INAUGURAL-DISSERTATION

zur

Erlangung der Doktorwürde

der

Gesamtfakultät für Mathematik, Ingenieur- und Naturwissenschaften

der

Ruprecht-Karls-Universität

Heidelberg

vorgelegt von

Großmann, Ernestine, M.Sc.

aus Freiberg, Germany

Tag der mündlichen Prüfung:

Data Reductions in Combinatorial Optimization for Independence Problems

Betreuer: Prof. Dr. Christian Schulz

Acknowledgements

First and foremost, I would like to thank my supervisor, Christian Schulz, for all the incredible opportunities he provided me during my time as a PhD student in the Algorithm Engineering group in Heidelberg. During that time, I learned how to program, conduct scientific research and had the chance to attend various conferences, symposia, and seminars. There, I met many inspiring people and learned about their interesting work. A special thanks also goes to my co-supervisor, Darren Strash. His work on the Independent Set problem has been a great source of inspiration for me. His support, especially at the beginning of my PhD journey, was very helpful. Furthermore, I am particularly grateful for the opportunity to undertake two extended research visits to Bergen in Norway. In that regard, I would like to thank everyone I had the pleasure of meeting during my stay. Particularly, I am thankful to Fredrik Manne and Kenneth Langedal for the invitation, the enriching discussions, and the new experiences during my stay.

I would also like to extend my gratitude to all my co-authors: Jannick Borowitz, Marcelo Fonseca Faraj, Tobias Heuer, Felix Joos, Sebastian Lamm, Kenneth Langedal, Thomas Möller, Jonas Sauer, Christian Schulz, Patrick Steil, Darren Strash, and Fabian Walliser. I am grateful to have had the opportunity to work with you and for everything I was able to learn along the way. Additionally, I would like to thank Adil Chhabra, Marcelo Fonseca Faraj, Henrik Reinstädtler, Christian Schulz, and Henning Woydt, my colleagues in the Algorithm Engineering group. I gained valuable knowledge thanks to several engaging conversations and collaborative projects. I have always looked forward to exploring new ideas with you.

Finally, I would like to thank my friends and family for their support throughout my PhD. Especially during the most challenging phases, you were always there for me and made even this time enjoyable. I am deeply grateful for all the moments with you that made this journey so much more fulfilling.

Abstract

The Maximum Weight Independent Set problem is a fundamental **NP**-hard problem in combinatorial optimization that additionally attracts interest because of its many real-world applications. The problem asks for a subset of vertices of heighest weight in an undirected vertex-weighted graph, such that the vertices in the set are pairwise non-adjacent. While this problem is the main focus of this thesis, we also consider other independence problems, including the Maximum (Weight) 2-Packing Set, and the Maximum Weight Hypergraph b-Matching problem. An important part of solving these problems in practice is data reduction rules. This dissertation introduces new data reduction rules as well as different approaches that utilize these rules for computing high-quality solutions.

For the Maximum Weight Independent Set problem, we contribute several new data reduction rules and a machine learning screening approach to speed up the application of these rules. Moreover, we present heuristics that find near-optimal solutions on static graphs and a new technique to compute high-quality independent sets in the dynamic setting. This technique is also applicable to the unweighted problem and as a local search on static graphs. We also introduce multiple new data reduction rules for the Maximum 2-Packing Set problem and its weighted generalization. Combining the new data reduction rules with a reduction to the Maximum (Weight) Independent Set problem, allows us to utilize existing independent set approache resulting in several mehods to compute high-quality 2-packing sets. Regarding the Maximum Weight b-Matching problem in hypergraphs, we present a set of new data reduction rules, a greedy strategy and a novel local search method to compute high-quality solutions.

All proposed approaches are discussed in detail and evaluated experimentally on a wide range of real-world benchmark instances. The experimental results indicate that for all of the problems discussed, using data reduction rules can significantly reduces the size of many input instances. This initial size reduction enables faster running times and can improve solution quality for all of our introduced approaches, but also for all other state-of-the-art methods tested. Furthermore, our experiments show that

our new approaches outperform state-of-the-art solvers in different metrics. Overall, this dissertation demonstrates the power of using data reduction rules in different ways to solve independence problems in practice.

Zusammenfassung

Das Problem eine stabile Menge mit größtem Gewicht in einem gegebenem Graphen zu finden ist ein grundlegendes **NP**-schweres Problem in der kombinatorischen Optimierung, welchem zusäzlich aufgrund seiner zahlreichen Anwendungen großes Interesse entgegenkommt. Dieses Schwerste Stabile Mengen-Problem besteht darin, für einen gegebenen, ungerichteten Graphen mit Knotengewichten eine Teilmenge der Knoten zu finden, sodass diese das höchstmögliche Gewicht hat. Dabei dürfern keine zwei Knoten in der Menge benachbart sein. Neben dem Schwerste Stabile Mengen-Problem, das im Hauptfokus dieser Arbeit steht, befasst sich diese Dissertation auch mit anderen Unabhängigkeitsproblemen, darunter das Grösste 2-Packing Menge-Problem, dessen gewichtete Veralgemeinerung das Schwerste 2-Packing Menge-Problem sowie das Schwerste b-Matching-Problem in Hypergraphen. Ein wichtiger Aspekt für die Lösung dieser Probleme sind Datenreduktionsregeln. Diese Dissertation stellt neue Datenreduktionsregeln für Unabhängigkeitsprobleme vor. Zusätzlich werden verschiedene Methoden eingeführt und detailiert diskutiert, welche diese Reduktionsregeln zur Berechnung qualitativ hochwertiger Lösungen verwenden.

Für das Schwerste Stabile Menge-Problem tragen wir neben neuen exakten Datenreduktionsregeln einen Screening-Ansatz bei, der mittels maschinellem Lernen die Reduktionsphase beschleunigen kann. Darüber hinaus präsentieren wir zwei Heuristiken, die nahezu optimale Lösungen auf statischen Graphen finden. Um das Problem für dynamische Graphen zu lösen stellen wir eine neue Technik vor, mit der qualitativ hochwertige Lösungen berechnet werden können. Diese Technik ist ebenfalls für das dynamische Grösste Stabile Menge-Problem geeignet, sowie auf statischen Graphen als Lokale Suche anwendbar. Wir führen auch mehrere neue Datenreduktionsregeln für das Grösste 2-Packing Menge-Problem und seine gewichtete Verallgemeinerung ein. Die Kombination dieser neuen Datenreduktionsregeln mit einer Reduktion auf das Grösste/Schwerste Stabile Menge-Problem ermöglicht es uns, bestehende Ansätze für stabile Mengen zu nutzen. Das führt zu mehreren Methoden, die qualitativ hochwertige 2-Packing Mengen berechnen können. In Bezug auf das Schwerste b-Matching-Problem in Hypergraphen präsentieren wir eine Reihe

neuer Datenreduktionsregeln und neue Greedy Strategien, sowie eine Lokale Suche zur Berechnung qualitativ hochwertiger Lösungen.

Alle vorgestellten Methoden werden ausführlich diskutiert und anhand einer Vielzahl realer Benchmark-Instanzen experimentell evaluiert. Diese experimentellen Ergebnisse zeigen, dass die Verwendung unserer Datenreduktionsregeln die Eingabegröße, von vielen Instanzen erheblich reduzieren kann, was bei allen hier eingeführten Algorithmen aber auch anderen getestete Methoden eine schneller Laufzeit und verbesserte Lösungsqualität ermöglicht. Darüber hinaus demonstrieren wir, dass unsere Ansätze die aktuell besten Methoden in verschiedenen Metriken übertreffen. Insgesamt zeigt diese Dissertation die Wichtigkeit und den starken Einfluss von Datenreduktionsregeln auf das Lösen von verschiedenen Unabhängigkeitsproblemen.

List of Publications

Related Publications for this Dissertation

Journal Articles

[1] Ernestine Großmann, Sebastian Lamm, Christian Schulz and Darren Strash. Finding Near-Optimal Weight Independent Sets at Scale. *Journal of Graph Algorithms and Applications*, 2024.

Journal Articles (submitted)

[2] Jannick Borowitz, Ernestine Großmann, Christian Schulz, Dominik Schweisgut. Finding Optimal 2-Packing Sets on Arbitrary Graphs at Scale. *Submitted to the Journal of Graph Algorithms and Applications.*

[3] Jannick Borowitz, Ernestine Großmann, Christian Schulz. Finding Maximum Weight 2-Packing Sets on Arbitrary Graphs. *Submitted to Networks.*

[4] Ernestine Großmann, Felix Joos, Henrik Reinstädtler, Christian Schulz. Engineering Algorithms for Hypergraph b-Matching. *Submitted to the Journal of Graph Algorithms and Applications.*

Conference Articles

[5] Ernestine Großmann, Sebastian Lamm, Christian Schulz and Darren Strash. Finding Near-Optimal Weight Independent Sets at Scale. *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2023.

[6] Jannick Borowitz, Ernestine Großmann and Christian Schulz. Optimal Neighborhood Exploration for Dynamic Independent Sets. *In Proceedings of the Conference Algorithm Engineering and Experiments (ALENEX)*, 2025.

Conference Articles (submitted)

[7] Ernestine Großmann, Kenneth Langedal, Christian Schulz. A New Concurrent Local Search for the Maximum Weight Independent Set Problem. *Submitted to the Symposium on Experimental Algorithms (SEA)*, 2025.

[8] Ernestine Großmann, Kenneth Langedal, Christian Schulz. Accelerating Reductions Using Graph Neural Networks for the Maximum Weight Independent Set Problem. *Submitted to the Conference on Applied and Computational Discrete Algorithms (ADCA)*, 2025.

Technical Reports

[9] Ernestine Großmann, Kenneth Langedal, Christian Schulz. A Comprehensive Survey of Data Reduction Rules for the Maximum Weighted Independent Set Problem. Technical Report, Heidelberg University, University of Bergen, 2024 (arXiv:2412.14198).

[10] Ernestine Großmann, Kenneth Langedal, Christian Schulz. Accelerating Reductions Using Graph Neural Networks and a New Concurrent Local Search for the Maximum Weight Independent Set Problem. Technical Report, Heidelberg University, University of Bergen, 2024 (arXiv:2412.14198).

Further Publications

Conference Articles

[11] Ernestine Großmann, Tobias Heuer, Christian Schulz, Darren Strash. The PACE 2022 Parametrized Algorithms and Computational Experiments Challenge: Directed Feedback Vertex Set. *In Proceedings of the 17th International Symposium on Parameterized and Exact Computation (IPEC 2022)*, 2022.

[12] Jannick Borowitz, Ernestine Großmann and Christian Schulz. Engineering Fully Dynamic Δ-Orientation Algorithms. *In Proceedings of the SIAM Conference on Applied and Computational Discrete Algorithms (ACDA)*, 2023.

[13] Ernestine Großmann, Jonas Sauer, Christian Schulz and Patrick Steil. Arc-Flags Meet Trip-Based Public Transit Routing. *In Proceedings of the Symposium on Experimental Algorithms (SEA)*, 2023.

[14] Marcelo Fonseca Faraj, Ernestine Großmann, Felix Joos, Thomas Möller and Christian Schulz. Engineering Weighted Connectivity Augmentation Algorithms. *In Proceedings of the Symposium on Experimental Algorithms (SEA)*, 2024.

[15] Ernestine Großmann, Henrik Reinstädtler, Christian Schulz and Fabian Walliser. Engineering Fully Dynamic Exact Δ-Orientation Algorithms. *In Proceedings of the Conference Algorithm Engineering and Experiments (ALENEX)*, 2025.

Technical Reports

[16] Ernestine Großmann, Jonas Sauer, Christian Schulz, Patrick Steil and Sascha Witt. Integrating Arc-Flags and Trip-Based Public Transit Routing. Technical Report, Heidelberg University, University of Bonn, Karlsruhe Institute of Technology, 2023 (arXiv:2312.13146).

Co-Supervised Theses

[1] Patrick Steil. Arc-Flags for Public Transit Routing. Bachelor thesis. Heidelberg University. 2022. https://schulzchristian.github.io/thesis/ba_steil.pdf

[2] Jannick Borowitz. Local Search with Optimal Neighborhood Exploration for the Maximum Weight Independent Set Problem. Bachelor thesis. Heidelberg University. 2022. https://schulzchristian.github.io/thesis/ba_borowitz.pdf

[3] Henrik Reinstädtler. Engineering Heuristics and Reductions for Weighted Hypergraph b-Matching. Master thesis. Heidelberg University. 2023. https://schulzchristian.github.io/thesis/ma_reinstaedtler.pdf

[4] Felix Wörner. Engineering Buffered Algorithms for NeighbourCover. Bachelor thesis. Heidelberg University. 2023. https://schulzchristian.github.io/thesis/ba_woerner.pdf

[5] Marie-Christin Litzinger. Edge Sparsification for Centrality Computation. Master thesis. Heidelberg University. 2023. https://schulzchristian.github.io/thesis/ma_litzinger.pdf

[6] Thomas Möller. Engineering Weighted Connectivity Augmentation Algorithms. Master thesis. Heidelberg University. 2023. https://schulzchristian.github.io/thesis/ma_moeller.pdf

[7] Konrad Straube. Engineering Data Reduction and Expansion Rules for the Maximum Weight Independent Set Problem. Master thesis. Heidelberg University. 2024. https://schulzchristian.github.io/thesis/ma_straube.pdf

[8] Fabian Walliser. Exploring Edge Orientation Algorithms in the Static and Dynamic Case. Master thesis. Heidelberg University. 2024. https://schulzchristian.github.io/thesis/ba_walliser.pdf

[9] Thanh Lan Tran. Finding 2-Packing Sets Using Hypergraph Matching. Bachelor thesis. Heidelberg University. 2024. https://schulzchristian.github.io/thesis/ba_tran.pdf

[10] Nick-Nikita Funk. Evolutionary Algorithm for the Weighted Connectivity Augmentation Problem. Bachelor thesis. Heidelberg University. 2024. https://schulzchristian.github.io/thesis/ba_funk.pdf

[11] Jannick Borowtz. Distributed Kernelization Techniques for the Maximum Weight Independent Set Problem. Master thesis. Karlsruhe Institute of Technology. 2025. https://publikationen.bibliothek.kit.edu/1000179280

Contents

Chapter 1

Introduction

In computer science, we often create abstract models of the world around us to understand our environment better and solve different problems efficiently. Graphs provide an elegant and versatile way to represent our environment, capturing relationships and interactions in a simple yet powerful structure. A graph consists of a set of vertices and edges. The vertices usually model entities, while the edges connect vertices and model the relationships between them. Scenarios that can be modeled with graphs appear everywhere in our day-to-day lives. In social networks, for example, vertices represent people, and their relationships are modeled as edges connecting them. We can also find these graph structures in transportation systems, where vertices describe cities connected via transportation infrastructure. These graphs can, however, also be found in very specialized areas, such as biological systems, where molecular interactions are analyzed. Once we can model different scenarios, we can start asking questions about them. For example, a company might be interested in finding highly influential people in a social network for their marketing strategy, or a country's power grid can be made more robust by finding weak spots in the grid. A navigation system finds a fast way between two places, A and B, by computations on a road network. To solve these problems on a computer, we can use graph models.

However, not all of these problems can be solved efficiently. In computer science, the difficulty of problems is categorized by complexity classes. The complexity class **P** contains all problems that can be solved in polynomial time. An example of a problem in this complexity class is finding the shortest path from A to B. The class **NP**, on the other hand, contains problems for which a solution can be verified in polynomial time. The other two examples of finding highly influential people in a social network and weak spots in a power grid belong to this complexity class.

It is still an open question whether **P** is equal to **NP**. However, it is widely believed that **P** is not equal to **NP**, meaning some problems will remain hard to solve.

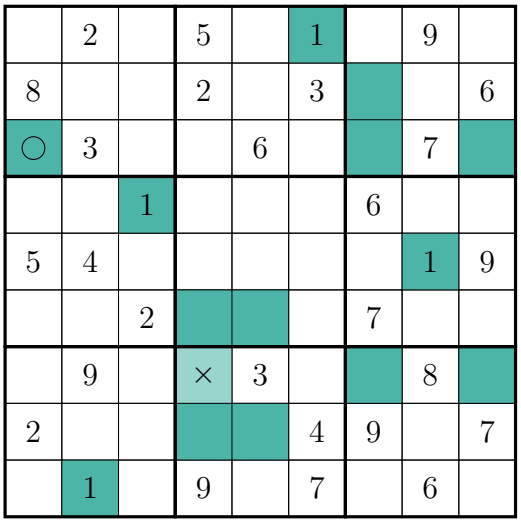


Figure 1.1: Sudoku example of applying different reduction rules.

In the following, we give an intuitive introduction to the concept of data reduction rules using the sudoku game, which is also a problem in **NP** [223]. Even though it is simple to check that a given sudoku solution is correct, solving large sudoku puzzles can be difficult. In a sudoku puzzle, we have given an $n^2 \times n^2$ grid, which is again divided into $n \times n$ boxes, illustrated with thick border lines. The goal of the game is to place the digits 1 to n^2 once in every row, column, and box. Typically, some of these numbers are given in the beginning to guarantee a unique solution. Figure 1.1 gives an example grid for $n = 3$. This is the most commonly known version, often found in newspapers.

To solve these puzzles, most people quickly devise efficient strategies to eliminate numerous options instead of trying all possible solutions. These strategies are the same as what we call data reduction rules. These data reduction rules typically reduce the problem instance efficiently. One very simple rule is the so-called *cross-hatching*. To apply the cross-hatching reduction, we look at the 3×3 boxes of the sudoku grid. In Figure 1.1, we highlight all possible cells where we can place the digit 1 without directly violating a sudoku rule. We see only one free cell in the top left box for the digit 1. Since all other free cells in that box already have a 1 in their row or column, we have to place the 1 in the cell marked with a circle. A more complicated reduction rule can be applied to the 7th row in combination with the bottom right box. Since the digit 1 has to go into this box, it can no longer go into any of the other blank cells in the 7th row of the sudoku grid outside of this box. Therefore, the digit 1 can not be placed in the cell marked with a cross.

In general, reduction rules are often used to help solve problems in **NP**. These rules are used to simplify the instance while preserving the optimal solution. This means

we can solve the reduced instance optimally after applying the reduction rules and then, with this solution, efficiently build an optimal solution to the original instance.

Data reduction rules are an important tool for parameterized complexity theory. Another complexity class of fixed-parameter tractable problems (**FPT**) is a research focus in this area. This complexity class contains problems that are generally difficult to solve. However, they can be solved efficiently when a problem parameter is small. There has been much theoretical work on fixed-parameter tractable algorithms and data reductions for many combinatorial optimization problems. However, there is still a big gap between the theoretical results and the practical applicability of these algorithms. In this dissertation, we aim to bridge this gap by focusing on the practical applicability of data reduction rules for different combinatorial optimization problems.

This dissertation focuses on data reduction rules specifically for independence problems. With this term, we group problems that ask for a subset of vertices or edges in an instance that fulfills a certain independence constraint. The most well-known independence constraint is that of the independent set, where no two vertices are adjacent. The 2-packing set constraint is a stronger variant, where two vertices have to have at least a distance of three in the graph. In matchings, we have independence constraints for edges, where no two edges share a vertex. The more general b-matching problem has a more relaxed independence constraint on the edges.

One graph problem where there has recently been much research in data reduction rules is the Maximum Independent Set and its generalization, the Maximum Weight Independent Set problem. Both are well-studied and fall into the category of independence problems. There, the task is to find the largest/highest-weight independent set in a graph. These problems and other closely related problems have several practical applications ranging from matching molecular structures to wireless networks [35]. Furthermore, Dong et al. [60] introduced a new collection of instances based on real-life long-haul vehicle routing problems at Amazon. For this application, we have routs, where each route has a weight, a load, and a driver. We want to find a subset of vehicle routes of maximum weight such that no two routes share a driver or a load. To model this problem as a Maximum Weight Independent Set problem, we build a conflict graph where vertices correspond to routes, and vertex weights model the route weights. Furthermore, if two routes have a conflict, i. e., they share a driver or a load, then the corresponding two vertices are connected by an edge.

1.1 Main Contributions

In this dissertation, we focus on the Maximum Weight Independent Set problem and other independence problems such as the Maximum Cardinality 2-Packing Set, Maximum Weight 2-Packing Set and the Hypergraph b-Matching problem. For all of these problems, we present new exact data reduction rules and algorithms using these rules to solve the different problems.

The main contribution of this work, are the presented data reduction rules for independence problems along with efficient approaches using these rules to compute high-quality solutions. This includes new data reduction rules for the following three different problems: the Maximum Weight Independent Set (MWIS), the Maximum (Weight) 2-Packing (M2PS/MW2PS), and the Hypergraph-b-Matching (HBM) problem. Additionally, we engineer novel algorithms using these reductions, including exact and heuristic approaches. We evaluate our methods on various real-world instances and show that our approaches outperform state-of-the-art solvers. The remainder of this section gives a more details of our contributions.

1.1.1 Maximum Weight Independent Set

Our contribution for the Maximum Weight Independent Set problem consists of multiple elements. We list them in here, along with a brief overview.

Reduction Rules. We provide a comprehensive overview of existing data reduction rules for the Maximum Weight Independent Set problem. Additionally, we propose new data reduction rules and discuss the different data reduction rules used in various practical solvers. Moreover, we present the results of extensive experiments regarding the ordering of the reductions and devise a robust reduction ordering that performs well in practice.

The LearnAndReduce Approach. With the LearnAndReduce approach, we developed an advanced, exact preprocessing tool that employs Graph Neural Networks (GNNs) to decide where to apply data reduction rules. With our GNN filtering, we can now apply reduction rules that were previously unused in other practical works due to the computational cost required to determine their applicability [105]. With LearnAndReduce, we also present a new dataset with labeled vertices for the problem of early reduction rule screening. The dataset contains two collections of graphs, one with unreduced instances and one after running a set of fast reductions. The second is the one we use for the early reduction rule screening. It consists of

more than one million labeled vertices. Using LearnAndReduce, we can reduce the instances to within one percent of what is possible using the full set of reduction rules while being four times as fast. These results are adjusted for fast reduction rules that would always be applied.

A Memetic Algorithm. We develop a state-of-the-art memetic algorithm based on recombination operations employing graph partitioning techniques. Our algorithm computes large-weight independent sets by incorporating several recently developed advanced reduction rules. In particular, our algorithm uses a wide range of frequently used data reduction techniques [91, 107, 149].

The algorithm may be viewed as performing two functions simultaneously. First, it uses reduction rules for the Maximum Weight Independent Set problem to boost the performance of the memetic algorithm. Second, the memetic algorithm opens up the reduction space to further reduce by including vertices that are likely to be in large-weight independent sets. On a high level, this algorithm is a reduce-and-peel approach with a high-quality peeling strategy. This technique finds near-optimal weight independent sets much faster than existing local search algorithms. It is competitive with state-of-the-art *exact* algorithms for smaller graphs and allows us to compute large-weight independent sets on huge, sparse instances. Overall, our algorithm configurations compute the best results among all state-of-the-art algorithms for every instance and thus can be seen as the best tool when large weight independent sets need to be computed in practice.

