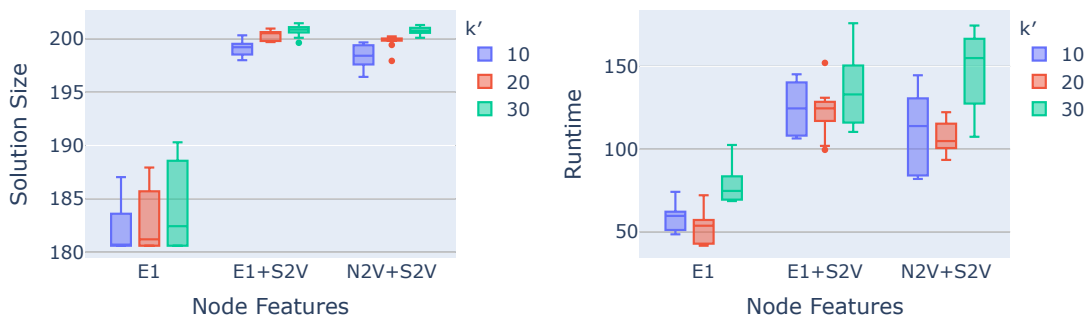


Figure 6.10: Impact of neighborhood size k' on the average solution quality and runtime in seconds using trained models for $|V| = 378$, $\text{dens}(G) = 0.92$, $\gamma = 0.999$.

depth $d = 2$. The remaining parameter settings are as discussed in Section 6.2. Figure 6.11 shows the results of the experiment for 20 randomly generated MQCP instances with size $|V| = 250$, $\text{dens}(G) = 0.9$, $\gamma = 0.999$. There is only a noticeable difference in solution quality in the test runs of the models using the feature initialization method N2V+S2V. However, the results seem unintuitive as the runtimes are slightly higher with sparse evaluation. We explain this observation by the fact that LSBM needs more iterations of the local search procedure, i.e., more swaps occur, when sparse evaluation is used, which balances out the gain in efficiency. In Figure 6.12 the results for 20 randomly generated MQCP instances with size $|V| = 300$, $\text{dens}(G) = 0.425$, $\gamma = 0.8$ are shown. Here, we notice a positive effect on the runtime when using sparse evaluation, while the obtained solution quality is similar for all trained models. We note that due to the lower density, the instances are generally easier to solve.

In conclusion, the presented method can reduce the computational effort on less dense instances, but we do not observe a significant improvement in terms of runtime on high-density instances.

6.4 Results on Benchmark Instances

For the final experiments on the benchmark instances, we choose one centrality-based node feature (E1), one learning-based node feature (N2V+S2V), and one node feature that combines both approaches (E1+S2V), as these features have shown to be promising during our evaluation. We use a neighborhood size $k' = 30$ and look-ahead search parameters $d = 2$, $k^* = 50$. The remaining parameter settings during training are exactly as described in Section 6.2. For each evaluated benchmark graph G , we train ten randomly initialized GNNs on independent random instances defined by $\mathcal{V} \sim \mathcal{N}(n, \frac{n}{100})$, $\mathcal{D} \sim \mathcal{U}(\text{dens}(G) - \frac{\text{dens}(G)}{100}, \text{dens}(G) + \frac{\text{dens}(G)}{100})$ and the problem-specific parameter γ (MQCP) or s (MDCP). After the training is completed, we adopt the GNN that yields the best results on 20 unseen uniform random graphs of similar size and density as those seen

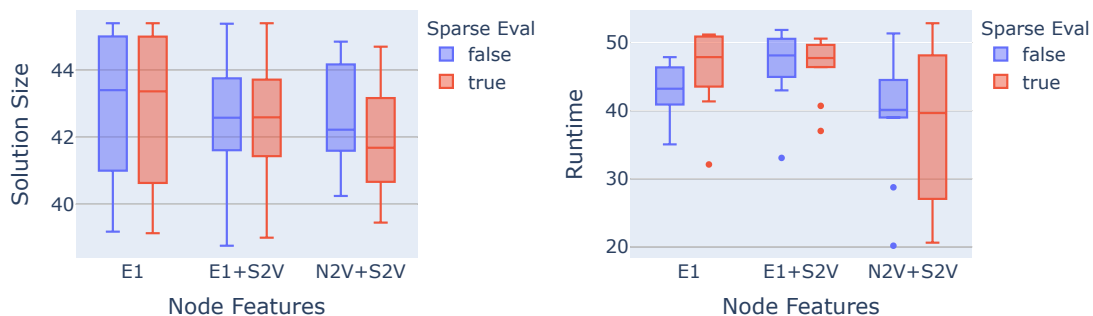


Figure 6.11: Impact of sparse evaluation on solution quality and runtime in seconds for MQCP instances with $|V| = 250$, $\text{dens}(G) = 0.9$, $\gamma = 0.999$

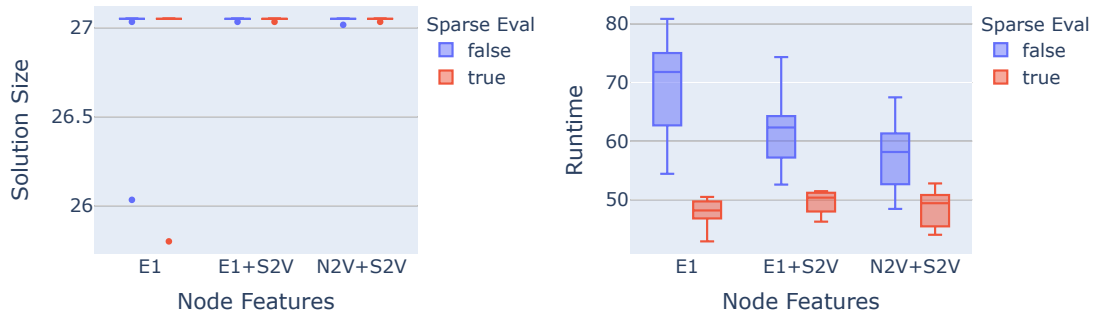


Figure 6.12: Impact of sparse evaluation on solution quality and runtime in seconds for MQCP instances with $|V| = 300$, $\text{dens}(G) = 0.425$, $\gamma = 0.8$

during training.

The results regarding the MQCP instances are shown in Table 6.4. We perform ten runs for each of the evaluated models and show the size of the best found solution, the average solution quality and the average runtime. The rightmost column contains the size of the best known solutions as reported by the most recent publications on the MQCP, [CCP⁺21] and [PWWW21]. The three considered variants of our algorithm, LSBM (E1), LSBM (N2V+S2V), and LSBM (E1+S2V) can match 17, 17 and 18 of the best known results out of the 26 evaluated instances, respectively. Again, this indicates that the impact of the chosen feature initialization method is only marginal. We notice that, especially on larger instances with high density (e.g., the *brock400* instances, or *MANN_a27*), our approach is not yet competitive with the state-of-the-art in terms of solution quality. A determining factor in this regard is most likely that our approach uses uniform random graphs during training, and some of these benchmark instances are created artificially to include certain structural properties which are not commonly seen

in randomly generated graphs. Furthermore, we do not observe any significant differences in runtime for the three variants of LSBM.

Table 6.5 shows the results on MDCP instances. The rightmost column shows the reported optimal values from [CZHX21] for reference. LSBM finds optimal solutions on most of the evaluated instances, except for the instances *keller4* and *san200_0.7_2*. We observe that the adaption to the MDCP is successful, as we notice similar results as with MQCP instances with high γ -values.