A Concurrent Iterated Local Search Algorithm. We propose a new concurrent iterated local search heuristic CHILS to compute large-weight independent sets very fast. In particular, our heuristic works by alternating between the full graph and the Difference-Core (D-Core), a subgraph constructed using multiple heuristic solutions. With this heuristic and LearnAndReduce, we outperform existing heuristics across a wide variety of real-world instances. Our main results can be summarized as follows.

For a large set of real-world test instances accumulated over several years by earlier works, CHILS combined with LearnAndReduce finds the best solution across *all* test instances. On the new vehicle routing instances [60], we compare CHILS with two recent heuristics, METAMIS [59] and a Bregman-Sinkhorn algorithm [113], both designed specifically for these instances. In contrast to the other two heuristics, CHILS is not optimized for vehicle routing instances and does not use the additional clique information provided. Nevertheless, it finds the best solution on 31/37

instances while being significantly closer to the best solutions in the cases where CHILS does not find the best solution. Running CHILS in parallel significantly improves performance to the point where CHILS computes the best solution on 35/37 instances. These results are with a one-hour time limit and a 16-core CPU. Using another 128-core machine for scalability experiments, we show that CHILS reaches speedups up to 104 for the same instances.

Dynamic Algorithms. The basic idea of our dynamic maximum (weight) independent set algorithms is as follows. Let $\mathcal{I}$ be the independent set on the current state of the graph before an update. Then, after this update, we compute a subset $H \subset V$ in our graph G, capturing the neighborhood around the update. These vertices in H are selected such that we can swap all independent set vertices in a solution on the subgraph $G[H]$ and replace the previous independent set vertices in $H \cap \mathcal{I}$. With this, we design the technique *optimal neighborhood exploration* that builds independent induced subgraphs by exploring the neighborhood of a vertex locally up to a certain distance. On these subgraphs, we solve the Maximum (Weight) Independent Set problem with the recent state-of-the-art exact branch-and-reduce algorithm KaMIS BnR [149]. With this technique, we develop dynamic algorithms that can handle insertions and deletions. To make the method feasible in practice, we propose various optimizations, such as limiting the size of the subgraphs, removing high-degree vertices from the subgraphs, or rarely performing expensive updates. In contrast to all theory of dynamic algorithms, our update operation has exponential worst-case time if we are interested in optimal solutions for the subgraphs. Still, our experiments show that the algorithms perform very well in practice. This opens a much wider discussion for dynamic algorithms with non-polynomial update time. Lastly, we provide simple greedy, fully-dynamic algorithms that provide good solutions quickly in practice.

1.1.2 Maximum (Weight) 2-Packing Set

We present new data reduction rules for the Maximum 2-Packing Set and Maximum Weight 2-Packing Set problem. All methods presented for these problems use the corresponding reductions to compute high-quality solutions for large-scale arbitrary graphs. This means, our algorithm is not restricted to inputs with a specific graph property, in contrast to most other related work [94, 78, 206]. We introduce a novel preprocessing technique combining graph reduction with a transformation. This transformation allows us to use algorithms for the Maximum (Weight) Independent Set problem to solve the Maximum (Weight) 2-Packing Set problem.

For the unweighted case, we contribute a new exact algorithm RED2PACK_B&R as well as a heuristic RED2PACK_HEURISTIC. These two approaches differ in how they compute solutions to the MAXIMUM INDEPENDENT SET problem on the transformed graph. Our experiments indicate that our methods outperform the current state-of-the-art approach for arbitrary graphs regarding solution quality and running time. For instance, we can compute optimal solutions for 63 % of our graphs in under a second, whereas the competing method for arbitrary graphs finds optimal solutions only for 5 % of the graphs, even with a ten-hour time limit. Lastly, our method solves many large instances that remained unsolved before.

For the weighted generalization of the problem, we also present new weighted reduction rules. Combined with the same graph transformation, we evaluate these reduction rules using several state-of-the-art independent set solvers. Utilizing our data reductions speeds up the computation of high-quality solutions by multiple orders of magnitude. Moreover, we present a new heuristic that can keep up with the independent set approaches regarding different metrics. In some of the largest instances in our dataset, this heuristic even finds the best solution quality, outperforming all independent set solvers combined with our preprocessing routine.

1.1.3 Hypergraph-b-Matching

Our contribution for the HYPERGRAPH b-MATCHING problem consists of several new exact data reductions and different algorithms to solve the problem on general hypergraphs using these reduction rules.

We present a greedy strategy, a local search heuristic, as well as an ILP formulation for the problem. While our main focus is on the most general WEIGHTED HYPERGRAPH b-MATCHING problem, we also compare our greedy initial solutions on graphs with solvers that are not designed for hypergraphs. Our experiments show that our greedy approaches obtain 10 % better initial solutions than alternative methods. Using the data reduction rules, we achieve a speedup of 6.85 for the WEIGHTED HYPERGRAPH b-MATCHING problem. Moreover, we find quality improvements of up to 30 % by our local search algorithm for the 1-MATCHING problem.

1.2 Outline

In this dissertation, we focus on the MAXIMUM WEIGHT INDEPENDENT SET problem and other independence problems such as the MAXIMUM CARDINALITY 2-PACKING SET, MAXIMUM WEIGHT 2-PACKING SET and the HYPERGRAPH B-MATCHING

problem. For all of these problems, we present new exact data reduction rules and algorithms using these rules to solve the different problems.

The remainder of this dissertation is structured in the following way. After introducing the definitions and notation in Chapter 2, we present the related work for the Maximum Weight Independent Set (MWIS), the Maximum (Weight) 2-Packing Set (M2PS/MW2PS), and the Hypergraph-b-Matching (HBM) problem in Chapter 3. After that, there are three main chapters that each focus on one of the three problems.

The first, Chapter 4, focuses on the MWIS problem. In Section 4.1, we introduce new data reduction rules and give an overview of existing data reduction rules. Then, in Section 4.2, we present a new preprocessing technique that uses Graph Neural Networks to find applicable reductions and thereby speed up the reduction process. In Section 4.3, we introduce an advanced memetic algorithm, and in Section 4.4, a new concurrent iterated local search approach to solve the MWIS problem. Finally, in Section 4.5, we introduce new approaches for the dynamic setting.

The next chapter, Chapter 5, deals with the M2PS and MW2PS problem. We start this chapter with the introduction of our general link-graph data structure and the reduction framework in Section 5.1 and then the reduction to the M(W)IS problem via a graph transformation in Section 5.2. In Section 5.3, we introduce new data reduction rules and algorithms for the cardinality problem, while Section 5.4 presents reduction rules and algorithms for the weighted case.

The third part, Chapter 6, focuses on the Hypergraph b-Matching problem. For this problem, we introduce new data reduction rules and present an exact algorithm and heuristics that use these reductions.

Lastly, Chapter 7 concludes this dissertation by providing a short summary and discussion of future research directions.

Chapter 2

Preliminaries

In this chapter, the fundamentals required for this dissertation are presented. This includes the notation and definitions for different graphs and the problems discussed in the following chapters. Additionally, we give a brief overview of important algorithmic techniques that are used in various algorithms throughout this dissertation. Lastly, we discuss the concept of algorithm engineering, our experimental methodology, and present reoccurring elements used in our experimental evaluations, such as plot types and the specific hardware used.

References. *This chapter introduces unified notation, definitions, and experimental methodology used by the publications that form the basis of this dissertation. Large parts are copied verbatim from these papers or the corresponding technical reports [30, 31, 32, 33, 100, 101, 102, 103, 104, 105].*

2.1 Definitions and Notation

This section presents an overview of the different notation and concepts used in this dissertation. First, preliminary definitions for different graphs are introduced in Section 2.1.1. Afterward, the combinatorial optimization problems considered in this dissertation are defined in Section 2.1.2.

2.1.1 Graph Definitions

In this thesis, a graph $G = (V, E)$ is an simple, undirected graph with $n = |V|$ and $m = |E|$, where $V = \{0, \dots, n-1\}$ and $E \subseteq \binom{V}{2}$. For the set of vertices of a given graph, we define the notation $V(G) = V$, and similarly, for edges, we define $E(G) = E$. The neighborhood $N(v)$ of a vertex $v \in V$ is defined as $N(v) = \{u \in V \mid \{u, v\} \in E\}$. The closed neighborhood of v is defined by $N[v] = N(v) \cup \{v\}$. Analog the

neighborhood $N(U)$ of a set of vertices $U \subseteq V$ is defined by $N(U) = \cup_{v \in U} N(v) \setminus U$ and $N[U] = N(U) \cup U$. The degree of a vertex $\deg(v)$ is defined as the number of its neighbors $\deg(v) = |N(v)|$. We define the maximum degree as $\Delta = \max_{v \in V} \deg(v)$. The complement graph is given by $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{\{u, v\} \mid \{u, v\} \notin E\}$ is the set of edges not present in G. A graph G is *connected* if a path exists between any two vertices. A *node-weighted* undirected graph $G = (V, E, \omega)$ is defined as a set of n vertices V and a set of m edges E with vertex weights $\omega : V \to \mathbb{R}_{>0}$. For a set of vertices $U \subseteq V$, we define the weight of U as $\omega(U) = \sum_{v \in U} \omega(v)$. Given a subset $V' \subseteq V$, the *induced subgraph* $G[V']$ is defined as $G[V'] \coloneqq (V', \{e \in E \mid e \cap V' = e\})$. For a set of edges $R \subseteq E$ of a graph $G = (V, E, \omega)$ we use the notation $G - R$ instead of $(V, E \setminus R, \omega)$ and $G + R$ instead of $(V, E \cup R, \omega)$. We use a similar notation for vertices where for a vertex $v \in V$, $G - v$ denotes the graph $G[V \setminus \{v\}]$ and for a set of vertices $V' \subseteq V$ we use $G - V'$ as a short notation for $G[V \setminus V']$.

Difference Core. We introduce the concept of a Difference-Core (D-Core), which is a subgraph of G defined using a set of solutions $S = \{S_1, S_2, \ldots, S_k\}$ for a combinatorial problem on a graph G. The D-Core is an induced subgraph $G[D]$ with the property that for every vertex $v \in D$, there exist two solutions $S_i, S_j \in S$ such that $v \in S_i$ and $v \notin S_j$. Our new Concurrent Difference Core Heuristic which is introduced in Section 4.4 is based on this concept.

Dynamic Graph. A graph-sequence $\mathbf{G} = (G_0, \ldots, G_t)$ for $t \in \mathbb{N}_0$ is an *edit-sequence of graphs* if for all $0 < i \leq t$ there exists exactly one update to the graph G_{i-1} resulting in the graph G_i. This update can be inserting a new edge $e \notin E(G_{i-1})$ such that $G_i = G_{i-1} + e$, or deleting an existing edge $e \in E(G_{i-1})$ such that $G_i = G_{i-1} - e$. Furthermore, a new vertex $v \notin V(G_{i-1})$ can be inserted yielding $G_i = G_{i-1} + v$, or an existing vertex $v \in V(G_{i-1})$ can be deleted, i. e., $G_i = G_{i-1} - v$, Additionally, an update can also change the weight of an existing vertex.

Link-Graph and Square Graph. A path p in G is defined as a sequence of adjacent edges. The *length* of the path is equal to its number of edges. We extend the graph definition to a *link-graph* $\mathcal{G} = (G, \mathcal{L})$, which is a tuple of a graph G and a set of *links* $\mathcal{L} \subseteq \binom{V}{2} \setminus E$. Two vertices connected by a link are called *linked* vertices. In the link-graph $\mathcal{G}$, a path can also contain links. For each link in the path, we add two to its length. Therefore, the shortest path between two linked vertices in $\mathcal{G}$ is of length two. We define the *induced link-subgraph* of a set of vertices $V' \subseteq V$ as $\mathcal{G}[V'] = (G[V'], \mathcal{L}[V'])$ with $\mathcal{L}[V'] = \{\{u, v\} \in \mathcal{L} \mid u, v \in V'\}$.

We use the short notation $\mathcal{G} - v$ for $\mathcal{G}[V \setminus \{v\}]$ and $\mathcal{G} - V'$ for $\mathcal{G}[V \setminus V']$. Similarly, to the neighborhood, we define the *link-neighborhood* of a vertex $v \in V$ as $L(v) = \{u \in V \mid \{u, v\} \in \mathcal{L} \vee u \in N(N[v])\}$. In the link-graph, the closed 2-neighborhood is defined as $N_2[v] = N[v] \dot\cup L(v)$ and the open 2-neighborhood by $N_2(v) = N_2[v] \setminus \{v\}$. By this definition, for all vertices $u \in L(v)$, the shortest path from u to v is of length two. The *link-degree* of a vertex is defined by the size of its link-neighborhood $\deg_{\mathrm{L}}(v) = |L(v)|$. The *square graph* $G^2 = (V, E^2)$ of a graph $G = (V, E)$ is defined as a graph with the same vertex set, but an extended edge set $E^2 = E \cup \tilde{E}$. For every pair of non-adjacent vertices $u, v \in V$ that share a common neighbor in G, we have $\{u, v\} \in \tilde{E}$. The square graph of a weighted graph $G = (V, E, \omega)$ is equivalently defined as $G^2 = (V, E^2, \omega)$.

We define a *distance-2-clique* as a set of vertices in $\mathcal{G}$ whose vertices are pairwise connected by a path of length at most two. A vertex v is *distance-2-simplicial* if the vertices of $N_2(v)$ form a distance-2-clique.

Hypergraph. A *weighted undirected hypergraph* $H = (V, E, \omega)$ is defined as a set of n vertices V and a multi-set of m hyperedges E with edge weights $\omega : E \to \mathbb{R}_{>0}$. In contrast to the previous definitions, we have edge weights instead of vertex weights. Each edge $e \in E$ is a subset of the vertex set V. For a subset $S \subset E$, we define $\hat{\omega}(S) \coloneqq \{\omega(x) \mid x \in S\}$ as the set of all weights of edges in S. We assume hyperedges to be sets rather than multi-sets; a vertex can only be contained in a hyperedge *once*, while multiple edges can contain the same set of vertices. Therefore, we write e for the set of vertices of a hyperedge e and define $|e|$ as the edge size. The maximum edge size is denoted by $\Delta_E := \max_{e \in E} |e|$. We refer to the edges of a vertex by $E(v) := \{e \in E \mid v \in e\}$ and for a (multi-)set M of edges we define $M(v) := E(v) \cap M$. A vertex v is *incident* to an edge e if $v \in e$. The degree of a vertex v is $|E(v)|$ and $\Delta_V := \max_{v \in V} |E(v)|$ is the maximum degree. Two vertices u, v are *adjacent* if there is an edge incident to u and v. Furthermore, two edges $e,\ f \in E$ are *adjacent* if $e \cap f \neq \emptyset$. Two edges $e,\ f$ are *linked* if there are only vertices of degree two incident to e and f. A set of edges S in H is *independent* if for all distinct edges $f, g \in S$ the vertex sets f and g are disjoint. We define $\mathcal{N}(e) := \bigcup_{v \in e} E(v)$ as the closed neighborhood of an edge. A hypergraph is *d-partite*, if the vertices of the hypergraph can be partitioned into d sets such that no edge is adjacent to two vertices in the same set. In a *d-uniform* hypergraph each edge contains exaclty d vertices. Throughout this dissertation, we use *edges* in the context of hypergraphs and graphs.

2.1.2 Problem Definitions

In the following, we introduce the combinatorial optimization problems considered in this dissertation. We start with the Maximum (Weight) Independent Set, followed by the Maximum (Weight) 2-Packing Set problem, and then we define the Hypergraph b-Matching problem.

Independent Set. For a given undirected graph $G = (V, E)$, a set $\mathcal{I} \subseteq V$ is called *independent set* (IS) if all vertices u, $v \in \mathcal{I}$ are non-adjacent, i. e., $\{u, v\} \notin E$. For a given IS $\mathcal{I}$, a vertex $v \notin \mathcal{I}$ is called *free* if $\mathcal{I} \cup \{v\}$ is still an independent set. An IS is called *maximal* if no free vertices exist. The Maximum Independent Set (MIS) problem is finding an IS with maximum cardinality. The Maximum Weight Independent Set (MWIS) problem is finding an IS with maximum weight. The weight of an independent set $\mathcal{I}$ is defined as $\omega(\mathcal{I}) = \sum_{v \in \mathcal{I}} \omega(v)$. The independence number $\alpha_\omega(G)$ denotes the weight of an MWIS of G. Let $\mathcal{I}$ be an independent set, then we define the *tightness* of a vertex $v \in V \setminus \mathcal{I}$ as the number of independent set vertices in its neighborhood, i. e., $\tau(v) = |N(v) \cap \mathcal{I}|$. Free vertices v have tightness $\tau(v) = 0$.

The complement of an independent set is a vertex cover, i. e., a subset $C \subseteq V$, such that every edge $e \in E$ is covered by at least one vertex $v \in C$. An edge is *covered* if it is incident to one vertex in the set C. The Minimum Vertex Cover problem, defined as looking for a vertex cover with minimum cardinality, is thereby complementary to the maximum independent set problem. That means if $\mathcal{I}$ is an independent set in G, the set of vertices $V \setminus \mathcal{I}$ is a vertex cover. Another closely related concept is cliques. A *clique* is a set $Q \subseteq V$ such that all vertices are pairwise adjacent. A clique in the complement graph $\overline{G}$ corresponds to an independent set in the original graph G. However, for sparse graphs G, solving the Maximum Weight Clique problem on the complement graph $\overline{G}$ is impractical as it is very dense and therefore unlikely to fit in memory for all but the smallest instances.

2-Packing Set. For a given undirected graph $G = (V, E)$ a *2-packing set* is defined as a subset $\mathcal{S} \subseteq V$ of all vertices such that for each pair of distinct vertices $u \neq v \in \mathcal{S}$ the shortest path between u and v has at least length three. In other words, this means the vertices u and v are not adjacent and have no common neighbor in G. The Maximum 2-Packing Set (M2PS) problem is finding a 2-packing set with maximum cardinality. For a weighted graph, the problem is generalized to its weighted version, the Maximum Weight 2-Packing Set (MW2PS) problem. The objective here is to find a 2-packing set $\mathcal{S}$ of maximum weight, i. e., such that $\omega(\mathcal{S}) = \sum_{v \in \mathcal{S}} \omega(v)$ is maximum. A further generalization is the Maximum k-Packing Set problem, where

the shortest path length is bounded by $k+1$ edges. For $k = 1$, this is equivalent to the Maximum Independent Set problem. The Maximum (Weight) 2-Packing Set problem is also referred to as the Maximum (Weight) Distance-d Independent Set problem with $d = k + 1$.

Analogously to the independence number $\alpha(G)$ for the Maximum Independent Set problem, we define $\alpha^2(\mathcal{G})$ as the size of the solution to the Maximum 2-Packing Set problem for a link-graph $\mathcal{G}$. For a graph G, we define $\alpha^2(G) = \alpha^2((G, \emptyset))$. For the weighted generalization, we define $\alpha^2_w(\mathcal{G})$ as the weight of the maximum weight 2-packing set in the link-graph $\mathcal{G}$ and $\alpha^2_w(G) = \alpha^2_w((G, \emptyset))$.

Hypergraph b-Matching. A matching $\mathcal{M} \subseteq E$ in a graph or hypergraph is a set of edges or hyperedges that are pairwise disjoint. In the following, both edges and hyperedges are referred to as edges. The *cardinality* of a matching is defined by the cardinality of the set $\mathcal{M}$. A matching $\mathcal{M}$ is *maximal* if no edge can be added to $\mathcal{M}$. A *maximum cardinality matching* is a matching that contains the largest possible number of edges of all matchings. A *maximum weight matching* $\mathcal{M}$ is a matching that maximizes the weight $\omega(\mathcal{M})$ among all feasible matchings. A matching is called *perfect* if every vertex is incident to an edge contained in the matching. The generalization of matching to b-matching is done by introducing a capacity for each vertex. For a given capacity function $b : V \to \mathbb{N}$, the b-Matching problem relaxes the edge-disjointness constraint so that each vertex can be incident to $b(v)$ edges. We define $b(v)$ as the capacity of vertex v and the maximum vertex capacity as $\beta = \max_{v \in V} b(v)$. For $b \equiv 1$, the b-Matching problem is equivalent to the standard Matching problem. The set of all vertices of an edge $e \in E$ where the capacity is exhausted is defined by the set $\mathit{blocked}(e, \mathcal{M}) := \{v \in e \mid |\mathcal{M}(v)| = b(v)\}$. Edges containing these vertices cannot be added to the matching. Similarly, we define $\mathit{blockedEdges}(e) := \bigcup_{v \in e : b(v)=1} E(v) \setminus \{e\}$ as the set of edges blocked by an edge e. An edge e is called *free*, if $\mathit{blocked}(e, \mathcal{M}) = \emptyset$. Finally, for a finite set $X \subset \mathbb{R}_{>0}$ we define $\mathrm{nmax}(X, k)$ as the k-th largest value in X if it exists, otherwise it is set to 0.

Algorithms and Optimality. In this dissertation, the term *algorithm* is not restricted to solving a problem optimally but also covers heuristic approaches. With the term *solving* a problem, we refer to finding a high-quality feasible solution. If we refer to methods that solve a problem to optimality, we explicitly mention that the algorithm is exact and the problem is solved optimally.

2.2 Key Algorithmic Concepts

This section gives a high-level overview of the main algorithmic techniques used by the different solvers introduced in this work and used in experimental comparisons. The terms kernelization and reduction are defined in Section 2.2.1. The reduction technique is used in many algorithms as a preprocessing step but can also be integrated into the main algorithm. Afterward, the concept of reducing and peeling is introduced in Section 2.2.2, iterated local search is described in Section 2.2.3. Section 2.2.4 presents the core concepts of evolutionary algorithms. Then, Section 2.2.5 discusses the branch-and-reduce paradigm, and Section 2.2.6 introduces integer linear programming.

2.2.1 Reducing with Data Reduction Rules

In this section, we first introduce the notion of reducing, a key concept used in the remainder of this dissertation. For the reduction process, multiple data reduction rules are applied exhaustively. A *data reduction rule* for a problem is a procedure $\mathcal{R}$ that transforms a given instance into an equivalent, generally smaller instance. This transformation typically involves removing or contracting local graph or hypergraph structures in polynomial running time. After applying the reduction rules, the resulting instance is called the *reduced* instance, denoted by $\mathcal{K}$. The original and reduced instances are equivalent in that the optimality of the solution is maintained during the reduction process. That means an optimal solution for the reduced instance can be used to easily construct an optimal solution for the original instance. Otherwise, there are no guarantees on the reduced instance. When an instance can not be reduced further with a given set of reduction rules, it is called *irreducible* for these rules.

With reduction rules, vertices can be identified as three different types. First, a vertex can be identified as part of a solution, also referred to as *including* the vertex. Second, vertices can be categorized as non-solution vertices, also called *excluding* the vertices. Third, vertices can be identified as deferred. In that case, the reduction rules combine multiple vertices into potentially new vertices. This combine procedure is called *folding*. Including or excluding the folded vertices from the final solution only depends on whether the vertices they are folded into are included or excluded in the solution on the reduced instance. To be more precise, folding a set of vertices $X \subseteq V$ into a new vertex v' generally results in a new graph $G' = G[(V \setminus X) \cup \{v'\}]$, where the new vertex v' is connected to all vertices in the neighborhood of X. If the set X is folded into a vertex v already existing in G, then the neighborhood is extended by the neighbors of X, i. e., $N(v) = N(v) \cup N(X)$.

Algorithm 1: ExactReduce(G, R): High level overview.

```
Data: Graph G = (V, E, ω), reduction list R
Result: Reduced instance 𝒦 and offset α
i ← 0
Q ← {{0, 1, ..., |V| − 1} · |R|}          // Initialize FIFO queues for each rule
while i < |R| do
    while not Q[i] empty do
        v = pop(Q[i])
        if v ∈ V(G) then
            𝒦, α' ← R[i](G, v, α)          // Try reduction for vertex v
            if 𝒦 ≠ G then
                G, α ← 𝒦, α'
                i ← 0                      // Continue with first reduction
                for each changed vertex u and all q ∈ Q do
                    if u ∉ q then
                        push(q, u)
    i ← i + 1                              // Continue with next reduction
return 𝒦, α
```

In the ExactReduce routine, given in Algorithm 1, the rules are applied in a predefined order. We employ an exhaustive search to identify applicable reductions, with added pruning for the different rules. This pruning typically limits the vertex degree for which the reductions are tested. If a reduction is successfully applied, testing possible reductions starts from the beginning, following the predefined order. All reductions already tested are now only checked in the areas of the graph that changed. Finally, we obtained the reduced graph $\mathcal{K}$ if no more reductions can be applied.

Reduction rules are effective in practice for computing a maximum independent set [44, 148, 196], minimum vertex cover [4], maximum clique [43, 208], and maximum k-plex [48, 129], as well as solving graph coloring [159, 208] and clique cover [98, 197]. Reduction rules are also used to solve the weighted generalizations of many of these problems [91, 149, 153]. For more details on reduction rules for other problems, we refer the reader to the survey on data reductions by Abu-Khzam et al. [2].

Data reduction rules were originally developed as a theoretical tool for parameterized algorithms [50, 81]. However, as also done in this thesis, some theoretical restrictions are relaxed in work focusing on practical algorithms. These differences are

discussed in the following section but are optional for understanding the remainder of this dissertation.

Excursion: Difference Between Reducing and Kernelization

The term kernelization of an instance, defined in the following, is more restrictive than the reduction process introduced above. Loosely speaking, kernelization additionally provides guarantees on the size of the reduced instance, which is then called a *kernel*. In many practical works, the term kernelization is used more broadly, including the reduction process described above. The following part of this section is devoted to showing the differences between the concepts of kernelization and reducing.

We formally introduce kernelization and fixed-parameter tractable problems closely following Fomin et al. [81]. We use the Vertex Cover problem for examples in this section. The Minimum Vertex Cover problem, where we want to find the smallest vertex cover for a graph, as introduced in Section 2.1.2 is the *optimization version* of this problem. To decide for a given graph G and integer k, whether there is a vertex cover of size at most k in G is an instance of the *decision version* of the Vertex Cover problem. The integer k is called the *parameter*. The solution size as parameter k is also called the natural parameter. In the remainder of this section, we will only consider the decision version of problems.

Formally, a *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ^* is the set of all finite strings over a fixed and finite alphabet Σ. Such a problem L is *fixed parameter tractable* if deciding whether $(x,\, k) \in L$ is possible in running time $\mathcal{O}(f(k) \cdot |x|^c)$, where, f is a computable function only depending on k, $f : \mathbb{N} \to \mathbb{N}$ and c is a constant. This definition means the running time can potentially be exponential in k but is polynomial in the instance size $|x|$. We also denote this running time as $\mathcal{O}^*(f(k))$. Problems with this property form the complexity class **FPT**.

For a parameterized problem L a *kernelization* algorithm is an algorithm that for any given $(x, k) \in \Sigma^* \times \mathbb{N}$ outputs an instance $(x', k') \in \Sigma^* \times \mathbb{N}$ in time polynomial in $|(x, k)|$ with the property

$$((x,\, k) \in L \Leftrightarrow (x',\, k') \in L) \text{ and } |x'|,\, k' \leq h(k),$$

where h is an arbitrary computable function. The result $(x',\, k') \in L$ of a kernelization algorithm K of an instance (x, k) of L is called the *kernel* (under K), the function h is the kernel *size*.

For the Vertex Cover problem, there is a kernelization algorithm K such that for every instance $(G,\, k)$, this algorithm outputs a kernel with $\mathcal{O}(k)$ vertices [80].

Even though the optimization versions of the Minimum Vertex Cover and Maximum Independent Set problem are closely related, the decision version of the Independent Set problem is most likely not fixed-parameter tractable [61].

To provide some intuition of why these problems are different in this theoretical context of kernelization, consider an almost complete graph G. Here, the vertex cover solution contains many vertices since all the edges need to be covered. This means the solution size k is close to the number of vertices in the graph. The independent set solution, however, gets smaller the denser the graph gets since more conflicts are created. Therefore, bounding the size of G by the size of the vertex cover is simpler than bounding the size of G with the size of the independent set.

When solving the optimization version of these problems in practice, all reduction rules applicable for Minimum Vertex Cover can also be used to reduce an instance of Maximum Independent Set. However, we do not get any guarantee on the size of the reduced instance with respect to the solution size. Nevertheless, in practice, reducing the instance size is generally helpful even if we do not have these guarantees. Therefore, we do not want to limit the reduction routines and problems considered to kernelization and fixed-parameter tractable problems. Thus, we use the broader concept of reducing, as introduced at the beginning of this section. This notion includes but is not limited to kernelization.

2.2.2 Reducing-Peeling

Reducing-peeling is a technique used by different state-of-the-art algorithms for solving the Maximum (Weight) Independent Set problem [44, 107]. We give a high-level overview in Algorithm 2. These algorithms mainly consist of two parts that are repeatedly applied. The first is reducing the graph by exact data reduction rules, as introduced in Section 2.2.1. When the graph is irreducible, *peeling* is applied. This peeling process usually involves inexact reductions to decrease the size of the graph further and make other, exact reductions applicable again. These inexact reductions can, for example, be (temporarily) removing the highest degree vertex. When the graph has been fully reduced, the algorithm reconstructs the solution. Afterward, the reconstructed solution is extended to a maximal solution by checking if temporarily inexactly removed vertices are free and can be added.

2.2.3 Local Search

Local search is a heuristic technique for tackling optimization problems by searching between neighboring feasible solutions. This involves iteratively transforming an

Algorithm 2: Reducing-Peeling: High-level overview.

Data: graph $G = (V, E, \omega)$, reduction list R
Result: solution S
$\mathcal{K} \leftarrow G$
while not $\mathcal{K}$ empty **do**
  $\mathcal{K}', \alpha_e \leftarrow$ ExactReduce($\mathcal{K}$, R) // Reducing
  $\tilde{\mathcal{K}}, \alpha_h \leftarrow$ heuristicReduce($\mathcal{K}'$) // Peeling
  $\mathcal{K} \leftarrow \tilde{\mathcal{K}}$
  $\alpha \leftarrow \alpha + \alpha_e + \alpha_h$
$S \leftarrow$restore($\mathcal{K}$, α, $\emptyset$)
$S \leftarrow$makeMaximal(G, S)
return S

initial solution locally by a simple move. For the Maximum Independent Set problem, such a move can be a (j, k)-swap, where j vertices from the solution are removed, and k non-solution vertices are added to the solution. These moves are performed so that the size or weight of the independent set gradually improves. The process terminates when a local optimum is reached. Then, no improving local move to a neighboring solution enhances the solution quality. Generally, there is no guarantee of the quality of the local optimum. Different techniques are used to improve the solution and escape these local optima. For example, simulated annealing [146] or tabu search [93] are ways to improve the plain local search idea. Tabu search uses a set of previous solutions to guide the search by prohibiting or penalizing moves that would result in neighboring solutions of these previous solutions. Of particular interest for this dissertation is the iterated local search metaheuristic [161]. Algorithms based on iterated local search start with an initial solution, which is then optimized using local search. A high-level overview of iterated local search is given in Algorithm 3.

The iterated local search approach generates new solutions by alternating between a series of moves and local search. The moves can be enhanced with various diversification techniques, such as randomization or tabu mechanisms. A new solution is accepted if it meets a specific criterion, such as surpassing the current best solution. Finally, the algorithm terminates based on a predefined stopping criterion.

2.2.4 Memetic and Evolutionary Approaches

In this dissertation, we consider a subgroup of evolutionary algorithms called memetic algorithms. Evolutionary algorithms are optimization strategies that are inspired by

Algorithm 3: Iterated local search: High-level overview, adapted from [198]

```
Data: graph G = (V, E, ω)
Result: solution S
S_0 ← CREATEINITIALSOLUTION(G)
S ← LOCALSEARCH(G, S_0)
while not stopping criterion holds do
    S_pert ← PERTURB(G, S)
    S_imp ← LOCALSEARCH(G, S_pert)
    if ACCEPT(S_imp) then
        S ← S_imp
return S
```

biological evolution. In general, these algorithms mimic the evolution of a population full of individuals using different operations such as mutation, combination, and the survival of the fittest idea. In this context, an *individual* is a potentially infeasible solution to an optimization problem, while the *population* is a diverse set of these individuals. Each individual is assigned a *fitness* value, reflecting the quality of the solution. During the process, the population evolves by applying mutation and combine operations. In the *combine* operations, large parts of two or more individuals, called *parents*, are exchanged or merged to create new offspring. Usually, the parents for the combine operation are chosen based on their fitness value. This results in high-quality solutions having a higher probability of being selected. Since the population size is restricted, individuals are usually evicted over time or when new offspring are generated. The *mutation* operation adds random variations to individuals, for example, by perturbation as used in the iterated local search; see Section 2.2.3. This operation ensures that the population's diversity is not lost.

Memetic algorithms are population-based approaches that have as a central theme the hybridization of different algorithmic approaches [137, 168]. A high-level overview of a memetic algorithm using local search is given in Algorithm 4. The term 'memetic' comes from the concept 'meme' introduced by R. Dawkins [54]. It denotes an analog to the biological gene in cultural evolution. For evolutionary algorithms, it is used to convey the message that, even though these approaches are inspired by biological evolution, they should not be limited to these ideas. These algorithms exploit problem-knowledge by incorporating preprocessing data reduction rules or specialized recombination operators but also local search or other pre-existing heur-

Algorithm 4: High-level overview of a memetic algorithm using local search.

```
Data: graph G = (V, E, ω)
Result: solution I_best
P_0 ← createInitialPopulation(G)
P ← localSearch(P_0)                    // Improve all individuals
while not stopping criterion holds do
    I_1, I_2 ← selectParents(P)         // Based on fitness of individuals
    O_1, O_2 ← combine(I_1, I_2)
    O*_1, O*_2 ← localSearch(O_1, O_2)  // Improve offspring
    Õ_1, Õ_2 ← mutate(O*_1, O*_2)       // Applied with some
     probability
    P ← update(P, Õ_1, Õ_2)             // Insert and evict individuals
return fittest individual I_best from population P
```

istics, approximation, or fixed-parameter tractable algorithms as well as truncated exact methods [168].

2.2.5 Branch and Reduce

The technique *branch-and-bound*, first introduced by Land and Doig in [151], is often used for exactly solving **NP**-hard optimization problems. The *branch-and-reduce* idea extends this technique by adding reductions. Algorithm 5 gives a high-level overview of a general branch-and-reduce approach.

Since it is usually impossible to test all feasible solutions for a problem, the search space of these feasible solutions must be searched systematically. In the *branching* step, the current problem is split into two or more subproblems. This is done recursively for these subproblems, creating a search tree.

A problem is split into subproblems by fixing a variable. In the independent set case, this means we create two branches where a vertex v is once included and once excluded from the solution. This way, there is one variable less to consider in the excluding branch and $|N[v]|$ less in the include branch. Usually, heuristic approaches are used to select the vertices to branch on. For the Maximum Independent Set problem, this could be choosing some random vertex with the highest degree. When a leaf of this branching tree is reached, this corresponds to a feasible solution to the problem.

This plain branching technique results in searching the whole solution space. To avoid this, *bounds* are used to prune branches early. These bounds are the upper and lower bounds on an optimal solution. For a maximization problem, we get a *lower*

Algorithm 5: BRANCH-AND-REDUCE: High-level overview.

Data: graph $G = (V, E, \omega)$, current solution c and best solution s (both initially zero)

Result: optimal solution s

Procedure `BRANCH-AND-REDUCE`(G, c, s)**:**

```
K, α ← ExactReduce(G, R)            // R reduction list
c ← c + α
if K empty then
    return RESTORE(K, α, s)          // Fully reduced, restore solution
if s = 0 then
    s ← lowerBound(K) + c            // Compute feasible solution
if UPPERBOUND(K) + c ≤ s then
    return s                         // Prune branch
(K_1, c_1), (K_2, c_2) ←BRANCH(K, c, s)
s ←BRANCH-AND-REDUCE(K_1, c_1, s)    // Update s
s ←BRANCH-AND-REDUCE(K_2, c_2, s)
return s
```

bound by any feasible solution. A heuristic is often run before starting the branch-and-bound process to get a good lower bound. The lower bound is updated during the procedure if a better solution is found. The *upper bound* restricts how much we can gain from the graph given the current partial solution. For the independent set problem, since at most one vertex per clique can be in an independent set, an upper bound is the size of a clique cover. Note that this bound does not have to be tight. A branch can be *pruned* if the upper bound on that branch is smaller than the lower bound.

An extension to the branch-and-bound idea is *branch-and-reduce*. These algorithms apply the reduction procedure described in Section 2.2.1 in a preprocessing step and after each branching step. These branching decisions change the graph structure of the subproblem and thereby often enable the further application of reduction rules.

2.2.6 Integer Linear Programming

The last technique discussed in this section is integer linear programming. Different optimization problems can be mathematically described by a linear programming (LP) formulation. This is possible if the problem can be defined as a maximization

or minimization of an objective function to a set of constraints. Both the objective function and the constraints have to be linear. If the variables in the objective function have to satisfy integer constraints, we call it an integer linear program (ILP). A standard ILP problem can be formulated as the following:

$$\begin{aligned} \text{maximize} \quad & cx \\ \text{subject to} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

Here, x is the vector of integer variables to be decided, $c \in \mathbb{R}^n$ gives the coefficients in the objective function, $A \in \mathbb{R}^{m \times n}$ is the matrix of coefficients in the constraints, and $b \in \mathbb{R}^m$ gives the right-hand side values.

A special variant of ILP is the binary linear programming problem, where $x \in \{0,1\}^n$. This variant, where only the constraints must be satisfied, was one of Karps 21 NP-complete problems [131].

The MWIS problem can be reduced to a binary linear programming problem by the following formulation. For a given graph $G = (V, E, \omega)$ we define the binary variables x_v for each vertex $v \in V$ as

$$x_v = \begin{cases} 1 & \text{if } v \text{ is in the independent set,} \\ 0 & \text{otherwise.} \end{cases}$$

The ILP formulation for the MWIS problem is then given by:

$$\begin{aligned} \text{maximize} \quad & \sum_{v \in V} \omega(v) x_v \\ \text{subject to} \quad & x_u + x_v \leq 1 \qquad \forall \{u, v\} \in E \\ & x_v \in \{0,1\} \qquad \forall v \in V \end{aligned}$$

The objective function maximizes the weight of the vertices chosen for the independent set, while the constraints ensure that no two adjacent vertices are selected simultaneously. Many combinatorial optimization problems can be reduced to an ILP, as shown above for the MWIS problem. That makes ILP solvers a go-to for comparing and evaluating algorithms for new problems. Therefore, ILP is a very important problem that attracts a lot of research interest, and many, even commercial solvers, are available to solve it.

ILP Solvers. Some of the most popular ILP solvers are CPLEX [126], Gurobi [109], and SCIP [26]. Much research has been done to improve the performance of these solvers. In this dissertation, we use the open-source solver SCIP as a black box solver. We choose to use SCIP since it is one of the fastest open-source ILP solvers available.

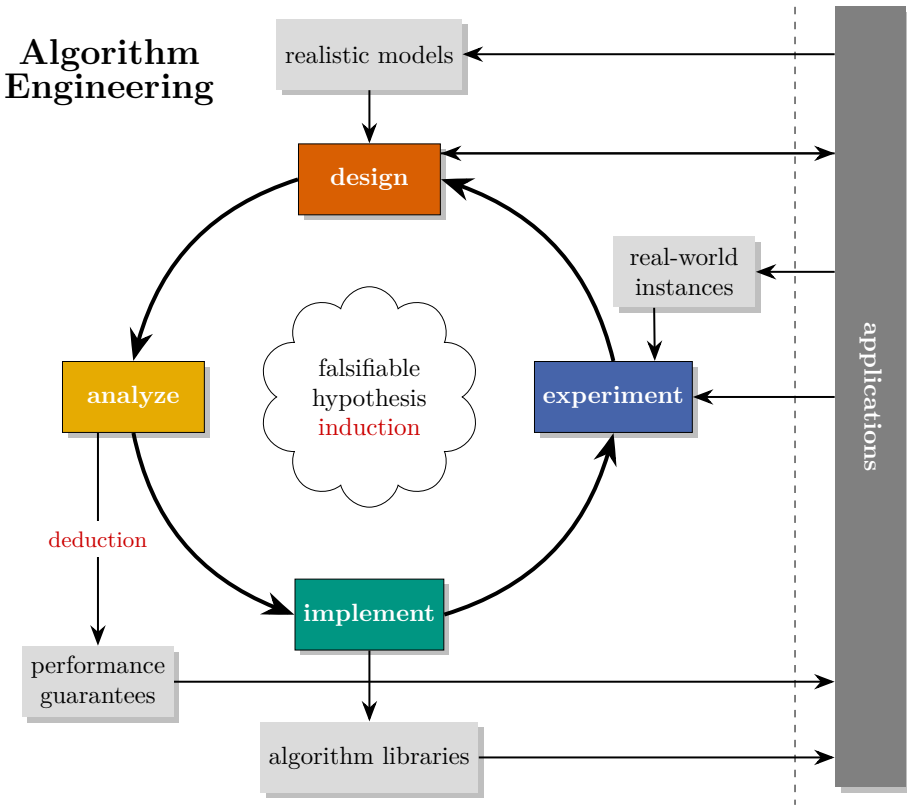


Figure 2.1: Illustration for the algorithm engineering cycle adapted from Peter Sanders [186]. It shows the four main parts of algorithm engineering and its connection to applications and other influences.

2.3 Methodology

This section presents the methodology used in this work. It is split into the methodology of algorithm engineering, which describes the process and environment under which the presented algorithms and data reduction rules were designed. This approach is introduced in Section 2.3.1. Afterward, we introduce the experimental setup in Section 2.3.2, including the description and discussion of performance profiles, which are the main figures used to present the experimental results as well as the machines used to perform experiments.

2.3.1 Algorithm Engineering

The term algorithm engineering describes a cyclic method for algorithm development. It is defined by Peter Sanders in [186]. Following this definition, we give a brief overview of the process. An illustration of the algorithm engineering cycle is shown in Figure 2.1.

The starting point for the algorithm engineering cycle is a falsifiable hypothesis. This can come from a creative idea or inductive reasoning from observations made in previous experiments. The algorithm engineering method consists of four main parts repeated in a cyclic process. These include the design, analysis, implementation, and experimental evaluation of an algorithm to support the hypothesis. Importantly, the whole approach is highly influenced by applications directly or via realistic models and real-world instances for evaluation to obtain practical methods. The analysis performed on the designed algorithm can be used to deduct performance guarantees, which are interesting for practical applications. Furthermore, the implementation part of algorithm engineering can result in the development of algorithm libraries, which can easily transfer the obtained results to applications. Even though highly connected, the applications themselves are not part of the algorithm engineering definition.

In algorithm engineering, as well as in algorithm theory, the goal is to develop efficient algorithms. In addition to good worst-case performances, as in algorithm theory, algorithm engineering focuses on the performance on real-world instances. The described cyclic process influenced by applications is key to achieving this goal.

Many methods presented in this dissertation are metaheuristics, such as randomized local search or evolutionary algorithms. Theoretically, these are usually hard to analyze. However, due to the algorithm engineering approach used to develop and improve these methods, they are very efficient in practice.

2.3.2 Experimental Setup

For our experimental setup, we introduce performance profiles, which are the main plots we present to visualize the experimental results, and we give details about the different machines used for the experiments.

Performance Profiles

To compare different algorithms, we use performance profiles [57]. These plots depict the relationship between the objective function value achieved or computational time spent for each algorithm and the corresponding values produced or consumed by a set of competing algorithms. Performance profiles are mostly unaffected if results change only in a few instances. Furthermore, these plots remain largely unchanged by minor changes across many instances [57]. Each profile of an algorithm yields a non-decreasing, piecewise constant function. Specifically, the y-axis represents the fraction of instances where the objective function is better than or equal to τ times the best objective function value. This is $\#\{objective \geq \tau * best\}/\#G$ for maximization

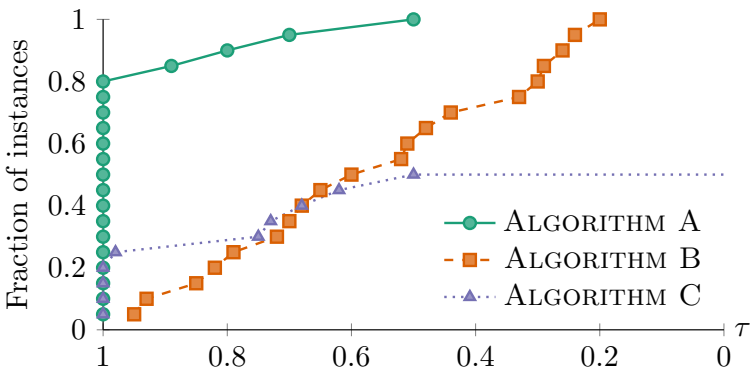


Figure 2.2: Performance profile to compare algorithms for a minimization problem.

and $\#\{objective \leq \tau * best\}/\#G$ for minimization problems. Here, *objective* refers to the result obtained by an algorithm on an instance, and *best* corresponds to the best result among all the algorithms shown in the plot. $\#G$ is the number of graphs in the data set. The parameter τ is plotted on the x-axis. We have $0 < \tau \leq 1$ for maximization problems, while for minimization problems, $\tau \geq 1$. When considering the running time, the y-axis displays the fraction of instances where the time taken by an algorithm is less than or equal to τ times the time taken by the fastest algorithm on that instance, i. e., $\#\{t \leq \tau * fastest\}/\#instances$. Here, t represents the time an algorithm takes on an instance, and *fastest* refers to the time taken by the fastest algorithm on that specific instance. Since the objective is to minimize the time, we have $\tau \geq 1$. Hence, if we want to determine the number of instances in which an algorithm is the best or the fastest compared to the other algorithms in the set, we only need to examine $\tau = 1$.

However, these fractions are only relative to the best solution in the current comparison. Therefore, these profiles can not be used to rank all algorithms [97]. In general, algorithms are considered to perform well if a high fraction of instances are solved within a factor of τ as close to 1 as possible, indicating that many instances are solved close to or better than the optimum solution found by all competing algorithms.

We give an example performance profile in Figure 2.2. It compares three algorithms on 20 instances for a maximization problem. Note that performing best in the following is always in comparison to the other algorithms in the performance profile and does not mean performing strictly better. If algorithms find the same, highest result, they both perform best in this context.