6. COMPUTATIONAL EXPERIMENTS & EVALUATION

Benchmark	Instance	V	E	dens(G)	γ	LSBM (E1)			LSBM (N2V+S2V)			LSBM (E1+S2V)			
						Best	Avg	T(s)	Best	Avg	T(s)	Best	Avg	T(s)	Best Known
DIMACS	brock200_1	200	14834	0.745	0.800	114	114.0	49.46	114	114.0	41.21	114	114.0	42.42	114
DIMACS	brock200_2	200	9876	0.496	0.800	24	24.0	57.46	24	24.0	52.99	24	24.0	55.82	24
DIMACS	brock200_3	200	12048	0.605	0.800	41	41.0	26.14	41	41.0	51.50	41	41.0	32.39	41
DIMACS	brock400_1	400	59723	0.748	0.999	25	24.9	23.23	25	25.0	23.61	25	25.0	24.10	27
DIMACS	brock400_2	400	59786	0.749	0.999	25	25.0	26.11	25	25.0	26.19	26	25.2	26.82	29
DIMACS	brock400_3	400	59681	0.748	0.999	27	25.4	26.23	25	24.9	26.12	27	26.2	27.99	31
DIMACS	C125	125	6963	0.898	0.999	34	34.0	7.308	34	34.0	7.32	34	34.0	7.05	34
DIMACS	C250	250	27984	0.899	0.999	44	44.0	24.28	44	44.0	32.26	44	43.4	24.43	44
DIMACS	C500	500	112332	0.900	0.999	58	57.2	25.13	55	45.4	24.73	43	43.0	22.49	58
DIMACS	c-fat200-5	200	8473	0.426	0.5	148	148.0	18.70	148	148.0	13.76	148	148.0	24.04	148
DIMACS	DSJC500	500	62624	0.502	0.800	34	34.0	29.44	34	34.0	31.68	34	34.0	28.03	34
DIMACS	hamming6-2	64	1824	0.905	0.950	37	37.0	9.10	37	37.0	8.26	37	37.0	8.11	37
DIMACS	hamming6-4	64	704	0.349	0.500	32	32.0	7.01	32	32.0	8.87	32	32.0	7.02	32
DIMACS	johnson8-4-4	70	1855	0.768	0.800	43	43.0	5.58	43	43.0	6.88	43	43.0	9.65	43
DIMACS	johnson16-2-4	120	5460	0.765	0.800	34	34.0	8.56	34	34.0	13.02	34	34.0	10.99	34
DIMACS	MANN_a9	45	918	0.927	0.999	16	16.0	5.73	16	16.0	5.82	16	16.0	5.83	16
DIMACS	MANN_a27	378	70551	0.990	0.999	130	130.0	43.02	130	130.0	36.9	130	130.0	44.12	135
DIMACS	p_hat300-1	300	10933	0.244	0.5	62	62.0	15.58	62	62.0	15.98	62	62.0	15.08	64
DIMACS	p_hat300-2	300	21928	0.489	0.8	114	114.0	19.33	114	114.0	25.26	114	114.0	34.33	114
DIMACS	san200_0_7_1	200	13930	0.700	0.95	55	55.0	20.94	55	55.0	10.65	55	55.0	21.08	57
DIMACS	san200_0_7_2	200	13930	0.700	0.95	31	31.0	18.24	31	31.0	18.55	31	31.0	18.75	34
DIMACS	san200_0_9_3	200	17910	0.900	0.999	44	44.0	31.37	44	44.0	20.19	44	44.0	24.31	44
BHOSLIB	fb30-15-1	450	83198	0.824	0.95	56	55.4	76.23	60	60.0	42.23	60	60.0	51.41	60
BHOSLIB	fb30-15-2	450	83151	0.823	0.95	57	56.3	82.10	54	54.0	43.72	58	58.0	56.15	58
BHOSLIB	fb30-15-4	450	83194	0.823	0.95	61	60.6	90.67	61	60.3	84.17	61	61.0	80.89	61
BHOSLIB	fb30-15-5	450	83231	0.824	0.95	60	60.0	89.20	60	60.0	65.17	60	53.6	50.76	60

Table 6.4: MQCP results for selected benchmark instances.

Benchmark	Instance	V	E	dens(G)	s	LSBM (E1)			LSBM (N2V+S2V)			LSBM (E1+S2V)			Opt
						Best	Avg	T(s)	Best	Avg	T(s)	Best	Avg	T(s)	
DIMACS	brock200_1	200	14834	0.745	1	21	21.0	9.49	21	21.0	10.12	21	21.0	9.68	21
					2	22	22.0	12.03	22	22.0	11.32	22	22.0	10.41	22
					1	12	12.0	14.30	12	11.6	13.42	12	12.0	9.92	12
					2	12	12.0	9.09	12	12.0	11.41	12	12.0	10.56	12
DIMACS	brock200_2	200	14834	0.745	2	13	12.8	7.83	13	13.0	16.52	13	13.0	13.15	13
					3	13	13.0	9.51	13	13.0	9.16	13	13.0	12.44	13
					4	13	13.0	10.01	15	15.0	10.69	15	15.0	12.22	15
					1	16	16.0	11.96	16	16.0	20.03	15	16.0	15.16	16
DIMACS	brock200_3	200	12048	0.605	2	16	16.0	10.19	16	16.0	11.51	16	16.0	13.22	16
					3	16	16.0	10.19	16	16.0	11.51	16	16.0	13.22	16
					4	17	17.0	18.31	17	17.0	13.54	17	17.0	16.14	17
					1	35	35.0	5.91	35	35.0	7.64	35	35.0	7.04	35
DIMACS	C125	125	6963	0.898	2	36	36.0	6.72	36	36.0	5.92	36	36.0	7.36	36
					3	37	37.0	9.13	37	37.0	6.76	37	37.0	7.41	37
					4	38	38.0	9.22	38	38.0	6.78	38	38.0	7.09	38
					1	58	58.0	6.57	58	58.0	10.10	58	58.0	8.89	58
DIMACS	c-fat200-5	200	8473	0.426	2	58	58.0	9.31	58	58.0	9.27	58	58.0	5.31	58
					3	58	58.0	9.63	58	58.0	6.95	58	58.0	8.19	58
					4	58	58.0	8.51	58	58.0	7.88	58	58.0	7.95	58
					1	10	58.0	6.86	10	10.0	7.62	12	10.8	9.76	12
DIMACS	keller4	171	9435	0.649	2	11	9.8	7.27	10	10.0	7.42	11.4	13.0	9.05	13
					3	14	11.8	7.96	14	14.0	9.52	14	14.0	10.90	14
					4	15	14.2	7.82	15	15.0	17.27	15	15.0	17.45	15
					1	17	17.0	3.23	17	17.0	3.10	17	17.0	2.76	17
DIMACS	MANN_a9	45	918	0.927	2	18	18.0	3.03	18	18.0	3.02	18	18.0	3.03	18
					3	19	19.0	3.03	19	19.0	3.06	19	19.0	3.34	19
					4	20	20.0	3.39	20	20.0	4.15	20	20.0	2.67	20
					1	16	15.9	8.34	17	16.3	9.24	15	14.6	9.72	19
DIMACS	sam200_0.7_2	200	13930	0.700	2	17	17.0	14.42	18	17.8	15.21	17	15.8	9.84	19
					3	18	17.4	12.01	18	18.0	14.16	17	16.8	12.28	20
					4	19	18.4	16.80	19	19.0	13.12	17.8	18.0	8.93	20
					1	19	18.4	16.80	19	19.0	13.12	17.8	18.0	8.93	20

Table 6.5: MIDCP results for selected benchmark instances.

Conclusions & Future Work

In this thesis, we presented a local search-based metaheuristic algorithm LSBM for edge-based relaxations of the MCP, namely, the MQCP and the MDCP, that utilizes a GNN-based scoring function to guide the local search. We developed a BS-based heuristic that is used to quickly obtain a feasible solution, and as an initial lower bound for the optimal objective value within LSBM. The structure of LSBM is built upon state-of-the-art heuristic algorithms ([DHB19], [ZBW20], and [CCP⁺21]) and is adapted to incorporate a GNN to extract and use structural information obtained from the input graph during its execution. Initially, a (possibly infeasible) candidate solution S of fixed size k is generated by a construction heuristic, which is then improved by a local search procedure centered around swapping pairs of vertices in and outside S to increase the objective value of the current candidate solution. If the candidate solution becomes a feasible solution, k is increased and the process is repeated until a stopping criterion is met. In the context of this algorithm, the GNN is used to restrict the neighborhood of a candidate solution to only the most promising vertices in and outside the current candidate solution.

As the main contribution of this work, we proposed an algorithm LSBM-T that can be used to generate training data and train the GNN. Training is done in an offline setting on representative problem instances using principles from imitation learning to imitate the exhaustive search of a user-defined neighborhood structure with respect to a candidate solution S . The results of our evaluation show that LSBM-T successfully trains the models to imitate the exhaustive search of a neighborhood.

As the input graphs for the considered problems are non-attributed, we investigated node feature initialization methods within the context of our algorithm, namely, degree-based initialization, EgoNet initialization, and the learning-based methods Node2Vec and Struc2Vec. We thoroughly evaluated, which feature initialization methods and combinations thereof can be used. Our computation experiments show that, at the

current state of our work, the differences in node feature initialization methods are only marginal.

We employed an autoregressive Encoder-Decoder-based GNN architecture inspired by the work shown in [KvHW19] to make the application of a NN feasible within our approach. The computationally expensive attention-based encoder, which utilizes GATv2 attention layers [BAY21], is only applied once per instance to produce node embeddings, whereas the lightweight decoder is applied after each change in the candidate solution. We are aware of the computational overhead introduced by the use of a NN, as the analysis of the additional computational effort is part of this work.

We evaluate selected benchmark instances for the MQCP and MDCP from the literature and compare our approach to the state-of-the-art. The results for the MQCP show that in terms of solution quality, the performance of the models trained using LSBM-T gets close to the leading methods and can match most best known results on instances with $|V| < 300$. On larger instances, however, our approach is not yet competitive. Partly this can be explained by the fact that we only used uniform random graphs when training the GNNs, whereas many of the benchmark instances are artificially created to exhibit certain structural properties that are unlikely to be found in uniform random graphs. We expect training on representative instances to yield better results, as this is where the strength of our approach lies. We note that in terms of runtime, LSBM is outperformed by the leading methods, as the computational overhead introduced by the GNN is too large. However, all experiments were conducted on CPUs. Since GPUs are known to speed up computations involving GNNs and NNs in general by a significant factor, the application of a GPU could greatly reduce the runtime of our approach. Furthermore, the evaluation of MDCP-instances shows that our approach is successfully adapted to the MDCP and produces similar results on this related problem.