In Figure 2.2, we see that at $\tau = 1$, the values for Algorithm A go up to 0.8, indicating that the algorithm performs best on 80 % of the instances. The Algorithm C achieves a value of 0.2, reflecting that it computes the best score on 20 % of the instances. Algorithm B, on the other hand, is not able to perform best on any of the

instances. The value of τ required to cover all instances for an algorithm is a further interesting indicator for the performance. For Algorithm A, all instances are solved with at most $\tau = 0.5$. Here, we can see that there is one instance in the set where Algorithm A finds a solution that is only half as good as the best solution found for this instance. For Algorithm B, there are seven instances where its solutions are even worse than half compared to the best. For 50 % of the instances, Algorithm C is not able to compute a solution within the experimental setup at all. This is indicated by a continued, straight line that never reaches 100 % of the instances.

Machine and Compilation Details

All algorithms are implemented in C++ (11 or 17) and compiled using g++ version 11.4 or higher with full optimization turned on (`-O3` flag). In the following, we list the machines used for experiments in this thesis and their details.

Machine 1. *A machine equipped with an AMD EPYC 7702P 64-core processor and 1 TB of memory, running Ubuntu 20.04.1 with Linux kernel 5.4.0-187.*

Machine 2. *A machine with an Intel Xeon w5-3435X 16-core processor and 132 GB of memory, running Ubuntu 22.04.4 with Linux kernel 5.15.0-113.*

Machine 3. *A machine with an AMD EPYC 9754 128-Core CPU running at 2.25GHz with 256MB L3 Cache and 768 GB of main memory. It runs Ubuntu 22.04 and Linux kernel 5.15.0-102.*

Machine 4. *These machines have been provided by a cluster and are equipped with two* 20*-core Intel Xeon Gold 6230 processors running at* 2.10 *GHz and having a cache of 27.5 MB. Each machine was equipped with* 96 *or* 192 *GB of main memory.*

Chapter 3

Related Work

This chapter gives an overview of existing work on the different problems discussed in this dissertation. We start by discussing the closely related problems Maximum Weight Independent Set, Minimum Weight Vertex Cover and Maximum Weight Clique and their connections.

In the following sections we include a detailed discussion of related work. It contains related work on the Maximum Independent Set problem in Section 3.1, and for the Maximum Weight Independent Set problem in Section 3.2. Since this is the main focus of this dissertation, we also give a detailed overview of the current state-of-the-art algorithms in Section 3.2.3. Then, we cover work on the related problems Minimum Weight Vertex Cover in Section 3.3 and Maximum Weight Clique in Section 3.4. Finally, we present work on the other independence problems, the Maximum (Weight) 2-Packing Set in Section 3.6, and the Maximum Weight Hypergraph-b-Matching problems in Section 3.7. For more details on data reduction techniques used on other problems, we refer the reader to the recent survey [2].

References. *This chapter introduces related work to the publications that form the basis of this dissertation. It is mainly based in the survey [105], which is joined work with Kenneth Langedal and Christian Schulz. However, large parts are also copied verbatim from the other papers or the corresponding technical reports which this dissertation is based on [30, 31, 32, 33, 100, 101, 102, 103, 104, 105].*

The Maximum Weight Independent Set, the Minimum Weight Vertex Cover, and Maximum Weight Clique problems, are complementary, meaning if we find a maximum weight independent set $\mathcal{I}$ in a graph G, we simultaneously find a minimum weight vertex cover $C = V \setminus \mathcal{I}$ in G and a maximum weight clique in $\overline{G}$. Despite MWIS and MWC being complementary problems, using an MWC algorithm

to solve the MWIS problem on a sparse graph G is impractical since the complement $\overline{G}$ can be very dense and is therefore unlikely to fit in memory for all but the smallest instances. This can also be observed in Figure 3.1 which illustrates the rich history and shows how new solvers are continually compared across these problems. We see that initially, there were some solvers for the MWIS and MWVC also comparing with MWC approaches, however as the instance sizes increased over time, there have been less comparisons between these solvers. The figure additionally highlights the shift towards using data reductions for all three problems. We also include our new solvers M^{2}WIS and CHILS for the MWIS presented in Chapter 4 in this comparison.

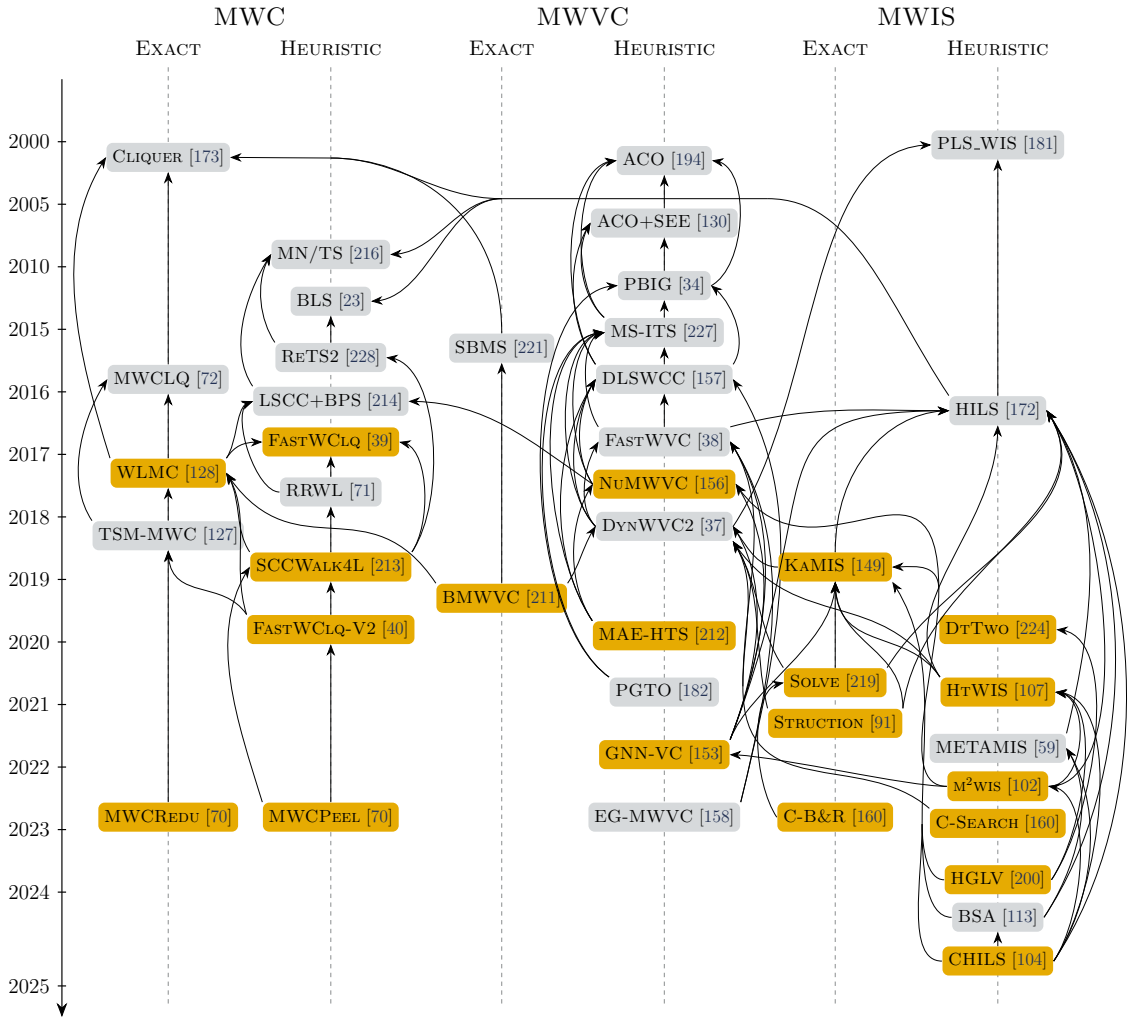


Figure 3.1: This figure illustrates the history of MWC, MWVC, and MWIS solvers. The left axis gives a rough overview of publication years. A directed edge from a solver indicates a comparison made to another solver in the experimental evaluation. For example, the edge from MWCRedu to TSM-MWC indicates that MWCRedu used TSM-MWC in the experimental evaluation. The solvers that are highlighted in yellow are using data reductions.

3.1 Maximum Independent Set

In this section, we cover the most relevant and recent work for the Maximum Independent Set (MIS) problem. First, we discuss exact algorithms and then heuristics.

3.1.1 Exact Methods

For the MIS problem, a large number of branch-and-reduce algorithms have been developed in the past. The currently best exact solver [122], which won the PACE challenge 2019 [122, 199], uses a portfolio of branch-and-reduce/bound solvers. For non-portfolio solvers, Plachetta et al. [178] improved on the branch-and-reduce approach by using SAT solvers for additional pruning. Recently, novel targeted branching strategies have been presented by Hespe et al. [121] and later enhanced by Langedal et al. [152] to improve both branch-and-bound and branch-and-reduce approaches further.

Figiel et al. [76] introduced a new idea added to the state-of-the-art way of applying reductions. They propose not only to perform reductions but also the possibility of undoing them during the reduction process. As they showed in their paper for the Minimum Vertex Cover problem, which is complementary to the MIS problem, this can lead to new possibilities to apply further reductions and finally to smaller reduced graphs.

Finally, there are exact procedures that are either based on other extensions of the branch-and-bound paradigm, e.g. [183], or on the reformulation into other **NP**-complete problems, for which a variety of solvers already exist.

3.1.2 Heuristic Methods

A widely used heuristic approach is called *local search*, which tries to improve a feasible solution by simple insertion, removal, or swap operations. Although, in theory, local search generally offers no guarantees for the quality of the solution, in practice, it routinely finds high-quality solutions significantly faster than exact procedures.

For unweighted graphs, the iterated local search (ARW) by Andrade et al. [7] introduced in 2012 was a very successful heuristic. It is based on so-called $(1, 2)$-swaps, which remove one vertex from the solution and add two new vertices, thus improving the current solution by one. Their algorithm uses special data structures that find such a $(1, 2)$-swap in $\mathcal{O}(m)$ time or prove that none exists. For further details on the ARW solver, see Section 3.2.3.

Building on the ARW framework, Dahlum et al. [52] introduced the OnlineMIS algorithm. They extend the local search approach with simple exact reductions applied

on the fly as well as peeling high-degree vertices. This way, they are able to reduce the search space and improve the performance of the local search algorithm. Section 3.2.3 provides more detailed information about the solver OnlineMIS.

With EvoMIS, Lamm et al. [147] presented an evolutionary approach to tackle the maximum independent set problem. The key feature of their heuristic was to use graph partitioning to come up with natural combine operations, where whole blocks of solutions can be exchanged easily. A local search algorithm was added to these combine operations to improve the solutions further.

Combining the data reductions with the evolutionary algorithm EvoMIS, a reduction evolution algorithm ReduMIS was presented by Lamm et al. [148]. In their experiments, ReduMIS outperformed the local search ARW as well as the pure evolutionary approach EvoMIS.

3.2 Maximum Weight Independent Set

The related work for the Maximum Weight Independent Set (MWIS) problem is covered in detail. First, the exact solvers and then the heuristics introduced to solve this problem are discussed. Finally, we present the most important algorithms for this problem in more detail in Section 3.2.3.

3.2.1 Exact Methods

Exact algorithms compute optimal solutions by systematically exploring the solution space. A frequently used paradigm in exact algorithms for combinatorial optimization problems is called *branch-and-bound* [151]. One of the earliest results using this technique for the problems we consider here was the MWC solver called Cliquer [173]. In the following, we cover the exact solvers developed for the MVC, MWVC, and MWIS in that order.

For solving the MWIS problem, reduction rules have been added to branch-and-bound methods yielding so-called *branch-and-reduce* algorithms [4]. These algorithms extend upon branch-and-bound by applying reduction rules to the current graph before each branching step. KaMIS BnR [149] was the first branch-and-reduce solver introduced for the weighted problems (MWIS, MWVC, and MWC). In Section 3.2.3, it is described in detail. KaMIS BnR has since become a highly influential solver with several new reduction rules. The authors first present two meta-reductions called neighborhood removal and neighborhood folding, from which they derived a new set of weighted reduction rules. On this foundation, a branch-and-reduce algorithm was de-

veloped using pruning with weighted clique covers similar to the approach by Warren and Hicks [215] for upper bounds and an adapted version of the ARW local search [7] for lower bounds. The KaMIS BnR algorithm was then extended to Struction by Gellner et al. [91] to utilize different struction based reduction rules that were originally introduced by Ebenegger et al. [66] and later improved by Alexe et al. [5]. In contrast to previous reduction rules, struction rules do not necessarily decrease the graph size but rather transform the graph, which can lead to further reduction. Two other exact solvers using the branch-and-reduce approach were also recently introduced, called Solve [219] and C-B&R [160]. These solvers use more computationally expensive reduction rules than KaMIS BnR.

In a recent theoretical result, Xiao et al. [218] presented a branch-and-bound algorithm idea using reduction rules working especially well on sparse graphs. They perform a detailed analysis of the running time bound on special graphs in their theoretical work. With the measure-and-conquer technique, they show that the running time of their algorithm is $\mathcal{O}^*(1.1443^{(0.624x-0.872)n})$ where x is the average degree of the graph. This improves previous time bounds for this problem using polynomial space complexity for graphs of average degree up to three.

3.2.2 Heuristic Methods

For the MWIS problem, iterated local search has been frequently used. This meta-heuristic makes random perturbations to the solution to escape local optima. Following the early results of PLS_WIS [181], the hybrid iterated local search heuristic HILS (often called HILS) by Nogueira et al. [172] adapted the ARW algorithm for weighted graphs. In addition to weighted $(1, 2)$-swaps, it also uses $(\omega, 1)$-swaps that add one vertex v into the current solution and exclude its neighbors. Recently, the heuristic METAMIS [59] further improved on HILS by incorporating alternating augmenting-path moves.

The reduce-and-peel approach is also frequently used for the MWIS problem. Here, this method was first used in DtTwo [224] and later improved resulting in HtWIS [107] and HGLV [200].

The most recent heuristic for the MWIS problem is called BSA and is presented by Haller and Savchynskyy [113]. This heuristic differed from the typical local search and reduce-and-peel heuristics presented earlier. Instead, Haller and Savchynskyy introduced a Bregman-Sinkhorn Algorithm (BSA) that addresses a family of clique cover LP relaxations. From the most recent heuristics, only BSA and METAMIS do not use reduction rules. These heuristics were evaluated on a newly published

dataset of vehicle routing instances [60] that are exceptionally hard to reduce. These instances present a new challenge, especially for practical data reductions.

3.2.3 State-of-the-Art Solvers in Detail

This section introduces the current state-of-the-art algorithms for solving the Maximum Cardinality and Weight Independent Set problem in more detail. These algorithms are frequently used to compare within our experiments. Additionally, some of these algorithms are the basis of the work presented in this thesis.

Branch and Reduce Solvers

Algorithm 6: KaMIS BnR by Lamm et al. [149].

Data: graph $G = (V, E, \omega)$, current solution weight c, initially zero, best solution weight $\mathcal{W}$, initially zero

Result: optimal weight $\mathcal{W}$

Procedure branch-and-reduce(G, c, $\mathcal{W}$):

- $\mathcal{K}, \alpha \leftarrow$ ExactReduce(G, R) // R ordered list of reductions
- $c \leftarrow c + \alpha$
- **if** $\mathcal{W} = 0$ **then**
  - $\mathcal{W} \leftarrow \text{ILS}(\mathcal{K}) + c$ // Lower bound
- **if** *UpperBound*$(\mathcal{K})$ $+c \leq \mathcal{W}$ **then**
  - **return** $\mathcal{W}$
- **if** $\mathcal{K}$ is empty **then**
  - **return** $\max\{c, \mathcal{W}\}$
- **if** $\mathcal{K}$ is disconnected **then**
  - **for** $\mathcal{K}_i \in$ Components$(\mathcal{K})$ **do**
    - $c \leftarrow c +$ branch-and-reduce$(\mathcal{K}_i, 0, 0)$
  - **return** $\max(\mathcal{W}, c)$
- $(\mathcal{K}_1, c_1), (\mathcal{K}_2, c_2) \leftarrow$ branch$(\mathcal{K}, c)$
- // Run 1st case, update currently best solution
- $\mathcal{W} \leftarrow$ branch-and-reduce$(\mathcal{K}_1, c_1, \mathcal{W})$
- // Use updated $\mathcal{W}$ to shrink the search space
- $\mathcal{W} \leftarrow$ branch-and-reduce$(\mathcal{K}_2, c_2, \mathcal{W})$
- **return** $\mathcal{W}$

The solver KaMIS BnR employs a branch-and-reduce framework. It maintains both the current solution weight and the best solution weight. The algorithm applies a wide set of reduction rules before branching on a vertex. After reducing, a local search algorithm is run on the reduced graph to compute a lower bound on the solution weight, which helps pruning the search space by excluding unnecessary parts of the search tree to be explored. If the graph is not connected, each connected component is solved separately. If the graph is connected, the algorithm branches into two cases by applying a branching rule. As for the branching rule, initially, vertices are sorted in non-decreasing order by degree, with ties broken by weight. Throughout the algorithm, the next vertex to be chosen is the highest vertex in the ordering. This way, the algorithm quickly eliminates the largest neighborhoods and makes the problem "simpler". If the algorithm does not finish within a certain time limit, the currently best solution is improved using a greedy algorithm. More precisely, the vertices are sorted in decreasing order of their weight and added in that order if feasible. Algorithm 6 gives an overview. Note that the pseudocode describes the algorithm such that it outputs the weight of an MWIS in the graph. However, the algorithm is implemented to output the set of vertices.

Building on the KaMIS BnR framework Gellner et al. [91] introduce a new algorithm called Struction. The authors add increasing transformations and a cyclic blow-up phase to the ExactReduce procedure. This way, instances can be reduced further compared to the KaMIS BnR reduction routine. These increasing transformations are based on reducing the weighted stability number α_ω by potentially increasing the graph. These reductions are formally introduced in Section 4.1. The blow-up phase is a cyclic process of increasing and reducing the graph. For this process, the authors introduce two configurations cyclicStrong and cyclicFast. These configurations differ in how many cycles are performed during this process and the bound for increasing the graph.

Iterated Local Search Based Heuristics

This section introduces iterated local search heuristics to compute high-quality (weight) independent sets. These approaches follow the scheme introduced in Section 2.2.3 and are based on the concept of (j, k)-swaps. These swaps remove j vertices from the solution and add k new vertices. Usually, these numbers k and j are very small.

The first method we discuss is the ARW, introduced in 2012 by Andrade et al. [7] for the Maximum Independent Set problem. It is based on $(1, 2)$-swaps, where one vertex v in the current solution is replaced by two 1-tight vertices $x, y \in N(v)$ in its neighborhood, which are non-adjacent. The authors introduce a special data

structure to represent the solution, which enables finding these swaps in $\mathcal{O}(m)$ time. If there are no swaps existing, the approach can prove this in the same time. This data structure divides the vertices into three blocks, separating solution, free, and non-solution vertices. This heuristic can find (near-)optimal solutions for small to medium-size instances in milliseconds but struggles on large, sparse instances with millions of vertices.

In the heuristic OnlineMIS by Dahlum et al. [52], the authors combined ARW [7] with both exact and inexact data reduction rules. The exact rules reduce the search space, while maintaining the solution quality. Especially for large-scale networks, the authors can boost the performance of their algorithm by running the local search on the reduced instance. In their paper, they show that by applying inexact reductions, they are accelerating the performance and are still able to compete with the best results reported in the literature. The approach OnlineMIS applies a set of simple reductions (for vertices of degree zero, one, and two) [4] on the fly. Only using these reductions enables the algorithm to reduce vertices by marking these and their neighbors as removed during the local search. This is done by first performing a quick single pass when computing the initial solution for ARW. The algorithm further marks the top 1 % of high-degree vertices as removed during this pass, which is an inexact data reduction. During the local search, whenever a vertex is checked for insertion into the solution, the reduction is checked for this vertex. If the reduction is applicable, the solution is updated.

For the Maximum Weight Independent Set problem, the local search heuristic HILS was introduced by Nogueira et al. [172]. Additionally to the $(1, 2)$-swaps, as used in ARW, the authors implement $(\omega, 1)$-swaps. With these moves, a single vertex v is added to the current solution $\mathcal{I}$, and all its neighbors are removed, resulting in the new solution $\mathcal{I}' = \mathcal{I} \setminus N(v) \cup \{v\}$. This swap is improving, if $\omega(v) > \sum_{u \in N(v) \cap \mathcal{I}} \omega(u)$. By keeping track of the differences $\omega(v) - \sum_{u \in N(v) \cap \mathcal{I}} \omega(u)$ for each vertex v, searching for improving $(\omega, 1)$-swaps can be done efficiently.

Reducing-Peeling Approach

The solver HtWIS, proposed by Gu et al. [107], is solving the Maximum Weight Independent Set problem heuristically with a reducing-peeling approach. The general idea of reducing-peeling is introduced in Section 2.2.2. For HtWIS, the authors integrate low-degree and edge-based reductions[1] during the reducing phase of their reducing-peeling heuristic.

[1] The low degree reductions used are 4.1, 4.2, 4.3 and the edge-based reductions used are 4.21 and 4.22.

The author's strategy for heuristically reducing vertices in the peeling phase is a hybrid approach considering the weight of the vertex and its neighborhood weight. For this strategy, the authors introduce a rating for each vertex v, which is the weight difference of v and its neighborhood $\omega(v) - \sum_{u \in N(v)} \omega(u)$. The vertices with the lowest rating are removed. To understand the intuition behind this strategy, consider a vertex v with a positive rating, i. e., $\omega(v) \geq \sum_{u \in N(v)} \omega(u)$. Since any set of vertices that can be added from $N(v)$ has less weight than $\omega(v)$, there is no better option than adding v to the solution. In this case, including v is an exact reduction. Removing the vertex with the smallest rating now shifts the focus to excluding vertices unlikely to be part of the optimal solution. Removing these vertices makes higher-weight vertices in their neighborhoods more likely to be added to the solution.

3.3 Minimum Weight Vertex Cover

The related work for the Minimum Weight Vertex Cover (MWVC) is introduced in this section with a focus on the most recent exact and heuristic solvers using data reduction rules.

3.3.1 Exact Methods

For the MWVC, only one recent exact solver, SBMS [221], did not use data reductions. Instead, SBMS uses a series of SAT formulations that each answered if there is an MWVC of a given size. Since SBMS, every exact algorithm presented for this problem relies on reduction rules. For the MWVC, this is only one other solver BMWVC [211]. The authors analyzed the effectiveness of the reductions and showed that reduction rules often reduce massive graphs to tractable sizes.

3.3.2 Heuristic Methods

For the MWVC problem, the earliest heuristics used ant colony optimization. The first was called ACO [194], which was later improved resulting in ACO+SEE [130]. The next two heuristics used multi-start iterated tabu search [227] (MS-ITS) and a population-based iterated greedy heuristic [34] (PBIG). Since then, a technique based on dynamic edge-weights has been widely adopted for the MWVC problem. This technique was first introduced in DLSWCC [157] and has since been used by several heuristics. Subsequent iterations of this technique brought new improvements, starting with FastWVC [39] that added a construction procedure to generate a high-quality initial vertex cover. Then, NuMWVC [156] added reduction rules as

a preprocessing step to reduce the graph size. Two heuristics called DynWVC and DynWVC2 [37] introduced dynamic strategies for selecting which vertices to add or remove during the search. MAE-HTS [212] combined an evolutionary algorithm with reduction rules on top of the local search. The most recent heuristic to use this edge-weight technique is a hybrid method called GNN-VC [153]. To construct the initial solution, GNN-VC combines data reductions and Graph Neural Networks in a reduce-and-peel approach. Two other recent heuristics deviate from this edge-weight technique. First, a population-based game-theoretic optimizer [182] (PGTO), and second, an evolutionary algorithm based on the snowdrift game [158] (EG-MWVC). Neither of these last two heuristics utilized data reductions.