Finally, the results of our evaluation show that our approach performs well when evaluated on uniform random graphs and using the trained GNN-based scoring functions within our algorithm leads to a substantial improvement in terms of solution quality compared to using the scoring function d_S .

Future Work

There are several promising directions in which the work of this thesis can be continued. We conclude this thesis by outlining possible ways of continuing this work and discussing questions and challenges that remained unsolved during our work.

One great challenge for future work on this algorithm is to define a stronger context that is fully independent of the d_S -values of the current candidate solution. Instead, it shall rely solely on information obtained from the corresponding node embeddings of the vertices in the current candidate solution S . The difficulty here lies in defining a context of, if possible, constant size, that is expressive enough to provide information about candidate solutions of variable size. Furthermore, the context should ideally be computable in time $O(n)$ as it needs to be computed in each iteration of the local search

procedure. Defining a stronger context could also lead to a stronger generalization of our approach.

Future work shall also include the evaluation of a more fine-grained GNN architecture and training data generation that allows the GNN to predict *two* scores for each vertex, one for vertices that should be added, and one for vertices that should be removed from the candidate solution. Splitting these prediction tasks and finding ways to adapt the algorithm to incorporate these changes could potentially improve the performance of the utilized GNN within LSBM.

During our evaluation, we noticed the potential for the improvement of generalization to larger instances. In this context, future work shall evaluate curriculum learning strategies as evaluated in [LAT20], which might lead to better generalization.

All experiments in this thesis were executed on CPUs. However, GPUs are known to speed up computations involving GNNs and NNs in general by a significant factor. Therefore, it would be of interest to perform larger-scale experiments using GPUs to investigate how strongly this speedup affects the execution of LSBM and LSBM-T.

At the current state of our work, training on large, dense graphs ($|V| > 500$) is very time-consuming. Another continuation of this work is therefore the optimization of the training algorithm to make training on larger instances computationally feasible in practice. As already discussed, one approach is to improve the generalization to larger instances and train on small instances, where training is less time-consuming. Besides that, other, methods to reduce the computational effort can be investigated, e.g., the preprocessing and reduction of instances.

LSBM-T generates graph instances on the fly during training. In our experiments, all models were trained on uniform random graphs. Naturally, the question arises, of how other random graph models can be utilized in LSBM-T and how well the trained GNNs generalize to graphs drawn from other distributions. Furthermore, real-world datasets should be analyzed and evaluated to determine, whether structural patterns in these datasets can be exploited within the context of our algorithm.

Finally, we note that LSBM and LSBM-T can be adapted in a relatively straightforward manner to other MCP-relaxations by defining relevant neighborhood structures and the objective value of a candidate solution in the context of LSBM.

List of Figures

2.1	A maximum clique of size four.	7
2.2	Consider the graph G given in Subfigure 2.2a. The considered MQCP instance is defined by G consisting of a K_3 and a K_7 connected by an edge and $\gamma = 0.5$. Since G has a density $\text{dens}(G) = \frac{25}{45} \approx 0.56$, the maximum 0.5-quasi clique in G is V . Assume a lower bound of $k = 6$ is known, therefore the two vertices on the left are $k\gamma$ -peelable and preprocessing would remove all edges incident to these vertices, as can be seen in Subfigure 2.2b. Therefore, the density of this preprocessed graph would be reduced to $\frac{22}{45} \approx 0.49$ and the optimal solution V would not be preserved.	9
3.1	Visual representation of the convolution operation for a vertex in a graph. Information of adjacent vertices is aggregated to compute the updated representation.	13
3.2	Illustration of the definition of unnormalized transition probabilities for a random walk that just traversed the edge (t, v) and is computing the transition probabilities for its successor. Figure recreated from [GL16].	17
4.1	Flowchart of the structure of the Local Search Based Metaheuristic.	33
5.1	Flowchart of the structure of the training algorithm.	40
5.3	Evaluation of a graph $G = (V, E)$ with candidate solution $S = v_2, v_5$ by the Encoder-Decoder-based architecture.	46
6.1	Beam Search solution sizes and runtimes in seconds for instances with $ V = 500, \text{dens}(G) \in \{0.75, 0.9\}$	57
6.2	BS solution sizes and runtimes in seconds for instances with $ V = 1000, \text{dens}(G) \in \{0.75, 0.9\}$	58
6.3	Each line in the plots represents an independent run of the training algorithm LSBM-T. In each iteration, a random instance is generated and LSBM is performed using the scoring functions g_S and d_S , producing the best found solutions $S_{g_S}^*, S_{d_S}^*$, respectively. The gaps between these solutions are computed as $\frac{ S_{g_S}^* - S_{d_S}^* }{ S_{d_S}^* }$	61
		79