3.4 Maximum Weight Clique

In this section, we cover the related work on the Maximum Weight Clique (MWC) problem, focusing on work using data reduction rules, starting with exact algorithms followed by heuristics.

3.4.1 Exact Methods

As for the MWIS problem, branch-and-bound algorithms are often used to solve the MWC problem. One of the earliest results using this technique for the MWC is a solver called Cliquer [173]. After the solver Cliquer was presented, several more branch-and-bound solvers for the MWC problem were introduced. These branch-and-bound solvers can broadly be split in two categories. The first category uses MaxSAT reasoning to prune the search space and includes the two branch-and-bound algorithms called MWCLQ [72], and TSM-MWC [127]. The second category focuses on data reductions instead. It includes the WLMC [128] and MWCRedu [70] algorithms. The first algorithm WLMC utilizes a straightforward upper/lower bound reduction rule, where the heaviest known clique is used as a lower bound. Then, for any vertex u, an upper bound on the heaviest clique containing u is $\mathrm{UB}_0(u) = \omega(N[u])$. If this upper bound is less than or equal to the lower bound, u can be removed. In addition to the fast UB_0, they also consider a slightly more complicated upper bound that tries to exclude the heaviest neighbor. With the most recent algorithm, MWCRedu, Erhardt et al. introduced several new reduction rules that significantly improved the state-of-the-art exact solvers. These include reductions based on twins, domination, and simplicial vertices.

3.4.2 Heuristic Methods

A central topic in local search for the MWC problem is how to escape from local optima. For the MWC, several techniques have been added to local search to address this, including tabu search used in MN/TS [216], adaptive perturbation in BLS [23], configuration checking in LSCC+BPS [214], smart restarts used in RRWL [71], and walk perturbation in SCCWalk and SCCWalk4L [213]. The two solvers ReTS1 and ReTS2 [228] also added a new *push* operator that can simultaneously add and remove vertices from a solution, compared to the typical add and swap operators. As with exact methods, using data reductions in heuristics is also becoming more common. For the MWC problem, this was first introduced in the FastWClq [39] and later improved under the same name [40]. We refer to the second version as FastWClq-V2. These heuristics used the upper/lower bound reductions mentioned earlier. The most recent heuristic for the MWC problem, MWCPeel [70], does not use local search but a technique called reduce-and-peel [44] instead. This *reduce-and-peel* is a greedy approach that uses exact reduction rules whenever possible. A heuristic tie-breaking mechanism is needed to ensure progress when exact reductions can no longer reduce the graph. The MWCPeel was introduced alongside MWCRedu and used the same extensive set of reductions.

3.5 Dynamic Maximum Independent Set

In 2021, Hanauer et al. [116] published a survey about fully dynamic graph algorithms. The Maximum Independent Set problem is covered in that survey. We follow their description closely to cover the related work for the dynamic MIS problem: As computing the size of an MIS is **NP**-hard, all dynamic algorithms for independent sets study the Maximal Independent Set problem. In a sequence of papers [11, 10, 21, 45, 108] the expected worst-case running time per update for the Maximal Independent Set problem was reduced to $\mathcal{O}(\log^4 n)$. All these algorithms maintain a maximal independent set. A query can either return the size of that set in constant time or output the whole set in time linear in its size.

While quite a large amount of engineering work has been devoted to the computation of independent sets/vertex covers in static graphs (see above), the amount of engineering work for the dynamic setting is very limited. Zheng et al. [226] presented a heuristic fully dynamic algorithm and proposed a lazy search algorithm to improve the size of the maintained independent set. A year later, Zheng et al. [225] improved the result such that the algorithm is less sensitive to the quality of the initial solution used for the evolving MIS. In their algorithm, called DgOracleTwo, the authors

used two well-known data reduction rules, degree one and degree two vertex reduction, frequently used in the static case. Moreover, DgOracleTwo can handle batch updates. Bhore et al. [27] focused on the special case of MIS for independent rectangles, frequently used in map labeling applications. The authors presented a deterministic algorithm for maintaining an MIS of a dynamic set of uniform rectangles with amortized sub-logarithmic update time. The authors evaluated their approach using extensive experiments. Recently, Gao et al. [90] published the dynamic approximation algorithm DyTwoSwap for the Maximum Independent Set problem, which relies on swapping solution and non-solution vertices. The authors show that their algorithm maintains a $(\frac{\Delta}{2}+1)$-approximate solution over dynamic graphs where Δ is the maximum degree of the graph.

To the best of our knowledge, there exists no related work on the Maximum Weight Independent Set problem in for the dynamic setting.

3.6 Maximum 2-Packing Set

Most of the contributions to the Maximum 2-Packing Set (M2PS) problem on arbitrary graphs are in the context of distributed algorithms for the Maximal 2-Packing Set problem [79, 87, 163, 193, 204]. Ding et al. [56] propose a self-stabilizing algorithm on arbitrary graphs. The algorithm consists of two operations: entering and exiting the solution candidate for each vertex in the graph. If a vertex enters the solution, its neighbors get locked, so they can not enter the solution and cause a conflict. The decision to enter or exit the solution depends on whether a vertex causes a conflict. Mjelde [167] presents a self-stabilizing algorithm for the Maximum k-Packing Set problem on tree graphs using dynamic programming.

There are further algorithms for the M2PS problem considering restricted graph classes. Soto et al. [94] analyze the size of an M2PS for 2-token graphs of paths. Flores-Lamas et al. [78] present an algorithm that finds an M2PS in $O(n^2)$ time for cactus graphs of order n. Trejo-Sánchez et al. [206] present an approximation algorithm for planar graphs using graph decompositions and LP-solvers. The approximation ratio is related to how the algorithm decomposes the input graph into smaller subgraphs, inspired by Baker [17].

Under the name of Maximum (Weight) Distance-3 Independent Set or Maximum Scattered Set problem, there is further theoretical work for special graph classes [14, 133]. For arbitrary graphs, Yamanaka et al. [222] introduced a theoretical exact algorithm to solve the Maximum Distance-3 Independent Set problem in $\mathcal{O}(1.4143^n)$ time. In their work, the authors use some simple reduction

rules. However, these rules only reduce a branch-and-reduce instance consisting of a graph combined with a set of vertices that have to be in the solution and a set that is not.

There are only a few contributions to sequential, practical algorithms for the Maximum 2-Packing Set problem on arbitrary graphs. Trejo-Sánchez et al. [202] are the first authors to have proposed a sequential method for connected arbitrary graphs. They developed a genetic algorithm for the M2PS problem using local improvements in each round of their algorithm and a penalty function.

For the weighted generalization of the problem, Atsuta and Takahashi [12] introduce an approach considering the decision version of the Maximum Weight Distance-d Independent Set problem in interval graphs. In contrast to the optimization problem that we are focusing on, in the decision version, the goal is to decide whether a distance-d independent set of cardinality at least k exists. To the best of our knowledge, there are no practical algorithms for the more general Maximum *Weight* 2-Packing Set problem. However, the possibility of solving the M2PS problem by using a graph transformation to the square graph and applying independent set solvers was first stated by Halldórsson et al. [112]. This approach is also applicable to the weighted case and used in this work.

3.7 Hypergraph b-Matching

There is a vast amount of literature for matchings in graphs [28, 62, 63, 64, 67, 86, 140, 162, 165, 179]. We refer the reader to the respective papers for more details. Here, we cover related work closer to problem variations of the most general weighted hypergraph b-matching problem considered in this dissertation.

Graph b-Matching. The b-Matching problem can be reduced to the simple matching problem according to Gabow [85] by substituting vertices. However, this is impractical on large graphs. An overview of exact approaches can be found in Müller-Hannemann and Schwartz [169]. Grötschel and Holland [106] use the cutting plane technique to tackle the problem. Based on belief propagation and assuming an unique solution exists, Huang and Jebara [125] developed an exact algorithm for the b-matching problem. Mestre [164] proved that the greedy algorithm is a half-approximation and generalized the PGA algorithm by Drake and Hougardy [62] to achieve an $\mathcal{O}(\beta m)$ time half-approximation. The LD algorithm was generalized to the b-matching case by Georgiadis and Papatriantafilou [92] in a distributed fashion. Khan et al. [136] introduced an approximation algorithm bSuitor that can be executed

in parallel. The approach is inspired by the results of Manne and Halappanavar [162] for matchings. Ferdous et al. [74] consider parallel algorithms for the b-matching with submodular objectives.

Hypergraph Matching. According to Hazan et al. [118], the maximum d-set packing problem and, therefore, the matching problem on d-partite, d-uniform hypergraphs can be poorly approximated, and there is no approximation within a factor of $\mathcal{O}(d/\log d)$. In general, as proven by Håstad [117], the matching problem in non-uniform hypergraphs is **NP**-hard and there is no $n^{1-\epsilon}$ factor approximation unless **P** =**NP**. There is a polynomial $(k+1+\epsilon)/3$-approximation algorithm for k-set packing. This corresponds to the matching problem in d-uniform, d-partite hypergraphs proposed by Cygan [49] using local search. Furthermore, Fürer and Yu [84] improved these results concerning the running time. Dufosse et al. [65] introduce several heuristics to reduce the complexity of the uniform problem by extending the well-known two Karp-Sipser [132] rules to hypergraphs. Dufosse et al. [65] present the idea of using the Sinkhorn-Knopp algorithm [195] for the normalization of incident tensors as a third selection rule. They perform practical experiments but are limited to only d-partite, d-uniform hypergraphs with uniform edge weights. Anneg et al. [8] give an improved optimality bound for LP-relaxation for the non-uniform case, which extends to b-matching. For the weighted k-set packing problem Thiery and Ward [201] show improved approximation bounds of 1.786 for $k = 3$. Recently, Neuwohner [171] showed how to proof a threshold below of $\frac{k}{2}$ by $\Omega(k)$. Both approaches improve the long-standing local search approach presented by Berman [25] for maximum weight independent set in d-claw free graphs.

Hypergraph b-Matching. The b-matching cardinality problem in hypergraphs also has no approximation scheme according to El Ouali and Jäger [69], even if the degree of vertices is bounded. Similarly, El Ouali et al. [68] showed that in k-uniform hypergraphs for the cardinality problem with $2 \leq b \leq k/\log k$, there is no polynomial-time approximation within any ratio smaller than $\Omega(\frac{k}{b \log k})$. For weighted b-matching on k-uniform hypergraphs Krysta [144] gave a greedy $k+1$ approximation, while Parekh and Pritchard [175] achieve a $(k-1+\frac{1}{k})$ approximation algorithm via linear programming. Koufogiannakis and Young [142] developed a k-approximation in a distributed fashion for weighted k-uniform hypergraphs. We are not aware of any practical implementation of those algorithms.

Chapter 4

Maximum Weight Independent Set

This chapter presents our contributions to the Maximum Weight Independent Set problem. These include several new data reductions as well as multiple algorithms to solve the problem in the static and dynamic setting. We start with giving an overview of existing data reductions for the Maximum Weight Independent Set (MWIS) problem in Section 4.1. Here, we also present our new reduction rules. Additionally, we present a new GNN-guided screening approach for using expensive reduction rules efficiently in Section 4.2. Following that, we extend the set of existing heuristics for the MWIS problem by introducing a novel memetic approach m^{2}wis in Section 4.3 and the Concurrent Iterated Local Search (CHILS) in Section 4.4. For computing high-quality maximum weight and cardinality independent sets, we introduce the new technique ONE in Section 4.5. This technique can also be used as a local search for static graphs.

References. *This chapter combines the contributions to the MWIS problem. It is based on the publications [104, 105] which are joint work with Kenneth Langedal and Christian Schulz, the paper [32], which is joint work with Jannick Borowitz and Christian Schulz, as well as the papers [102, 103] which are joint work with Sebastian Lamm, Christian Schulz and Darren Strash. Large parts are copied verbatim from the papers or the corresponding technical reports.*

4.1 Exact Data Reduction Rules

This section presents exact data reduction rules for the Maximum Weight Independent Set (MWIS) problem. First, we give a comprehensive overview of existing data reduction rules and present our new data reduction rules. Additionally, we discuss these rules from a practical perspective in Section 4.1.2 and present experimental

results on the best way to order these reduction rules in Section 4.1.3. The rules introduced in this section are used in the algorithms described in the later sections of this chapter.

References. *This section is based on two publications which are joint work with Kenneth Langedal and Christian Schulz [104, 105]. Large parts are copied verbatim from the papers.*

4.1.1 Catalogue of Data Reduction Rules

This section documents the data reduction rules for the MWIS problem. The reductions are grouped into different categories based on common properties. Each section starts with a brief introduction and an intuition for the presented rules. Note that the rules are not ordered by their complexity. The reduction rules are presented using a standardized scheme shown in Reduction 4.0.

Reduction 4.0 ([Reduction Name] by [Authors])

Description of the pattern that can be reduced.

Reduced Graph	*How to build the reduced graph G'*
Offset	*Which weight can be added to the offset*
Reconstruction	*How to construct the solution $\mathcal{I}$ for the original graph given the solution $\mathcal{I}'$ on the reduced graph G'*

First, we give the name of the reduction rule and cite the papers where the rule was first introduced. Then, we define the pattern that this rule can reduce. Finally, we give details on how to perform the actual reduction. This last information consists of three parts. First, the information on constructing the reduced graph G'. Then, the *offset* describes the difference between the weight of an MWIS on the reduced graph $\alpha_\omega(G')$ and the weight of an MWIS on the original graph $\alpha_\omega(G)$. Lastly, the information on how the solution $\mathcal{I}'$ on the reduced instance can be lifted to a solution $\mathcal{I}$ on the original graph is provided.

In addition to including or excluding vertices directly, some reduction rules combine multiple vertices into potentially new vertices. This combine procedure is called *folding*. Including or excluding the folded vertices from the solution $\mathcal{I}$ only depends on whether the vertices they are folded into are included or excluded in the solution $\mathcal{I}'$ on the reduced instance. To be more precise, folding a set of vertices $X \subset V$ into a new vertex v' generally results in a new graph $G' = G[V \setminus X \cup \{v'\}]$, where the new vertex v' is connected to all vertices in the neighborhood of X. If the set X is

folded into a vertex v already existing in G, then the neighborhood is extended by the neighbors of X, i. e., $N(v) = N(v) \cup N(X)$.

Low Degree Reduction Rules

In this section, we cover data reduction rules applicable to vertices of a specific degree. The presented rules fully cover all vertices of degree one and degree two. These are special cases of more powerful reductions presented in later sections.

Reduction 4.1 (Degree One by Gu et al. [107])

Let $u, v \in V$ with $N(v) = \{u\}$.

- *If $\omega(v) \geq \omega(u)$, include v.*

Reduced Graph	$G' = G - N[v]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{v\}$

- *If $\omega(v) < \omega(u)$, fold u and v into new vertex v'.*

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus \{u, v\}]$ *with* $N(v') = N(u)$ *and* $\omega(v') = \omega(u) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $v' \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \setminus \{v'\} \cup \{u\}$, *else* $\mathcal{I} = \mathcal{I}' \cup \{v\}$

Reduction 4.2 (Triangle by Gu et al. [107]. Figure 4.1)

Let $v \in V$ be a degree-two vertex with two adjacent neighbors $x, y \in V$. Without loss of generality, assume $\omega(x) \leq \omega(y)$.

- *If $\omega(v) < \omega(x)$, fold v into x and y.*

Reduced Graph	$G' = G - v$ *and* $\omega(x) = \omega(x) - \omega(v)$, $\omega(y) = \omega(y) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $x, y \notin \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \{v\}$, *else* $\mathcal{I} = \mathcal{I}'$

- *If $\omega(x) \leq \omega(v) < \omega(y)$, exclude x and fold v into y.*

Reduced Graph	$G' = G - \{v, x\}$ *and* $\omega(y) = \omega(y) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $y \notin \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \{v\}$, *else* $\mathcal{I} = \mathcal{I}'$

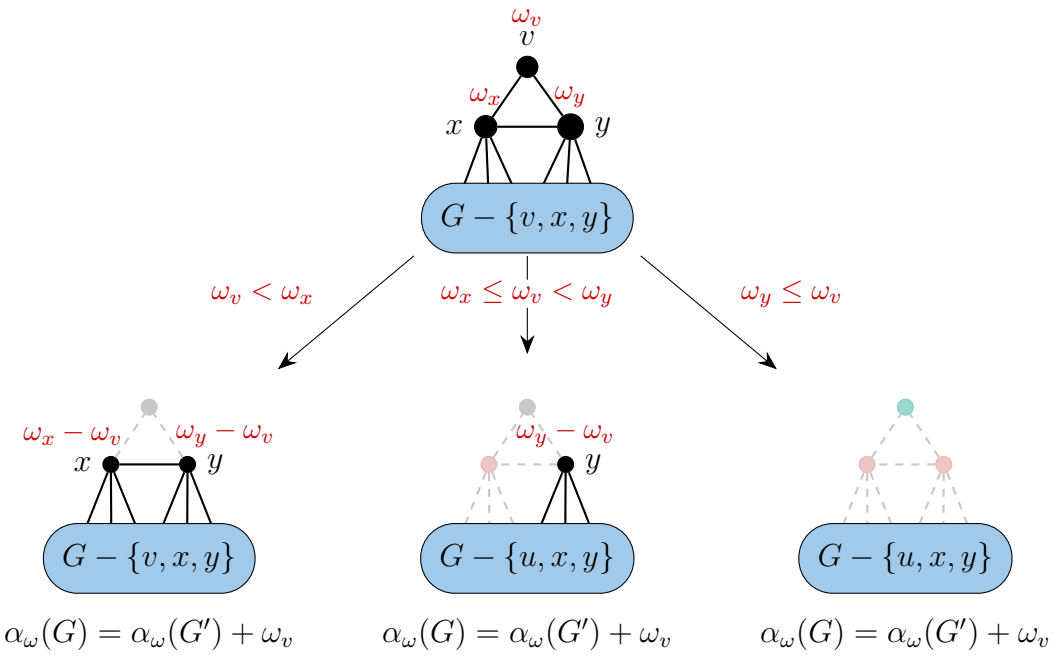


Figure 4.1: Different cases of Reduction 4.2 with $\omega_x \leq \omega_y$. A vertex after reducing is green if included, red if excluded, and gray if folded.

- *If* $\omega(v) \geq \omega(y)$, *include* v.

Reduced Graph	$G' = G - N[v]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{v\}$

Reduction 4.3 (V-Shape by Gu et al. [107] and Lamm et al. [149]. Figure 4.2)

Let $v \in V$ *be a degree-two vertex with two non-adjacent neighbors* $x, y \in V$. *Without loss of generality, assume* $\omega(x) \leq \omega(y)$.

- *If* $\omega(v) < \omega(x)$, *fold* v *into new vertex* v'.

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus \{v\}]$ *with* $N(v') = N(x) \cup N(y)$ *and set* $\omega(x) = \omega(x) - \omega(v)$, $\omega(y) = \omega(y) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $x \in \mathcal{I}'or y \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \setminus \{v\}$, *else* $\mathcal{I} = \mathcal{I}'$

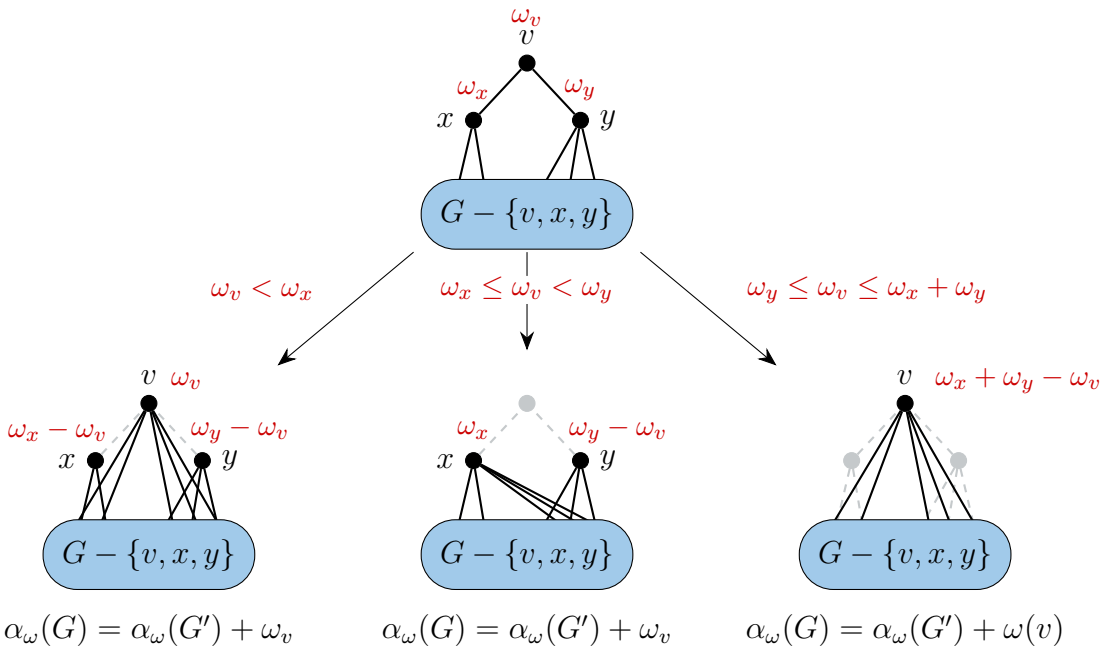


Figure 4.2: Different folding cases of Reduction 4.3 with weights $\omega_x \leq \omega_y$.

- *If* $\omega(x) \leq \omega(v) < \omega(y)$*, fold* v *into* x *and* y*.*

Reduced Graph	$G' = G - v$ *with* $N(x) = N(x) \cup N(y)$ *and* $\omega(y) = \omega(y) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $x, y \notin \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v\}$*, else* $\mathcal{I} = \mathcal{I}'$

- *If* $\omega(y) \leq \omega(v)$ *and* $\omega(x) + \omega(y) \leq \omega(v)$*, include* v*.*

Reduced Graph	$G' = G - N[v]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{v\}$

- *If* $\omega(y) \leq \omega(v)$ *and* $\omega(x) + \omega(y) > \omega(v)$*, fold* v*,* x*,* y *into a new vertex* v'*.*

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus \{v, x, y\}]$ *with* $N(v') = N(x) \cup N(y)$ *and* $\omega(v') = \omega(x) + \omega(y) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $v' \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup \{x, y\} \setminus \{v'\}$*, else* $\mathcal{I} = \mathcal{I}' \cup \{v\} \setminus \{v'\}$

The following reductions deal with special patterns containing degree two vertices, such as paths and cycles.

Reduction 4.4 (3-Path Reduction by Xiao et al. [218])
Let $v_1v_2v_3v_4$ be a 3-path such that $\deg(v_2) = \deg(v_3) = 2$ and $\omega(v_1) \geq \omega(v_2) \geq \omega(v_3) \geq \omega(v_4)$, then fold v_2 and v_3 into the path.

Reduced Graph	$G' = G - \{v_2, v_3\}$, *add the edge* $\{v_1, v_4\}$ *and set* $\omega(v_1) = \omega(v_1) + \omega(v_3) - \omega(v_2)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v_2)$
Reconstruction	*If* $v_1 \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \{v_3\}$, *else* $\mathcal{I} = \mathcal{I}' \cup \{v_2\}$

Reduction 4.5 (4-Path Reduction by Xiao et al. [218])
Let $v_1v_2v_3v_4v_5$ be a 4-path such that $\deg(v_2) = \deg(v_3) = \deg(v_4) = 2$ and $\omega(v_1) \geq \omega(v_2) \geq \omega(v_3) \leq \omega(v_4) \leq \omega(v_5)$, then fold v_2 and v_4 into the path.

Reduced Graph	$G' = G - \{v_2, v_4\}$, *add edges* $\{v_1, v_3\}$ *and* $\{v_3, v_5\}$, *and set* $\omega(v_1) = \omega(v_1) + \omega(v_3) - \omega(v_2)$ *and* $\omega(v_5) = \omega(v_5) + \omega(v_3) - \omega(v_4)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v_2) + \omega(v_4) - \omega(v_3)$
Reconstruction	*If* $v_3 \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \setminus \{v_3\} \cup \{v_2, v_4\}$, *else if* $v_1 \in \mathcal{I}'$ *and* $v_5 \notin \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \{v_4\}$ *else if* $v_1 \notin \mathcal{I}'$ *and* $v_5 \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \{v_2\}$ *else* $\mathcal{I} = \mathcal{I}' \cup \{v_3\}$

Reduction 4.6 (4-Cycle Reduction by Xiao et al. [218])
Let $v_1v_2v_3v_4$ be a 4-cycle such that $\deg(v_2) = \deg(v_3) = 2$ and $\omega(v_1) \geq \omega(v_2) \geq \omega(v_3)$, then fold v_2 and v_3 into the cycle.

Reduced Graph	$G' = G - \{v_2, v_3\}$ *and* $\omega(v_1) = \omega(v_1) + \omega(v_3) - \omega(v_2)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v_2)$
Reconstruction	*If* $v_1 \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \{v_3\}$, *else* $\mathcal{I} = \mathcal{I}' \cup \{v_2\}$

Reduction 4.7 (5-Cycle Reduction by Xiao et al. [218])
Let $v_1v_2v_3v_4v_5$ be a 5-cycle such that $\deg(v_2) = \deg(v_3) = \deg(v_5) = 2$ such that $\min\{\deg(v_1), \deg(v_4)\} \geq 3$ and $\omega(v_1) \geq \omega(v_2) \geq \omega(v_3) \leq \omega(v_4)$.

- *If $\omega(v_3) > \omega(v_5)$, then fold v_5 into the cycle.*

Reduced Graph	$G' = G - v_5$ *and for all* $i \in \{1, 2, 3, 4\}$ *set* $\omega(v_i) = \omega(v_i) - \omega(v_5)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + 2\omega(v_5)$
Reconstruction	*If* $v_1, v_4 \notin \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \{v_5\}$, *else* $\mathcal{I} = \mathcal{I}'$

- *If* $\omega(v_3) \leq \omega(v_5)$*, then fold* v_2 *and* v_3 *into the cycle.*

Reduced Graph	$G' = G - \{v_2, v_3\}$ *and set* $\omega(v_1) = \omega(v_1) - \omega(v_2)$, $\omega(v_4) = \omega(v_4) - \omega(v_3)$ *and* $\omega(v_5) = \omega(v_5) - \omega(v_3)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v_2) + \omega(v_3)$
Reconstruction	*If* $v_1, v_4 \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}'$*,* *else if* $v_1 \in \mathcal{I}'$ *and* $v_4 \notin \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v_3\}$*,* *else if* $v_1 \notin \mathcal{I}'$ *and* $v_4 \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v_2\}$*,* *else* $\mathcal{I} = \mathcal{I}' \cup \{v_2\}$

Reduction 4.8 (6-Cycle Reduction by Xiao et al. [218])

Let $v_1v_2v_3v_4v_5v_6$ *be a 6-cycle such that* $\deg(v_2) = \deg(v_3) = \deg(v_5) = \deg(v_6) = 2$*,* $\omega(v_1) \geq \max\{\omega(v_2), \omega(v_6)\}$*,* $\omega(v_4) \geq \max\{\omega(v_3), \omega(v_5)\}$ *and* $\omega(v_6) \geq \omega(v_5)$*.*

- *If* $\omega(v_2) \geq \omega(v_3)$*, then fold* v_5 *and* v_6 *into the cycle.*

Reduced Graph	$G' = G - \{v_5, v_6\}$*, set* $\omega(v_2) = \omega(v_2) + \omega(v_6)$ *and* $\omega(v_3) = \omega(v_3) + \omega(v_5)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	*If* $v_2 \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v_6\}$*,* *else if* $v_3 \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v_5\}$*,* *else* $\mathcal{I} = \mathcal{I}$*.*

- *Else, fold* v_6 *into the cycle.*

Reduced Graph	$G' = G - v_6$*, add the edge* $\{v_1, v_5\}$ *and set weights* $\omega(v_2) = \omega(v_2) + \omega(v_6)$*,* $\omega(v_3) = \omega(v_3) + \omega(v_5)$ *and* $\omega(v_5) = \omega(v_6) + \omega(v_3) - \max\{\omega(v_2) + \omega(v_6),\ \omega(v_3) + \omega(v_5)\}$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	*If* $v_1,\ v_3 \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v_5\}$*,* *else if* $v_2,\ v_4 \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v_6\}$*,* *else if* $v_1,\ v_4 \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}'$*,* *else* $\mathcal{I} = (\mathcal{I}' \setminus \{v_2,\ v_5\}) \cup \{v_3,\ v_6\}$

The patterns reduced in reductions 4.4 to 4.8 can also be reduced by applying reductions 4.3 and 4.2 to the degree two vertices. Since reductions 4.4 to 4.6 need to search for a more complicated pattern in the graph, it might be more beneficial to only use the more general reductions dealing with degree two vertices in practical application. However, note that using reductions 4.4 to 4.6 can result in different reduced instances.

Neighborhood Based Reduction Rules

This section presents reduction rules that reduce a vertex v based on estimating or computing the weight of the MWIS in the neighborhood $N(v)$. These are special cases and extensions derived from Reduction 4.9.

Reduction 4.9 (Heavy Vertex by Lamm et al. [149])

Let $v \in V$ and $\omega(v) \geq \alpha_\omega(G[N(v)])$, then include v.

Reduced Graph	$G' = G - N[v]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{v\}$

Reductions 4.10 and 4.11 are using an estimate for the MWIS in the neighborhood $\alpha_\omega(G[N(v)])$ to apply the idea more efficiently.

Reduction 4.10 (Neighborhood Removal by Lamm et al. [149])

Let $v \in V$ and $\omega(v) \geq \omega(N(v))$, then include v.

Reduced Graph	$G' = G - N[v]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{v\}$

While Reduction 4.10 uses a simple bound by summing the vertex weights in the neighborhood, this bound can be tightened by using a clique cover in the neighborhood $N(v)$ and summing over the maximum weight vertices per clique. This sum gives an upper bound to the optimal solution and results in Reduction 4.11.

Reduction 4.11 (Clique Neighborhood Removal by Lamm et al. [150, 149])

Let $v \in V$ and C be a set of disjoint cliques in $N(v)$. If $\omega(v) \geq \sum_C \max\{\omega(x) \mid x \in C\}$ include v.

Reduced Graph	$G' = G - N[v]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{v\}$

In the following, reductions 4.12 and 4.13 introduce further reduction possibilities for the case of $\omega(v) < \alpha_\omega(G[N(v)])$.

Reduction 4.12 (Neighborhood Folding by Lamm et al. [149])

Let $v \in V$, and suppose that $N(v)$ is independent. If $\omega(N(v)) > \omega(v)$, but $\omega(N(v)) - \min\{\omega(u) \mid u \in N(v)\} < \omega(v)$, then fold v and $N(v)$ into a new vertex v'.

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus N[v]]$ *with* $N(v') = N(N(v))$ *and* $\omega(v') = \omega(N(v)) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $v' \in \mathcal{I}'$, *then* $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup N(v)$, *else* $\mathcal{I} = \mathcal{I}' \cup \{v\}$

The more general form of reducing a vertex v and its neighborhood is described in Reduction 4.13. For this reduction rule, potentially multiple independent set problems have to be solved in the neighborhood, making the rule very expensive.

Reduction 4.13 (Generalized Neighborhood Folding by Lamm et al. [150, 149])
Let $v \in V$, *then*

- *if* $G[N(v)]$ *contains only one independent set* $\tilde{\mathcal{I}}$ *with* $\omega(\tilde{\mathcal{I}}) > \omega(v)$, *fold* v *and* $N(v)$ *into a new vertex* v'.

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus N[v]]$ *with* $N(v') = N(N(v))$ *and* $\omega(v') = \omega(\tilde{\mathcal{I}}) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $v' \in \mathcal{I}'$, *then* $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup \tilde{\mathcal{I}}$, *else* $\mathcal{I} = \mathcal{I}' \cup \{v\}$

- *if for* $u \in N(v)$ *all independent sets in* $G[N(v)]$ *including* u *have less weight than* $\omega(v)$, *exclude* u.

Reduced Graph	$G' = G - u$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	$\mathcal{I} = \mathcal{I}'$

In [224], Zheng et al. introduce the 2-Vertex Neighbor Removal, an extension of Reduction 4.10 to two non-adjacent vertices.

Reduction 4.14 (Two Vertex Neighborhood Removal by Zheng et al. [224])
Let $u, v \in V$ *be non-adjacent and* $\omega(u) + \omega(v) \geq \omega(N(u) \cup N(v))$. *Further assume for all vertices* $x \in V$ $\omega(x) < \omega(N(x)$ *(i. e., Reduction 4.10 was applied), then include* u *and* v.

Reduced Graph	$G' = G - N[\{u, v\}]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(u) + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{u, v\}$

Xiao et al. [219] have further generalized the idea of Reduction 4.14 in Reduction 4.15 where they tighten the bound for these vertices further. An independent

set $\mathcal{I}$ is called a heavy set, if for any independent set C in the induced subgraph $G[N(\mathcal{I})]$ it holds $\omega(N(C) \cap \mathcal{I}) \geq \omega(C)$. This concept was introduced by [219]. Using this definition, for every independent set $\tilde{\mathcal{I}}$, an equivalent or higher weight independent set $\mathcal{I}^*$ can be constructed by $\mathcal{I}^* := \tilde{\mathcal{I}} \setminus N(\mathcal{I}) \cup \mathcal{I}$. Therefore, these vertices can always be included.

Reduction 4.15 (Heavy Set by Xiao et al. [219])

Let u and v be non-adjacent vertices having at least one common neighbor x. If for every independent set $\tilde{\mathcal{I}}$ in the induced subgraph $G[N(\{u, v\})]$, $\omega(N(\tilde{\mathcal{I}}) \cap \{u, v\}) \geq \omega(\tilde{\mathcal{I}})$, then include u and v.

Reduced Graph	$G' = G - N[\{u, v\}]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(u) + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{u, v\}$

Remark 4.1. *Xiao et al. [219] use the Reduction 4.15 for a heavy sets $\{u, v\}$ only if the neighborhood is small, i. e., $|N(\{u, v\})| \leq 8$.*

We extend the Reduction 4.15 presented by Xiao et al. [219] to the case of a heavy set of three vertices in Reduction 4.16.

Reduction 4.16 (Heavy Set 3)

Let u, v and w be vertices forming a heavy set, then include u, v and w.

Reduced Graph	$G' = G - N[\{u, v, w\}]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(u) + \omega(v) + \omega(w)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{u, v, w\}$

Proof. The proof follows by the definition of a heavy set. □

Clique Based Reduction Rules

The following reductions are based on the observation that in a clique, at most one vertex can be part of a maximum weight independent set. A vertex v where the neighborhood $N(v)$ forms a clique is called *simplicial*. The first rule in this section works with these vertices.

Reduction 4.17 (Simplicial Vertex by Lamm et al. [149])

Let $v \in V$ be simplicial with maximum weight in its neighborhood, i. e., $\omega(v) \geq \max\{\omega(u) \mid u \in N(v)\}$, then include v.

Reduced Graph	$G' = G - N[v]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{v\}$

Reduction 4.18 (Simplicial Weight Transfer by Lamm et al. [149])

Let $v \in V$ be simplicial, and let $S(v) \subseteq N(v)$ be the set of all simplicial vertices. Further, let $\omega(v) \geq \omega(u)$ for all $u \in S(v)$.

- *If $\omega(v) \geq \max\{\omega(u) \mid u \in N(v)\}$, then use Reduction 4.17.*
- *Else, fold v into $N(v)$.*

Reduced Graph	*Construct G' by removing v and all neighbors $u \in N(v)$ with $\omega(u) \leq \omega(v)$. Additionally, set the weight of all remaining neighbors $x \in N(v)$ to $\omega(x) = \omega(x) - \omega(v)$*
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If $\mathcal{I}' \cap N(v) = \emptyset$, then $\mathcal{I} = \mathcal{I}' \cup \{v\}$, else $\mathcal{I} = \mathcal{I}'$*

The idea of Reduction 4.17 is that a simplicial vertex v can be included in the solution if it is of maximum weight in its neighborhood. This rule was first introduced by Lamm et al. [149]. In the following reduction, we extend this idea further. Here, we consider a vertex v which is almost simplicial, meaning there is one vertex $u \in N(v)$ such that if it is removed $N(v) \setminus \{u\}$ forms a clique. We call the pattern of the two vertices $u, v \in V$ such that $u \in N(v)$ and $N(v) \setminus \{u\}$ forms a clique for a *u-v-funnel*.

The main idea of this reduction is that, under certain weight constraints, if u and v form a *u-v-funnel* either u or v is in an MWIS. In this situation, only the three following solution patterns in $N[v]$ can occur. First, the vertex v is in an MWIS. Second, the vertex u and one other neighbor $x \in N(v) \setminus N[u]$ are in the solution. For this case, we need to add additional edges between the vertex u and the remaining vertices $x \in N(v) \setminus N[u]$. Third, only the vertex u is part of the solution. Note that the third case can only occur when $\omega(u) > \omega(v)$. These three cases lead to the weighted version of the funnel reduction, see Reduction 4.19. The unweighted version of this reduction was presented by Xiao et al. [220].

Reduction 4.19 (Weighted Funnel. Figure 4.3)

Assume $u, v \in V$ forms a u-v-funnel and that $\omega(v) \geq \max\{\omega(x) \mid x \in N(v) \setminus \{u\}\}$. Furthermore, let $N'(v) = \{x \in N(v) \setminus N[u] \mid \omega(x) + \omega(u) > \omega(v)\}$.

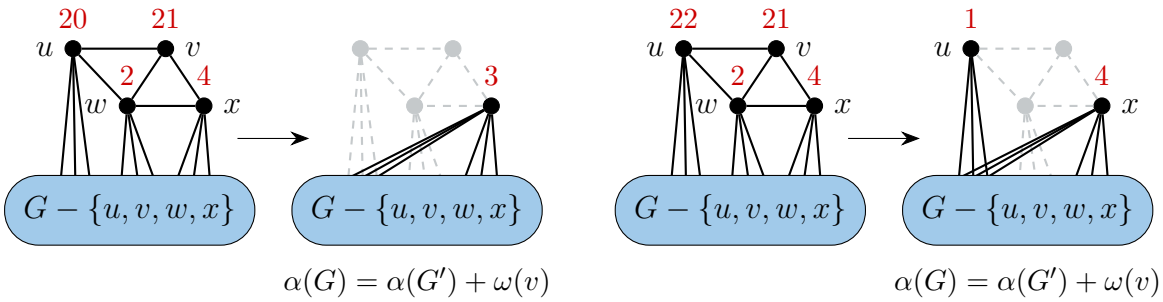


Figure 4.3: Illustration for the two cases of the Weighted Funnel, Reduction 4.19.

- *If* $\omega(v) \geq \omega(u)$*, fold* v *and* u *into its neighborhood.*

Reduced Graph	$G' = G - (N[v] \setminus N'(v))$ *and for all* $x \in N'(v)$*, let* $N(x) = N(x) \cup N(u)$ *and* $\omega(x) = \omega(x) + \omega(u) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $\mathcal{I}' \cap N(v) = \emptyset$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v\}$*, else* $\mathcal{I} = \mathcal{I}' \cup \{u\}$

- *If* $\omega(v) < \omega(u)$*, fold* v *into its neighborhood.*

Reduced Graph	$G' = G - (N[v] \setminus (N'(v) \cup \{u\}))$ *with* $\omega(u) = \omega(u) - \omega(v)$ *and for all* $x \in N'(v)$*, let* $N(x) = N(x) \cup N(u)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $\mathcal{I}' \cap N(v) = \emptyset$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v\}$

Proof. Assume $u, v \in V$ forms a *u-v-funnel* and $\omega(v) \geq \max\{\omega(x) \mid x \in N(v) \setminus \{u\}\}$. Let $N'(v)$ be as defined in the reduction rule. As a first step, we prove that an MWIS contains either u or v. Therefore, assuming u is not in an MWIS, we can exclude u, resulting in v being a simplicial vertex of maximum weight. Now, we can include v (see Reduction 4.17). In the other case, vertex u is in an MWIS. Using this, we can exclude their common neighborhood $N(v) \cap N(u)$. Since $N'(v)$ still forms a clique, only the following three cases for an MWIS $\mathcal{I}$ in G have to be considered. First, v is in $\mathcal{I}$, second only u is part of $\mathcal{I}$ and third, u and *one* of the remaining neighbors $x \in N'(v)$ is in the solution.

We now apply Reduction 4.28 with the independent sets described. Note that when the weight of the independent set is less than $\omega(v)$, no vertex is added in the transformation. In our case, this means that we can remove the remaining neighbors $x \in N'(v)$ if $\omega(x) + \omega(u) < \omega(v)$.

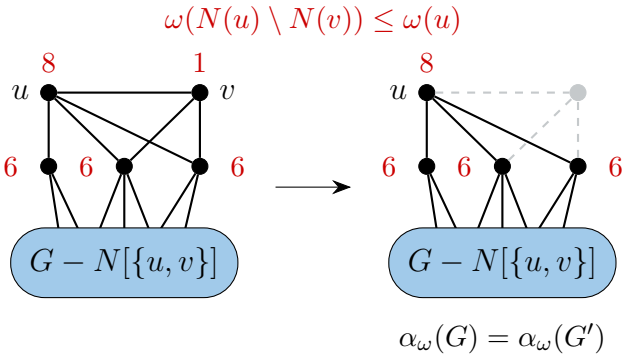


Figure 4.4: Illustration for Basic Single Edge; see Reduction 4.21.

Furthermore, every solution containing a vertex $x \in N'(v)$ has to also contain u by the weight assumption $\omega(v) \geq \omega(x)$. The vertices $x \in N'(v)$ now represent the solutions containing x and u. Therefore, all these remaining neighbors have to be connected to the neighborhood of u. By Reduction 4.28 it holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$.

Next, we consider the different weight relations between u and v. If $\omega(u) < \omega(v)$, there is no solution of higher weight than $\omega(v)$ that only contains u. Therefore, u is not present in the transformed graph.

Otherwise, the vertex u remains in the reduced instance. Note that in this case, edges connect u to all remaining neighbors of v. For each remaining neighbor $x \in N'(v)$, its weight is increased by $\omega(u) - \omega(v)$. Afterward, we apply Reduction 4.23 to all pairs of u and an $x \in N'(v)$, which removes all edges connecting u to $x \in N'(v)$. Reduction 4.23 also reduces the weights of the vertices in $N'(v)$ by the current weight of u, which was reduced by the weight of v. This results in the original weight of the neighbors since $\omega(x) = \omega(x) + \omega(u) - \omega(v) - (\omega(u) - \omega(v))$. These steps give the reduced graph G' as described by Reduction 4.19. □

Domination Based Reduction Rules

The following rules always compare the relation between two adjacent vertices and their neighborhood. In reductions 4.20 and 4.21, a vertex v can be removed since it can always be replaced with its neighbor u in an MWIS.

Reduction 4.20 (Domination by Lamm et al. [149])

Let $u, v \in V$ be adjacent vertices such that $N[u] \subseteq N[v]$. If $\omega(v) \leq \omega(u)$, then exclude v.

Reduced Graph	$G' = G - v$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	$\mathcal{I} = \mathcal{I}'$

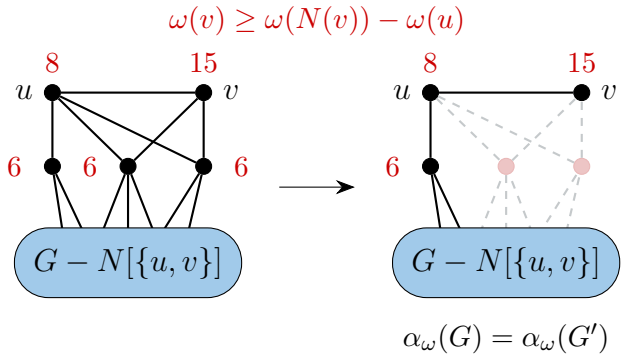


Figure 4.5: Illustration for Extended Single Edge; see Reduction 4.22.

Reduction 4.21 (Basic Single-Edge by Gu et al. [107]. Figure 4.4)

Let u, $v \in V$ be adjacent vertices with $\omega(N(u) \setminus N(v)) \leq \omega(u)$, then exclude v.

Reduced Graph	$G' = G - v$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	$\mathcal{I} = \mathcal{I}'$

In contrast to the previous reductions in this section, Reduction 4.22 covers the case where one of the two vertices u or v have to be in the solution.

Reduction 4.22 (Extended Single-Edge by Gu et al. [107]. Figure 4.5)

Let u, $v \in V$ be adjacent vertices with $\omega(v) \geq \omega(N(v)) - \omega(u)$, then exclude $N(u) \cap N(v)$.

Reduced Graph	$G' = G - (N(u) \cap N(v))$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	$\mathcal{I} = \mathcal{I}'$

We derive the following two reduction rules by extending Reduction 4.20 first introduced by Lamm et al. [149]. The idea of the original domination rule is to find two adjacent vertices $u, v \in V$ where any independent set, including v, can be transformed into an equal or higher weight independent set by replacing v with u.

In the extended domination rule, see Reduction 4.23, instead of removing vertices, we remove edges and reduce vertex weights. With this reduction rule, we allow two adjacent vertices u and v to be both in the solution on the reduced instance. If the edge removal leads to a solution including both vertices, we remove the previously dominated, lower-weight vertex from the solution in the restoring process. With this reduction rule, we sparsify the graph and potentially make other rules applicable.

Reduction 4.23 (Extended Domination)
Let $u, v \in V$ be adjacent vertices such that $N[u] \subseteq N[v]$. If $\omega(v) > \omega(u)$, then we remove the edge between u and v.