6.4	Comparison of solution quality of g_S and d_S . The average solution quality on 20 random instances by d_S is marked by the dashed line. LSBM is executed with the same parameters for g_S and d_S	62
6.6	Impact of feature initialization method on random graphs and selected benchmark instances.	64
6.7	Results of training on smaller instances. The left Subfigure shows the effect of training on graphs of size $ V \in \{200, 400\}$, evaluation on graphs of size 400. The right Subfigure shows training for $ V \in \{150, 300, 450\}$, evaluation on graphs of size 450. On all instances, we set $\gamma = 0.999$	65
6.8	Impact of look-ahead depth d on the average solution quality of the trained models ($\gamma = 0.999$).	66
6.9	Impact of using the context embedding for MQCP instances with $ V = 250, \text{dens} = 0.9, \gamma = 0.999$ (left) and $ V = 400, \text{dens} = 0.75, \gamma = 0.999$ (right).	67
6.10	Impact of neighborhood size k' on the average solution quality and runtime in seconds using trained models for $ V = 378, \text{dens}(G) = 0.92, \gamma = 0.999$	69
6.11	Impact of sparse evaluation on solution quality and runtime in seconds for MQCP instances with $ V = 250, \text{dens}(G) = 0.9, \gamma = 0.999$	70
6.12	Impact of sparse evaluation on solution quality and runtime in seconds for MQCP instances with $ V = 300, \text{dens}(G) = 0.425, \gamma = 0.8$	70

List of Tables

5.1	Worst-case time complexity of the main operations used during LSBM when using the scoring functions d_S and g_S	52
6.1	LSBM parameters	54
6.2	LSBM-T parameters	55
6.3	Benchmark instances used for experiments	59
6.4	MQCP results for selected benchmark instances.	72
6.5	MDCP results for selected benchmark instances.	73

List of Algorithms

4.1	General structure of LSBM	25
4.2	Extend a feasible solution	25
4.3	Lower Bound heuristic based on Beam Search	28
4.4	Construction Heuristic with focus on exploration	29
4.5	Local Search Procedure	32
4.6	Search restricted neighborhood	33
5.1	Training algorithm LSBM-T	39
5.2	Computation of target values	40
5.3	Initialize scoring function d_S	50

Bibliography

- [ACL01] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001.
- [ARS02] James Abello, Mauricio G. C. Resende, and Sandra Sudarsky. Massive quasi-clique detection. In Sergio Rajsbaum, editor, *LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002, Proceedings*, volume 2286 of *Lecture Notes in Computer Science*, pages 598–612. Springer, 2002.
- [AXSS19] Kenshin Abe, Zijian Xu, Issei Sato, and Masashi Sugiyama. Solving np-hard problems on graphs by reinforcement learning without domain knowledge. *CoRR*, abs/1905.11623, 2019.
- [BAY21] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *CoRR*, abs/2105.14491, 2021.
- [BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [BHB08] Mauro Brunato, Holger H. Hoos, and Roberto Battiti. On effectively finding maximal quasi-cliques in graphs. In Vittorio Maniezzo, Roberto Battiti, and Jean-Paul Watson, editors, *Learning and Intelligent Optimization*, pages 41–55, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [BRZ22] Immanuel M Bomze, Francesco Rinaldi, and Damiano Zeffiro. Fast cluster detection in networks by first order optimization. *SIAM Journal on Mathematics of Data Science*, 4(1):285–305, 2022.
- [BW06] S. Butenko and W.E. Wilhelm. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173(1):1–17, 2006.