Reduced Graph	$G' = (V, E \setminus \{u, v\})$ *and* $\omega(v) = \omega(v) - \omega(u)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	*If* $v \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \setminus \{u\}$, *else* $\mathcal{I} = \mathcal{I}'$

Proof. Let $u, v \in V$ be adjacent vertices such that $N[u] \subseteq N[v]$ and $\omega(v) > \omega(u)$. Further, let G' be the reduced graph after applying the reduction to u and v. We use u' and v' to refer to u and v in G'. First, we consider the case that $v' \in \mathcal{I}'$ and show that $\mathcal{I} = \mathcal{I}' \setminus \{u'\}$ is an MWIS in G. Since $N[u] \subseteq N[v]$ and $v' \in \mathcal{I}'$ it holds that $u' \in \mathcal{I}'$, because $\mathcal{I}'$ is maximal. Now, assume there is an MWIS $\tilde{\mathcal{I}}$ in G with $\omega(\tilde{\mathcal{I}}) > \omega(\mathcal{I})$. Then, the MWIS $\tilde{\mathcal{I}}$ is also an independent set in the reduced instance and $\omega(\tilde{\mathcal{I}}) > \omega(\mathcal{I}' \setminus \{u'\}) + \omega(u) = \omega(\mathcal{I}' \setminus \{u', v'\}) + \omega(v) - \omega(u) + \omega(u) = \omega(\mathcal{I}' \setminus \{u', v'\}) + \omega(v') + \omega(u') = \omega(\mathcal{I}')$. This contradicts that $\mathcal{I}'$ is an MWIS in G'. Furthermore, since $\mathcal{I}'$ is an independent set and we only removed u to obtain $\mathcal{I}$, the independent set property still holds, and therefore $\mathcal{I}$ is an MWIS in G.

We prove the second case $v' \notin \mathcal{I}'$ in the same way. Here, $\mathcal{I}'$ is also an independent set in G. Since $v' \notin \mathcal{I}'$, it holds that the weight of $\mathcal{I}'$ is the same in G and G'. Therefore, there can not exist an independent set of higher weight than $\mathcal{I}'$ in G since this would also exist in the reduced instance, contradicting $\mathcal{I}'$ being an MWIS in G'. □

Additionally, we add a reduction rule which reverses Reduction 4.23. This way, we can add back edges that were previously removed and introduce new edges later in the reduction process.

Reduction 4.24 (Extended Domination Reversed)
Let $u, v \in V$ be non-adjacent vertices such that $N(u) \subseteq N(v)$. If $\omega(u) + \omega(v) < \omega(N(v))$, then we can add an edge between u and v.

Reduced Graph	$G' = (V, E \cup \{u, v\})$ *and* $\omega(v) = \omega(v) + \omega(u)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	*If* $v \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \{u\}$, *else* $\mathcal{I} = \mathcal{I}'$

Proof. Let $u, v \in V$ be non-adjacent vertices such that $N(u) \subseteq N(v)$ and $\omega(u)+\omega(v) < \omega(N(v))$. Further, let G' be the reduced graph after applying the reduction to u and v. We use u' and v' to refer to u and v in G'. Note that since $\omega(v') = \omega(v) + \omega(u)$ the weight $\omega(\mathcal{I}) = \omega(\mathcal{I}')$. First, we consider the case that $v' \in \mathcal{I}'$. We show that

$\mathcal{I} = \mathcal{I}' \cup \{u'\}$ is an MWIS in G. Since $v' \in \mathcal{I}'$ and u' and v' are adjacent in G' it holds that $u' \notin \mathcal{I}'$. Now, assume there is a MWIS $\tilde{\mathcal{I}}$ in G with $\omega(\tilde{\mathcal{I}}) > \omega(\mathcal{I})$. We only consider the case where $u, v \in \tilde{\mathcal{I}}$, since otherwise, either $\tilde{\mathcal{I}}$ is not maximal, or $\mathcal{I}, \mathcal{I}'$ and $\tilde{\mathcal{I}}$ are all equal. With $u, v \in \tilde{\mathcal{I}}$ we construct an independent set $\tilde{\mathcal{I}}' = \tilde{\mathcal{I}} \setminus \{u, v\} \cup \{v'\}$ in G'. It holds that $\omega(\tilde{\mathcal{I}}') = \omega(\tilde{\mathcal{I}})$ since $\omega(v') = \omega(v) + \omega(u)$. Since $\mathcal{I}'$ is an MWIS in G', it follows that $\omega(\mathcal{I}') \geq \omega(\tilde{\mathcal{I}}')$. Now, $\omega(\mathcal{I}) < \omega(\tilde{\mathcal{I}}) = \omega(\tilde{\mathcal{I}}') \leq \omega(\mathcal{I}')$ which contradicts that $\omega(\mathcal{I}) = \omega(\mathcal{I}')$. For the case of $v \notin \mathcal{I}'$, it holds that the weights and sets are directly equivalent between G and G', and therefore $\mathcal{I} = \mathcal{I}'$ is an MWIS in G. □

Remark 4.2. *Initially, adding edges as described in Reduction 4.24 may seem counter-intuitive since the goal is to reduce the graph size. However, this approach is particularly valuable for more complex reduction rules that involve solving independent sets within neighborhoods. By adding edges, additional vertices are incorporated into the direct neighborhood. This expansion can potentially enable further applications of other reduction rules, such as Reduction 4.15 or 4.16.*

Struction Based Rules

The reduction rule struction is based on the stability number reduction and given in Reduction 4.25. It was first introduced by Ebenegger et al. [66] for the unweighted problem. All struction variants presented reduce the stability number $\alpha_\omega(G)$ by the weight of the center vertex v to which the rule is applied. Note that these rules can increase the graph size, which is why we refer to this process as *transforming*. An important concept used in those reduction rules is *layering*. For a given set M of vertices $v_{x,y}$ with two indices $x \in X$ and $y \in Y$ a *layer* $L_k = \{v_{x,y} \in M \mid x = k\}$ contains all elements with the first index equal to k. Note that in Reduction 4.28, the sets X and Y can contain vertices or vertex sets.

Reduction 4.25 (Original Weighted Struction by Gellner et al. [91])

Let $v \in V$ such that $\omega(v) = \min\{\omega(u) \mid u \in N[v]\}$, then transform v.

Reduced Graph *Construct the graph G' as follows*

- *remove v and set $\omega(u) = \omega(u) - \omega(v)$ for each $u \in N(v)$*
- *for all $x, y \in N(v)$, if $\{x, y\} \notin E$ and $x < y$, then add a vertex $v_{x,y}$ with $N(v_{x,y}) = N(\{x, y\}) \setminus \{v\}$ and $\omega(v_{x,y}) = \omega(v)$*

- *for each* $q \in N(v)$ *and all* $v_{q,x}, v_{q,y} \in L_q$*, if* $\{x, y\} \in E$*, then add the edge* $\{v_{q,x}, v_{q,y}\}$
- *for all* $q, r \in N(v)$ *with* $q \neq r$ *and all* $v_{q,x} \in L_q$ *and* $v_{r,y} \in L_r$*, add the edge* $\{v_{q,x}, v_{r,y}\}$

Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $\mathcal{I}' \cap N(v) = \emptyset$ *then* $\mathcal{I} = \mathcal{I}' \cup \{v\}$*, else* $\mathcal{I} = \mathcal{I}' \cap V$

Gellner et al. [91] created further reductions based on Reduction 4.25. The first reduction rule they propose is a modification of Reduction 4.25 such that the number of vertices in the solution $\mathcal{I}'$ is the same as in the original graph. This is achieved by assigning different weights and inserting additional edges, resulting in Reduction 4.26.

Reduction 4.26 (Modified Weighted Struction by Gellner et al. [91])
Let $v \in V$ *such that* $\omega(v) = \min\{\omega(u) \mid u \in N[v]\}$*, then transform* v*.*

Reduced Graph	*Construct the graph* G' *as follows* • *remove* v *and set* $\omega(u) = \omega(u) - \omega(v)$ *for each* $u \in N(v)$ • *for all* $x, y \in N(v)$*, if* $\{x, y\} \notin E$ *and* $x < y$*, then add a vertex* $v_{x,y}$ *with* $N(v_{x,y}) = N(\{x, y\}) \setminus \{v\}$ *and* $\omega(v_{x,y}) = \omega(y)$ • *for each* $q \in N(v)$ *and all* $v_{q,x}, v_{q,y} \in L_q$*, if* $\{x, y\} \in E$*, then add the edge* $\{v_{q,x}, v_{q,y}\}$ • *for all* $q, r \in N(v)$ *with* $q \neq r$ *and all* $v_{q,x} \in L_q$ *and* $v_{r,y} \in L_r$*, add the edge* $\{v_{q,x}, v_{r,y}\}$ • *for each* $k \in N(v)$ *and all* $v_{x,y} \notin L_k$*, add the edge* $\{v_{x,y}, k\}$ • *for all* $x, y \in N(v)$*, add the edge* $\{x, y\}$ *to extend* $N(v)$ *to a clique*
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $\mathcal{I}' \cap N(v) = \emptyset$ *then* $\mathcal{I} = \mathcal{I}' \cup \{v\}$*,* *else* $\mathcal{I} = (\mathcal{I}' \cap V) \cup \{y \mid v_{x,y} \in \mathcal{I}' \setminus V\}$

The Reduction 4.26 is extended by removing the weight restriction for the vertex v resulting in Reduction 4.27.

Reduction 4.27 (Extended Weighted Struction by Gellner et al. [91])
Let $v \in V$ *and* C *the set of all independent sets* c *in* $G[N(v)]$ *with* $\omega(v) < \omega(c)$*, then transform* v*.*

Reduced Graph *Construct the graph G' as follows*
- *remove $N[v]$*
- *for each $c \in C$*
  - *add a vertex v_c with $\omega(v_c) = \omega(c) - \omega(v)$*
  - *for each $w \in N(c) \setminus N(v)$ add the edge $\{w, v_c\}$*
  - *for each $c' \in C \setminus \{c\}$, add the edge $\{v_c, v_{c'}\}$, forming a clique*

Offset $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$

Reconstruction *If $\mathcal{I}' \cap V = \{v_c\}$, then $\mathcal{I} = (\mathcal{I}' \cap V) \cup c$, else $\mathcal{I} = \mathcal{I}' \cup \{v\}$*

To potentially reduce the number of vertices, the authors in [91] also proposed Reduction 4.28. This rule restricts the independent set $c \in C$ used in Reduction 4.27 with an additional weight constraint. With this additional restriction, for a vertex v, only independent sets just greater than the weight $\omega(v)$ are used to create new vertices.

Reduction 4.28 (Extended Reduced Weighted Struction by Gellner et al. [91])

Let C be the set of all independent sets in $G[N(v)]$. We define the set $C' = \{c \in C \mid \nexists c' \in C$ such that $c' \subsetneq c$ and $\omega(c') > \omega(v)\}$ as the set of independent set with weight "just" greater than $\omega(v)$, then transform v.

Reduced Graph *Construct the graph G' as follows*
- *remove $N[v]$*
- *for each independent set $c \in C'$*
  - *add a vertex v_c with weight $\omega(v_c) = \omega(c) - \omega(v)$; call the set of these added vertices $V_{C'}$*
  - *for each $y \in N(v) \setminus N(c)$, add a vertex $v_{c,y}$ with weight $\omega(v_{c,y}) = \omega(y)$; call the set of these added vertices V_E*
  - *for each $w \in N(c) \setminus N(v)$ add the edge $\{w, v_c\}$*
  - *for all $v_{c,y} \in L_c$ and $w \in (N(y) \cup N(c)) \setminus N(v)$ add the edge $\{w, v_{c,y}\}$*
- *for each $v_c \in V_{C'}$ and all $v_{c,x}, v_{c,y} \in L_c$, if $\{x, y\} \in E$, then add the edge $\{v_{c,x}, v_{c,y}\}$*
- *for each $v_{c,y} \in V_E$ and all $v_{c'} \in V_{C'} \setminus \{v_c\}$, add the edge $\{v_{c'}, v_{c,y}\}$*

- *for all* $v_c, v_{c'} \in V_{C'}$*, add the edge* $\{v_c, v_{c'}\}$*, such that* $V_{C'}$ *is forming a clique*
- *for all* $v_{c,y}, v_{c',y'} \in V_E$ *with* $c \neq c'$*, add the edge* $\{v_{c,y}, v_{c',y'}\}$

Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	*If* $\mathcal{I}' \cap V_{C'} = \emptyset$*, then* $\mathcal{I} = \mathcal{I}' \cup \{v\}$*, else replace the one vertex* $v_c \in \mathcal{I}' \cap V_{C'}$ *with the vertices in* c *and all vertices* $v_{c,y} \in \mathcal{I}' \cap V_E$ *with* y *resulting in* $\mathcal{I} = \mathcal{I}' \setminus (V_{C'} \cup V_E) \cup c \cup \{y \mid v_{c,y} \in \mathcal{I}' \cap V_E\}$

Global Reduction Rules

The following data reductions are not local but could potentially extend to the entire graph. We further split these into three categories based on the type of reduction. These are simultaneous sets and unconfined vertices, cut based reductions and the critical set reduction.

Simultaneous Sets and Unconfined Vertices.

Reduction 4.29 (Simultaneous Set by Xiao et al. [219])

A set of vertices $S \subseteq V$ *such that there is an MWIS that either contains all or none of the vertices in* S *is called* simultaneous*. Let* $S \subseteq V$ *be a simultaneous set, then fold* S *into a new vertex* v'*.*

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus S]$ *with* $\omega(v') = \omega(S)$ *and* $N(v') = N(S)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	*If* $v' \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \cup S$*, else* $\mathcal{I} = \mathcal{I}'$

As introduced above, Reduction 4.29 is a meta-reduction not used in practice. However, in the following, we cover rules that are special cases of this reduction. For the next reduction, a vertex is assumed to be part of all MWIS. If this assumption leads to a contradiction, the vertex can be excluded following the described algorithm.

Reduction 4.30 (Unconfined Vertices by Xiao et al. [219])

A vertex v *can be excluded if a contradiction is obtained from the assumption that every maximum weight independent set of* G *includes* v*. Let* S *be an independent set of* G*. A vertex* $x \in N(S)$ *is called a child of* S *if* $\omega(x) \geq \omega(S \cap N(x))$ *and a child is called an extending child if* $|N(x) \setminus N[S]| = 1$*. For an extending child, the only vertex* $y \in N(x) \setminus N[S]$ *is called a satellite of* S*. Starting with* $S = \{v\}$*, we can find*

a contradiction by repeatedly extending S with a satellite until one of the following conditions hold:

1. *There exists a child x such that $N(x) \setminus N[S] = \emptyset$*
2. *All children $x \in S$ have $|N(x) \setminus N[S]| > 1$*

In the second case, the set S confines v, meaning every maximum weight independent set that contains v also contains S, and we can not reduce. In the first case, v is called unconfined and can be excluded.

Reduced Graph	$G' = G - v$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	$\mathcal{I} = \mathcal{I}'$

Next, we extend Reduction 4.30 by Xiao et al. [218]. The intuition behind this rule is that a vertex v can be removed if a contradiction is obtained by assuming every MWIS of G includes v. To define the extended rule, we first introduce the following definitions and lemma.

Let S be an independent set of G. As in the original unconfined reduction, a vertex $x \in N(S)$ is called a *child* of S if $\omega(x) \geq \omega(S \cap N(x))$. For each vertex $y \in N(x) \setminus N[S]$, let $\tilde{\mathcal{I}}_y$ be the MWIS of $G[(N(x) \setminus \{y\}) \setminus N[S]]$. Unlike the original rule, here, a child is called an *extending child* if for some $y \in N(x) \setminus N[S]$ it holds that $\omega(x) \geq \omega(S \cap N(x)) + \omega(\tilde{\mathcal{I}}_y)$. Such a vertex y is called a satellite of S. Intuitively, a satellite is a vertex that must be included in every MWIS under the assumption that S is contained in every MWIS. If it was not, we would have the contradiction we are looking for.

Lemma 4.3. *Let S be an independent set contained in every MWIS of G. Then, every MWIS also contains the satellites from each extending child x of S.*

Proof. Let S be an independent set contained in every MWIS of G and $x \in N(S)$ be an extending child. Assume, towards a contradiction, that a satellite y exists that is not part of every MWIS. By definition, $\omega(x)$ is now greater or equal to $\omega(S \cap N(x)) + \omega(\tilde{\mathcal{I}}_y)$. Thus, for any MWIS that includes S but not y, we can replace $S \cap N(x)$ and $\tilde{\mathcal{I}}_y$ with x to obtain a greater or equally large independent set, which contradicts the assumption that S was contained in every MWIS of G. □

Reduction 4.31 (Extended Unconfined Vertices)

A vertex $v \in V$ can be removed if it is unconfined, proven by the following procedure. Start with a set $S = \{v\}$. We assume S is contained in every MWIS in G. We can

search for a contradiction by repeatedly extending S with satellites from one extending child until one of the following conditions hold

1. *There exists a child x such that $\omega(x) \geq \omega(S \cap N(x)) + \alpha_\omega(G[N(x) \setminus N[S]])$.*
2. *There exist no further satellites to extend S.*

In the second case, the set S confines v, meaning every maximum weight independent set that contains v also contains S, and we can not remove v. In the first case, v is called unconfined and can be excluded.

Reduced Graph	$G' = G - v$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	$\mathcal{I} = \mathcal{I}'$

Proof. Assume that S, initially just $\{v\}$, is contained in every MWIS and that Condition 1 holds. By Lemma 4.3, after every extension of S with satellites, S is still contained in every MWIS in G. But when Condition 1 holds, any MWIS $\mathcal{I}$ of G with $S \subseteq \mathcal{I}$ can be modified using the child x to obtain a new independent set $\mathcal{I}' = \{x\} \cup (\mathcal{I} \setminus N(x))$. From Condition 1, it follows that $\omega(\mathcal{I}') \geq \omega(\mathcal{I})$, breaking the assumption that S is contained in every MWIS of G. Therefore, v is removable. □

Remark 4.4. *This extended version of unconfined was already suggested in a remark by Xiao et al. [219], but they did not introduce it formally as is done here. It is also important to note that this version of unconfined is not practical in its most general form. Several MWIS problems need to be solved for each extending child, making it too computationally intensive for practical implementations. We implement this rule by restricting it to only search for satellites in neighborhoods that form an independent set. This way, we can detect multiple satellites from an extending child without solving any additional MWIS problems.*

After having computed the confining sets in Reduction 4.30 or Reduction 4.31, we can reduce the instance further by working with these sets. In [219] Xiao et al. introduced the notion of a *simultaneous* set, which is a set of vertices $\{u, v\} \subseteq V$, where u is in the confining set of v and v is in the confining set of u, i. e., $u \in S_v$ and $v \in S_u$. In this case, both vertices are either in all MWIS or can both be excluded. Therefore, these vertices can be folded, as described in Reduction 4.32.

Reduction 4.32 (Simultaneous Confined by Xiao et al. [219])

Let $u, v \in V$ and S_u, S_v be their corresponding confining sets computed by Reduction 4.30. If $u \in S_v$ and $v \in S_u$, then fold u and v into a new vertex v'.

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus \{u, v\}]$ *with* $\omega(v') = \omega(u) + \omega(v)$ *and* $N(v') = N(\{u, v\})$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	*If* $v' \in \mathcal{I}'$, *then* $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup \{u, v\}$, *else* $\mathcal{I} = \mathcal{I}'$

The next rule works similarly to Reduction 4.30. However, here, we assume a vertex to be part of *no* MWIS. If a contradiction is found, we include the vertex.

Reduction 4.33 (Uncovered Vertices by Liu et al. [160])

A vertex v can be included if a contradiction is obtained, assuming that no maximum weight independent set of G includes v. Let $C \subset V$ be a set of vertices that are in no MWIS of G. For a vertex $x \in C$ we define a mirror *as a vertex $y \in N^2(x)$ satisfying $\omega(x) \geq \alpha_\omega(G[N(x) \setminus (C \cup N(y))])$. Starting with $C = \{v\}$, we can find a contradiction by repeatedly extending C with mirrors until one of the following conditions hold:*

1. *There exists a vertex $y \in C$ such that $\omega(y) \geq \alpha_\omega(G[N(y) \setminus C])$*
2. *There are no mirrors to extend the set C*

In the second case, the set C covers v, meaning every maximum weight independent set that does not contain v also does not contain C, and we can not reduce. In the first case, v is called uncovered and can be included.

Reduced Graph	$G' = G - N[v]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{v\}$

Reduction 4.34 (Simultaneous Cover by Liu et al. [160])

Let $u, v \in V$ and S_u, S_v be their corresponding covering sets computed by Reduction 4.33. If $u \in S_v$ and $v \in S_u$, then fold u and v into a new vertex v'.

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus \{u, v\}]$ *with* $\omega(v') = \omega(u) + \omega(v)$ *and* $N(v') = N(\{u, v\})$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	*If* $v' \in \mathcal{I}'$, *then* $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup \{u, v\}$, *else* $\mathcal{I} = \mathcal{I}'$

Cut Based Reduction Rules. The next two reduction rules are based on small vertex-cuts of a graph. If there is such a cut, one component of the graph can be solved for all solution combinations in the cut and folded accordingly into new vertices.

Reduction 4.35 (One Vertex Cut by Xiao et al. [218])

Let v be an articulation point in G and G^* a connected component in $G - v$. Let $\mathcal{I}_1$ be the optimal solution on G^* and $\mathcal{I}_2$ the optimal solution on $G^* - N(v)$.

- *If $\omega(v) + \omega(\mathcal{I}_2) \leq \omega(\mathcal{I}_1)$, then exclude v and include the vertices in $\mathcal{I}_1$.*

Reduced Graph	$G' = G - (V(G^*) \cup \{v\})$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(\mathcal{I}_1)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \mathcal{I}_1$

- *Else, fold v and G^* to a new vertex v'.*

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus V(G^*)]$ *with* $\omega(v') = \omega(v) + \omega(\mathcal{I}_2) - \omega(\mathcal{I}_1)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(\mathcal{I}_1)$
Reconstruction	*If* $v' \in \mathcal{I}'$, *then* $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup \{v\} \cup \mathcal{I}_2$, *else* $\mathcal{I} = \mathcal{I}' \cup \mathcal{I}_1$

Reduction 4.36 (Two Vertex Cut by Xiao et al. [218])

Let u, v be a vertex cut, i. e., after removing u and v, the graph G is disconnected into two components. Let G^* be a connected component in $G - \{u, v\}$. Let the following sets be MWIS for the respective subgraphs, $\mathcal{I}_v$ for $G^* - N[v]$, $\mathcal{I}_u$ for $G^* - N[u]$, $\mathcal{I}_{u,v}$ for $G^* - N[\{u, v\}]$ and $\mathcal{I}^*$ for G^*. We assume w.l.o.g. that $\omega(\mathcal{I}_v) \geq \omega(\mathcal{I}_u)$, then fold G^* into new vertices $x_u, x_v, x_{u,v}$.