- [CCK⁺21] Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *CoRR*, abs/2102.09544, 2021.
- [CCP⁺21] Jiejiang Chen, Shaowei Cai, Shiwei Pan, Yiyuan Wang, Qingwei Lin, Mengyu Zhao, and Minghao Yin. NuQClq: An Effective Local Search Algorithm for Maximum Quasi-Clique Problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, page 9, 2021.
- [CZHX21] Xiaoyu Chen, Yi Zhou, Jin-Kao Hao, and Mingyu Xiao. Computing maximum k-defective cliques in massive graphs. *Computers & Operations Research*, 127:105131, 2021.
- [DHB19] Youcef Djeddi, Hacene Ait Haddadene, and Nabil Belacel. An extension of adaptive multi-start tabu search for the maximum quasi-clique problem. *Computers & Industrial Engineering*, 132:280–292, June 2019.
- [DHD⁺19] Chi Thang Duong, Thanh Dat Hoang, Ha The Hien Dang, Quoc Viet Hung Nguyen, and Karl Aberer. On node features for graph neural networks. *CoRR*, abs/1911.08795, 2019.
- [DHM21] Victor-Alexandru Darvari, Stephen Hailes, and Mirco Musolesi. Solving graph-based public goods games with tree search and imitation learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 1739–1751. Curran Associates, Inc., 2021.
- [DKR⁺13] Matjaž Depolli, Janez Konc, Kati Rozman, Roman Trobec, and Dušanka Janežič. Exact parallel maximum clique algorithm for general and protein graphs. *Journal of Chemical Information and Modeling*, 53(9):2217–2228, Sep 2013.
- [DKZ⁺17] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *CoRR*, abs/1704.01665, 2017.
- [DSAA14] Ahmed Douik, Sameh Sorour, Mohamed-Slim Alouini, and Tareq Y. Al-Naffouri. On minimizing the maximum broadcast decoding delay for instantly decodable network coding. In *IEEE 80th Vehicular Technology Conference, VTC Fall 2014, Vancouver, BC, Canada, September 14-17, 2014*, pages 1–5. IEEE, 2014.
- [ER59] P. Erdős and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6, 1959.
- [FHdW89] C. Friden, A. Hertz, and D. de Werra. STABULUS: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42(1):35–44, Mar 1989.

- [For09] Santo Fortunato. Community detection in graphs. *CoRR*, abs/0906.0612, 2009.
- [FRS17] Daniel R. Figueiredo, Leonardo Filipe Rodrigues Ribeiro, and Pedro H. P. Saverese. struc2vec: Learning node representations from structural identity. *CoRR*, abs/1704.03165, 2017.
- [GIFC15] Timo Gschwind, Stefan Irnich, Fabio Furini, and Roberto Wol?er Calvo. Social Network Analysis and Community Detection by Decomposing a Graph into Relaxed Cliques. Working Papers 1520, Gutenberg School of Management and Economics, Johannes Gutenberg-Universität Mainz, December 2015.
- [GIP18] Timo Gschwind, Stefan Irnich, and Isabel Podlinski. Maximum weight relaxed cliques and russian doll search revisited. *Discrete Applied Mathematics*, 234:131–138, 2018. Special Issue on the Ninth International Colloquium on Graphs and Optimization (GO IX), 2014.
- [GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864. ACM, 2016.
- [GSR⁺17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.
- [GXLY22] Jian Gao, Zhenghang Xu, Ruizhi Li, and Minghao Yin. An exact algorithm with new upper bounds for the maximum k-defective clique problem in massive sparse graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10174–10183, 2022.
- [GY20] Shenshen Gu and Hanmei Yao. A pointer network based deep learning algorithm for maximum clique problem. In *2020 10th International Conference on Information Science and Technology (ICIST)*, pages 229–233, 2020.
- [Hås99] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, Mar 1999.
- [HLMP21] Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. *CoRR*, abs/2110.05291, 2021.
- [IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

- [JCRL21] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning TSP requires rethinking generalization. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPICs*, pages 33:1–33:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- [JDHB22] Yan Jin, John H. Drake, Kun He, and Una Benlic. Reinforcement learning based coarse-to-fine search for the maximum k-plex problem. *Applied Soft Computing*, page 109758, 2022.
- [Kar72] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.
- [KS11] Arash Khosraviani and Mohsen Sharifi. A distributed algorithm for γ -quasi-clique extractions in massive graphs. In Pit Pichappan, Hojat Ahmadi, and Ezendu Ariwa, editors, *Innovative Computing Technology*, pages 422–431, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [KvHW19] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.
- [LAT20] Michal Lisicki, Arash Afkanpour, and Graham W. Taylor. Evaluating curriculum learning strategies in neural combinatorial optimization. *CoRR*, abs/2011.06188, 2020.
- [LCK18] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 537–546, 2018.
- [LMA89] J. E. (Ned) Lecky, O. J. Murphy, and Richard Absher. Graph theoretic algorithms for the PLA folding problem. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 8(9):1014–1021, 1989.
- [Loc21] Carlo Lucibello and other contributors. Graphneuralnetworks.jl: a geometric deep learning library for the julia programming language, 2021.
- [MAY10] Noël Malod-Dognin, Rumen Andonov, and Nicola Yanev. Maximum cliques in protein structure comparison. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples*,

Italy, May 20-22, 2010. Proceedings, volume 6049 of *Lecture Notes in Computer Science*, pages 106–117. Springer, 2010.

- [MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.
- [Mok79] Robert J. Mokken. Cliques, clubs and clans. *Quality and Quantity*, 13(2):161–173, Apr 1979.
- [MPR21] Fabrizio Marinelli, Andrea Pizzuti, and Fabrizio Rossi. LP-based dual bounds for the maximum quasi-clique problem. *Discrete Applied Mathematics*, 296:118–140, June 2021.
- [OCP⁺16] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, page 1105–1114, New York, NY, USA, 2016. Association for Computing Machinery.
- [OPR13] AB Oliveira, A Plastino, and CC Ribeiro. Construction heuristics for the maximum cardinality quasi-clique problem. In *10th Metaheuristics International Conference. Singapore*, volume 84, 2013.
- [ORRH22] Fabio F. Oberweger, Gunther R. Raidl, Elina Ronnberg, and Marc Huber. A learning large neighborhood search for the staff rostering problem. In Pierre Schaus, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 300–317, Cham, 2022. Springer International Publishing.
- [PAS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’14, New York, NY, USA - August 24 - 27, 2014*, pages 701–710. ACM, 2014.
- [PDFV05] Gergely Palla, Imre Derenyi, Illes Farkas, and Tamas Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, Jun 2005.
- [PPRR15] Bruno Q Pinto, Alexandre Plastino, Celso C Ribeiro, and Isabel Rosseti. A biased random-key genetic algorithm to the maximum cardinality quasi-clique problem. In *eleventh metaheuristics international conference*, pages 1–4, 2015.

- [PRRP18] Bruno Q. Pinto, Celso C. Ribeiro, Isabel Rosseti, and Alexandre Plastino. A biased random-key genetic algorithm for the maximum quasi-clique problem. *European Journal of Operational Research*, 271(3):849–865, December 2018.
- [PRRR21] Bruno Q. Pinto, Celso C. Ribeiro, José A. Riveaux, and Isabel Rosseti. A brkga-based matheuristic for the maximum quasi-clique problem with an exact local search strategy. *RAIRO-Operations Research*, 55:S741–S763, 2021.
- [PVBB13] Jeffrey Pattillo, Alexander Veremyev, Sergiy Butenko, and Vladimir Boginski. On the maximum quasi-clique problem. *Discrete Applied Mathematics*, 161(1-2):244–257, January 2013.
- [PWWW21] Bo Peng, Lifan Wu, Yang Wang, and Qinghua Wu. Solving maximum quasi-clique problem by a hybrid artificial bee colony approach. *Information Sciences*, 578:214–235, November 2021.
- [PYB13] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013.
- [RGR⁺22] Kristjan Reba, Matej Guid, Kati Rozman, Dušanka Janežič, and Janez Konc. Exact maximum clique algorithm for different graph types using machine learning. *Mathematics*, 10(1), 2022.
- [RR19] Celso C. Ribeiro and José A. Riveaux. An exact algorithm for the maximum quasi-clique problem. *International Transactions in Operational Research*, 26(6):2199–2229, November 2019.
- [RZA17] Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed. Deep feature learning for graphs. *CoRR*, abs/1704.08829, 2017.
- [SCS06] Hanif D. Sherali and J. Cole Smith. A polyhedral study of the generalized vertex packing problem. *Mathematical Programming*, 107(3):367–390, Jul 2006.
- [SF78] Stephen Seidman and Brian Foster. A graph-theoretic generalization of the clique concept*. *Journal of Mathematical Sociology*, 6:139–154, 01 1978.
- [SLE21] Yuan Sun, Xiaodong Li, and Andreas Ernst. Using statistical measures and machine learning for graph reduction to solve maximum weight clique problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(5):1746–1760, 2021.
- [SS97] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

- [SST02] Hanif D. Sherali, J. Cole Smith, and Antonio A. Trani. An airspace planning model for selecting flight-plans under workload, safety, and equity considerations. *Transportation Science*, 36(4):378–397, 2002.
- [TBBB13] Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications*, 56(1):113–130, Sep 2013.
- [VCC⁺18] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
- [VPBP16] Alexander Veremyev, Oleg A. Prokopyev, Sergiy Butenko, and Eduardo L. Pasiliao. Exact mip-based approaches for finding maximum quasi-cliques and dense subgraphs. *Comput. Optim. Appl.*, 64(1):177–214, 2016.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.
- [WH13] Qinghua Wu and Jin-Kao Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *J. Comb. Optim.*, 26(1):86–108, 2013.
- [WPC⁺19] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.
- [YPTG06] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 02 2006.
- [ZBW20] Qing Zhou, Una Benlic, and Qinghua Wu. An opposition-based memetic algorithm for the maximum quasi-clique problem. *European Journal of Operational Research*, 286(1):63–83, October 2020.
- [ZCH⁺20] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.
- [ZSX⁺18] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *CoRR*, abs/1803.07294, 2018.

- [Zuc06] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 681–690, New York, NY, USA, 2006. Association for Computing Machinery.