Reduced Graph	*Construct G' by* • *removing G^** • *adding x_u with $\omega(x_u) = \omega(\mathcal{I}_u) - \omega(\mathcal{I}_{u,v})$* • *adding x_v with $\omega(x_v) = \omega(\mathcal{I}_v) - \omega(\mathcal{I}_{u,v})$* • *adding $x_{u,v}$ with $\omega(x_{u,v}) = \omega(\mathcal{I}^*) - \omega(\mathcal{I}_u)$* • *adding edges $\{v, x_u\}$, $\{x_u, x_v\}$, $\{x_v, u\}$, $\{v, x_{u,v}\}$ and $\{u, x_{u,v}\}$*
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(\mathcal{I}_{u,v})$
Reconstruction	*If* $u \notin \mathcal{I}'$ *and* $v \in \mathcal{I}'$, *then* $\mathcal{I} = (\mathcal{I}' \setminus \{x_v\}) \cup \mathcal{I}_v$, *else if* $u \in \mathcal{I}'$ *and* $v \notin \mathcal{I}'$, *then* $\mathcal{I} = (\mathcal{I}' \setminus \{x_u\}) \cup \mathcal{I}_u$, *else if* $u, v \in \mathcal{I}'$, *then* $\mathcal{I} = \mathcal{I}' \cup \mathcal{I}_{u,v}$, *else* $\mathcal{I} = (\mathcal{I}' \cap V) \cup \mathcal{I}^*$

Remark 4.5. *For the Reductions 4.35 and 4.36, the authors additionally impose a bound for the component G^*.*

Critical Weight Independent Set. The critical weight independent set is a costly but powerful reduction rule. It covers the related crown reductions for the vertex cover problem [46, 73]. It is also closely related to the known fact that an optimal solution to the LP-relaxation is always half-integral [170], i. e., the optimal solution will always have each vertex as 0, 1, or $\frac{1}{2}$. The vertices assigned 0 or 1 in such an optimal LP solution can be included or excluded, respectively. The following critical set reduction describes this scenario using the notion of a critical weight IS.

Reduction 4.37 (CWIS by Butenko and Turkhanov [36])

Let $\mathcal{I}_c \subseteq V$ be a critical weighted IS of G, i. e., $\omega(\mathcal{I}_c) - \omega(N(\mathcal{I}_c)) = \max\{\omega(\mathcal{I}) - \omega(N(\mathcal{I})) \mid \mathcal{I}$ is an IS of $G\}$, then include vertices in $\mathcal{I}_c$.

Reduced Graph	$G' = G - N[\mathcal{I}_c]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(\mathcal{I}_c)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \mathcal{I}_c$

Despite the short definition, it is probably the most complicated rule to implement out of all the rules presented in this survey. Ageev [3] introduced a polynomial time algorithm for how to find a critical weight IS. In the following, we give an outline of that algorithm. To start, consider the following ILP formulation. For this, we use two binary vectors X and Y, where X represents the vertices in $\mathcal{I}_c$ and Y the vertices $N(\mathcal{I}_c)$. Because X and Y are binary vectors representing these sets, in saying add u to X, we mean to set the u-th index in X to 1.

$$\max \sum_{u \in V} \omega(u) X_u - \omega(u) Y_u$$

$$\text{s.t.} \quad Y_u \geq X_v \qquad \forall \{u, v\} \in E$$

$$X_u, Y_u \in \{0, 1\} \qquad \forall\, u \in V$$

In this formulation, for each vertex u added to X, the neighborhood $N(u)$ must be added to Y. The objective value for an optimal solution to this ILP is non-negative since adding all vertices to X and Y is a feasible solution of weight zero. Note that with this definition, an optimal solution is not guaranteed to be an independent set. However, as Ageev points out, we can always find an independent set with the exact same objective value by selecting all isolated vertices in the induced graph obtained from the elements in X. This is true because any vertices in X with other neighbors in X must also be in Y. Therefore, these vertices contribute exactly zero to the objective value. We introduce this ILP formulation because it is an instance of a simpler problem called the Selection problem that can be solved in polynomial time [18].

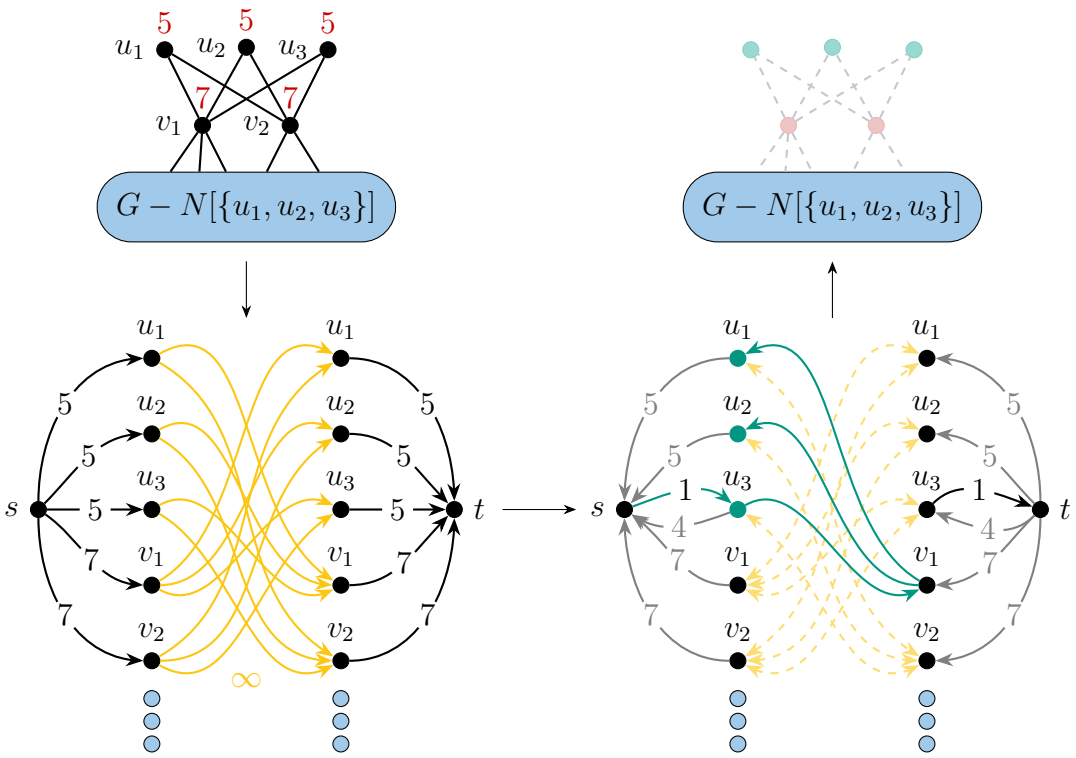


Figure 4.6: This illustration shows how a CWIS can be found efficiently by identifying a maximum flow in a bipartite graph constructed from the original graph. Starting from the original graph in the top left, we construct the bipartite graph shown on the bottom left. All the edges between the two main layers, shown in yellow, have infinite capacity. After identifying a maximum flow in this bipartite graph, the residual graph is shown on the bottom right. A CWIS can now be found by running a BFS or DFS in the residual graph starting from s. All the vertices we can reach in the first layer make up a CWIS. In the example, the vertices u_1, u_2, and u_3 can be reached using the path highlighted in green. Note that we must construct the bipartite graph for the entire input graph to ensure correctness.

Balinski introduced an algorithm for the Selection problem that we can use to identify a CWIS directly [18]. The algorithm solves a Maximum Flow problem on a special flow graph constructed from the original graph. An illustration of this algorithm is shown in Figure 4.6. The flow graph construction starts with a directed bipartite graph with two sets of the original vertices V and V'. For each edge $\{u, v\} \in E$ in the original graph, we add a directed edge from $u \in V$ to $v \in V'$ with infinite capacity in the bipartite graph. Then, add two more vertices s and t. For each vertex $u \in V$, add the directed edge (s, u) with capacity $\omega(u)$, and for each vertex $u \in V'$, add the directed edge (u, t) with capacity $\omega(u)$. In this flow graph, we want to find a minimum cut. By removing the edges in this minimum cut, any vertices in V that

can still be reached from s make up our critical weight IS. As shown in Figure 4.6, we can do this directly by running a BFS or DFS from s in the residual graph after solving the Maximum Flow problem.

Twin Based Reduction Rules

Two vertices u and v which are not connected but have the same neighborhood are called twins. These present a special case of Reduction 4.29 and can be reduced by the following reduction. With a hash function, this reduction can be checked very efficiently.

Reduction 4.38 (Twin by Lamm et al. [149])

Let $u, v \in V$ have equal, independent neighborhoods $N(u) = N(v) = \{p, q, r\}$.

- *If $\omega(\{u, v\}) \geq \omega(\{p, q, r\})$, then include u and v.*

Reduced Graph	$G' = G - N[\{u, v\}]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(u) + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{u, v\}$

- *If $\omega(\{u, v\}) < \omega(\{p, q, r\})$ but $\omega(\{u, v\}) > \omega(\{p, q, r\}) - \min\{\omega(x) \mid x \in \{p, q, r\}\}$, then fold u, v, p, q, r into a new vertex v'.*

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus (N[v] \cup \{u\})]$ *with* $\omega(v') = \omega(\{p, q, r\}) - \omega(\{u, v\})$ *and* $N(v') = N(\{p, q, r\})$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(\{u, v\})$
Reconstruction	*If* $v' \in \mathcal{I}'$, *then* $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup \{p, q, r\}$, *else* $\mathcal{I} = \mathcal{I}' \cup \{u, v\}$

Reduction 4.38, which is a special case of Reduction 4.29, can also be applied if the neighborhood is larger, as mentioned in [149]. We generalize Reduction 4.38 to the case if the neighborhood if not an independent set. The main idea is that for two twins vertices, any maximal solution should include either both or none of these vertices. Therefore, we can fold the twin vertices. In special cases, we can also fold these twin vertices with their neighborhood.

Reduction 4.39 (Extended Twin)

Let $u, v \in V$ be non-adjacent and with equal neighborhoods $N(u) = N(v)$. Let further $\mathcal{I}_{N(v)}$ be an MWIS on $G[N(v)]$.

- *If* $\omega(u) + \omega(v) \geq \omega(\mathcal{I}_{N(v)})$*, then include* u *and* v*.*

Reduced Graph	$G' = G - N[\{u, v\}]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(u) + \omega(v)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{u, v\}$

- *If* $\omega(u) + \omega(v) < \omega(\mathcal{I}_{N(v)})$ *but* $\mathcal{I}_{N(v)}$ *is the only independent set in* $N(v)$ *with this property, then fold* u*,* v*, and* $N(v)$ *into* v'*.*

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus N[\{u, v\}]]$ *with* $N(v') = N(N[\{u, v\}])$ *and* $\omega(v') = \omega(\mathcal{I}_{N(v)}) - \omega(u) - \omega(v)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(u) + \omega(v)$
Reconstruction	*If* $v' \in \mathcal{I}'$*, then* $\mathcal{I} = (\mathcal{I}' \setminus \{v'\}) \cup \mathcal{I}_{N(v)}$*, else* $\mathcal{I} = \mathcal{I}' \cup \{u, v\}$

- *Otherwise, fold* u *and* v *into a new vertex* v'*.*

Reduced Graph	$G' = G[(V \cup \{v'\}) \setminus \{u, v\}]$ *with* $N(v') = N(\{u, v\})$ *and* $\omega(v') = \omega(v) + \omega(u)$
Offset	$\alpha_\omega(G) = \alpha_\omega(G')$
Reconstruction	*If* $v' \in \mathcal{I}'$*, then* $\mathcal{I} = \mathcal{I}' \setminus \{v'\} \cup \{u, v\}$*, else* $\mathcal{I} = \mathcal{I}'$

Proof. Let $u, v \in V$ be twin vertices, i. e., non-adjacent vertices such that $N(u) = N(v)$. Since they share the same neighborhood, no maximal solution only includes one of these vertices. Therefore, we can fold the two vertices into a new vertex v' with weight $\omega(v') = \omega(v) + \omega(u)$ as is done in the third case. For the first case, if $\omega(u) + \omega(v) \geq \omega(\mathcal{I}_{N(v)})$ and there is a solution $\mathcal{I}$ not including v', then we can always construct an equal or better solution $\mathcal{I}' = \mathcal{I} \setminus N(v) \cup \{u, v\}$ and therefore we can include the vertices u and v.

In the second case, we first assume that $v' \in \mathcal{I}$ and show that $\omega(\mathcal{I}_{N(v)}) + \alpha_\omega(G - N[N[v]]) \geq \omega(v) + \omega(u) + \alpha_\omega(G - N[v])$. This implies that the set $\mathcal{I}_{N(v)}$ is contained in some MWIS of G. Since $v' \in \mathcal{I}$, we know that

$$\begin{aligned}\omega(u) + \omega(v) + \alpha_\omega(G') &= \omega(u) + \omega(v) + \omega(v') + \alpha_\omega(G' - N_{G'}[v']) \\ &= \omega(u) + \omega(v) + \omega(\mathcal{I}_{N(v)}) - \omega(u) - \omega(v) + \alpha_\omega(G' - N_{G'}[v']) \\ &= \omega(\mathcal{I}_{N(v)}) + \alpha_\omega(G' - N_{G'}[v']).\end{aligned}$$

Since $\mathcal{I}'$ is an MWIS of G', we have $\omega(u) + \omega(v) + \alpha_\omega(G') \geq \omega(u) + \omega(v) + \alpha_\omega(G' - v') = \omega(u) + \omega(v) + \alpha_\omega(G - N[v])$. Now suppose that $v' \notin \mathcal{I}$. For this case we show $\omega(u) + \omega(v) + \alpha_\omega(G - N[v]) \geq \omega(\mathcal{I}_{N(v)}) + \alpha_\omega(G - N[N[v]])$, which implies that u

and v are in some MWIS of G. Since now $v \notin \mathcal{I}'$, we get $\omega(u) + \omega(v) + \alpha_\omega(G') = \omega(u) + \omega(v) + \alpha_\omega(G' - v') = \omega(u) + \omega(v) + \alpha_\omega(G - N[v])$. As $\mathcal{I}'$ is an MWIS of G', we also get

$$\begin{aligned}\omega(u) + \omega(v) + \alpha_\omega(G') &\geq \omega(u) + \omega(v) + \omega(v') + \alpha_\omega(G' - N_{G'}[v']) \\ &= \omega(u) + \omega(v) + \omega(\mathcal{I}_{N(v)}) - \omega(u) - \omega(v) + \alpha_\omega(G - N[N[v]]) \\ &= \omega(\mathcal{I}_{N(v)}) + \alpha_\omega(G - N[N[v]]).\end{aligned}$$

Since $\mathcal{I}_{N(v)}$ is the only independent set in $G(N[v])$ with higher weight than $\omega(u)+\omega(v)$ v is in some MWIS of G. We get $\alpha_\omega(G) = \omega(u) + \omega(v) + \alpha_\omega(G - N[v]) = \omega(u) + \omega(v) + \alpha_\omega(G')$. □

Note that the third case of Reduction 4.39 is already used in [150] but not introduced in this way by Lamm et al. [149]. Now we extend the idea of Reduction 4.39. For the following reduction rule, we no longer require the neighborhoods of the two vertices u and v to be equal but assume that $N(u) \subseteq N(v)$. If $\omega(u)+\omega(v) \geq \omega(N(v))$ the vertex u is always in an MWIS. The idea is that we can always replace vertices $x \in N(v)$ in a solution with u and v and thereby get an equal or better solution.

Reduction 4.40 (Almost Twin. Figure 4.7)

Let $u, v \in V$ be non-adjacent vertices such that $N(u) \subseteq N(v)$. If $\omega(u) + \omega(v) \geq \omega(N(v))$, then include u.

Reduced Graph	$G' = G - N[u]$
Offset	$\alpha_\omega(G) = \alpha_\omega(G') + \omega(u)$
Reconstruction	$\mathcal{I} = \mathcal{I}' \cup \{u\}$

Proof. Let $u, v \in V$ be non-adjacent vertices such that $N(u) \subseteq N(v)$ and $\omega(u) + \omega(v) \geq \omega(N(v))$. Assume there is an MWIS $\mathcal{I}$ of G not containing u. Then, there is a vertex $x \in N(u)$ such that $x \in \mathcal{I}$, which again results in $v \notin \mathcal{I}$. We can therefore construct a new solution $\mathcal{I}' = \mathcal{I} \setminus N(v) \cup \{u, v\}$ with $\omega(\mathcal{I}') \geq \omega(\mathcal{I})$. It follows that there always exists an MWIS that includes u. □

Remark 4.6. *The Reduction 4.40 can be extended as well. Instead of requiring that $\omega(u) + \omega(v) \geq \omega(N(v))$ the same reduction can also be applied if $\omega(u) + \omega(v) \geq \alpha_\omega(G[N(v)])$ since an MWIS for $N(v)$ is a subset of $N(v)$.*

Remark 4.7. *Note that applying Reduction 4.23 creates exactly the pattern needed for Reduction 4.40. If the weight constraint is satisfied, u and v are almost twins, and Reduction 4.40 can be applied.*

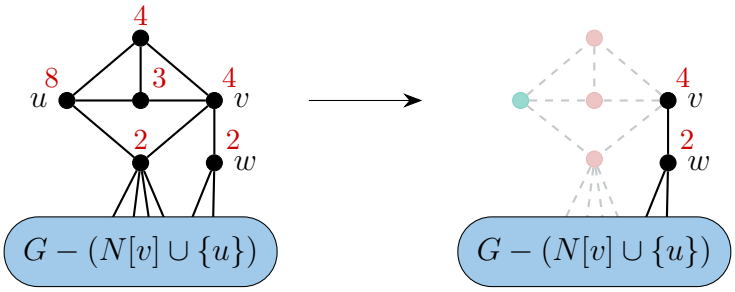


Figure 4.7: Illustration of the Almost Twin (Reduction 4.40). In the original graph on the left, $N(u) \subseteq N(v)$ with $\omega(u) + \omega(v) \geq \omega(N(v))$. By applying Reduction 4.40, u is included and its neighbors excluded from $\mathcal{I}$, resulting in the reduced graph on the right.

4.1.2 Discussion of Data Reductions in Practice

This section focuses on the practical application of data reductions for the MWIS and MWVC problems. First, we discuss the relations between the reductions and their computational cost in practice. Then, we survey the use of these data reduction rules in practical solvers developed for these problems.

Relations Between Data Reductions

Table 4.1: Overview of all data reduction rules grouped by their type. We give additional information about where they are (first) introduced and on what page of the paper. Reduction rules marked with *code* were not formally introduced in a paper but were implemented in the associated source code.

Ref.	Reduction Name	Type	Introduced By	In	At
4.1	Degree One	Low Degree	Gu et al. [107]	2021	p.4
4.2	Triangle	Low Degree	Gu et al. [107]	2021	p.5
4.3	V-Shape	Low Degree	Lamm et al. [149]	2019	p.7
			Gu et al. [107]	2021	p.5
4.4	3-Path	Low Degree	Xiao et al. [218]	2024	p.10
4.5	4-Path	Low Degree	Xiao et al. [218]	2024	p.12
4.6	4-Cycle	Low Degree	Xiao et al. [218]	2024	p.11
4.7	5-Cycle	Low Degree	Xiao et al. [218]	2024	p.13
4.8	6-Cycle	Low Degree	Xiao et al. [218]	2024	p.15
4.9	Heavy Vertex	Neighborhood	Lamm et al. [149]	2019	p.5

Continued on next page

Table 4.1 – *Continued from previous page*

Ref.	Reduction Name	Type	Introduced By	In	At
4.10	Neighborhood Removal	Neighborhood	Lamm et al. [149]	2019	p.6
4.11	Clique Neighborhood Removal	Neighborhood	Lamm et al. [150, 149]	2019	code
4.12	Neighborhood Folding	Neighborhood	Lamm et al. [149]	2019	p.5
4.13	Generalized Neighborhood Folding	Neighborhood	Lamm et al. [150, 149]	2019	code
4.14	Two Vertex Neighborhood Removal	Neighborhood	Zheng et al. [224]	2020	p.3
4.15	Heavy Set	Neighborhood	Xiao et al. [219]	2021	p.4
4.16	Heavy Set 3	Neighborhood	New		
4.17	Simplicial Vertex	Clique	Lamm et al. [149]	2019	p.6
4.18	Simplicial Weight Transfer	Clique	Lamm et al. [149]	2019	p.6
4.19	Weighted Funnel	Clique	New		
4.20	Domination	Domination	Lamm et al. [149]	2019	p.7
4.21	Basic Single Edge	Domination	Gu et al. [107]	2021	p.6
4.22	Extended Single Edge	Domination	Gu et al. [107]	2021	p.6
4.23	Extended Domination	Domination	New		
4.24	Extended Domination Reversed	Domination	New		
4.25	Original Weighted Struction	Struction	Gellner et al. [91]	2021	p.4
4.26	Modified Weighted Struction	Struction	Gellner et al. [91]	2021	p.4
4.27	Extended Weighted Struction	Struction	Gellner et al. [91]	2021	p.5
4.28	Extended Reduced Weighted Struction	Struction	Gellner et al. [91]	2021	p.5
4.29	Simultaneous Set	Global	Xiao et al. [219]	2021	p.4
4.30	Unconfined Vertices	Global	Xiao et al. [219]	2021	p.4
4.31	Extended Unconfined Vertices	Global	New		

Continued on next page

Table 4.1 – *Continued from previous page*

Ref.	Reduction Name	Type	Introduced By	In	At
4.32	Simultaneous Confined	Global	Xiao et al. [219]	2021	p.4
4.33	Uncovered Vertices	Global	Liu et al. [160]	2023	p.16
4.34	Simultaneous Cover	Global	Liu et al. [160]	2023	p.17
4.35	One Vertex Cut	Global	Xiao et al. [218]	2024	p.17
4.36	Two Vertex Cut	Global	Xiao et al. [218]	2024	p.19
4.37	Critical Weight Independent Set	Global	Butenko and Turkhanov [36]	2007	p.2
4.38	Twin	Twin	Lamm et al. [149]	2019	p.7
4.39	Extended Twin	Twin	New		
4.40	Almost Twin	Twin	New		

In the previous sections, we gave an overview of several different reduction rules with varying complexities, summarized in Table 4.1. Some of these rules are fast, e. g., Reduction 4.1, while others, if not bound, have exponential running time (Reduction 4.35). Furthermore, most of the reduction rules are special cases of other, more general reduction rules. This section discusses the relations between different reduction rules and gives a rough overview of their (practical) running times and complexities. Figure 4.8 presents most of the introduced reductions and their relations. The reduction rules are approximately sorted by decreasing practical running times from top to bottom. The most general rule for simultaneous sets is ranked highest and only added as a meta reduction since there is no efficient way of finding general simultaneous sets. The CWIS reduction has a polynomial running time but must be applied to the whole graph. Compared to other reductions that can be bounded and applied locally only for small neighborhoods, the CWIS reduction is more computationally expensive in practice, even if some of these other reductions have exponential running time if left unbounded (marked in yellow). Their performance heavily depends on the size bound for the subproblem to solve. Among these yellow-marked rules, we sorted them according to how often an independent set has to be solved on the bounded subgraph. For example, the heavy set and generalized fold reductions have to solve multiple MWIS and are therefore considered slower than the heavy vertex reduction. The Degree One, V-Shape, and Triangle are the fastest reduction rules. Note that all path and cycle rules in Section 4.1.1 are covered by Triangle and V-Shape, but not necessarily faster and therefore omitted in this figure.

The arrows in Figure 4.8 describe the relation between different reduction rules. These are always directed from a *general* to a *special* case. Note the use of transitivity

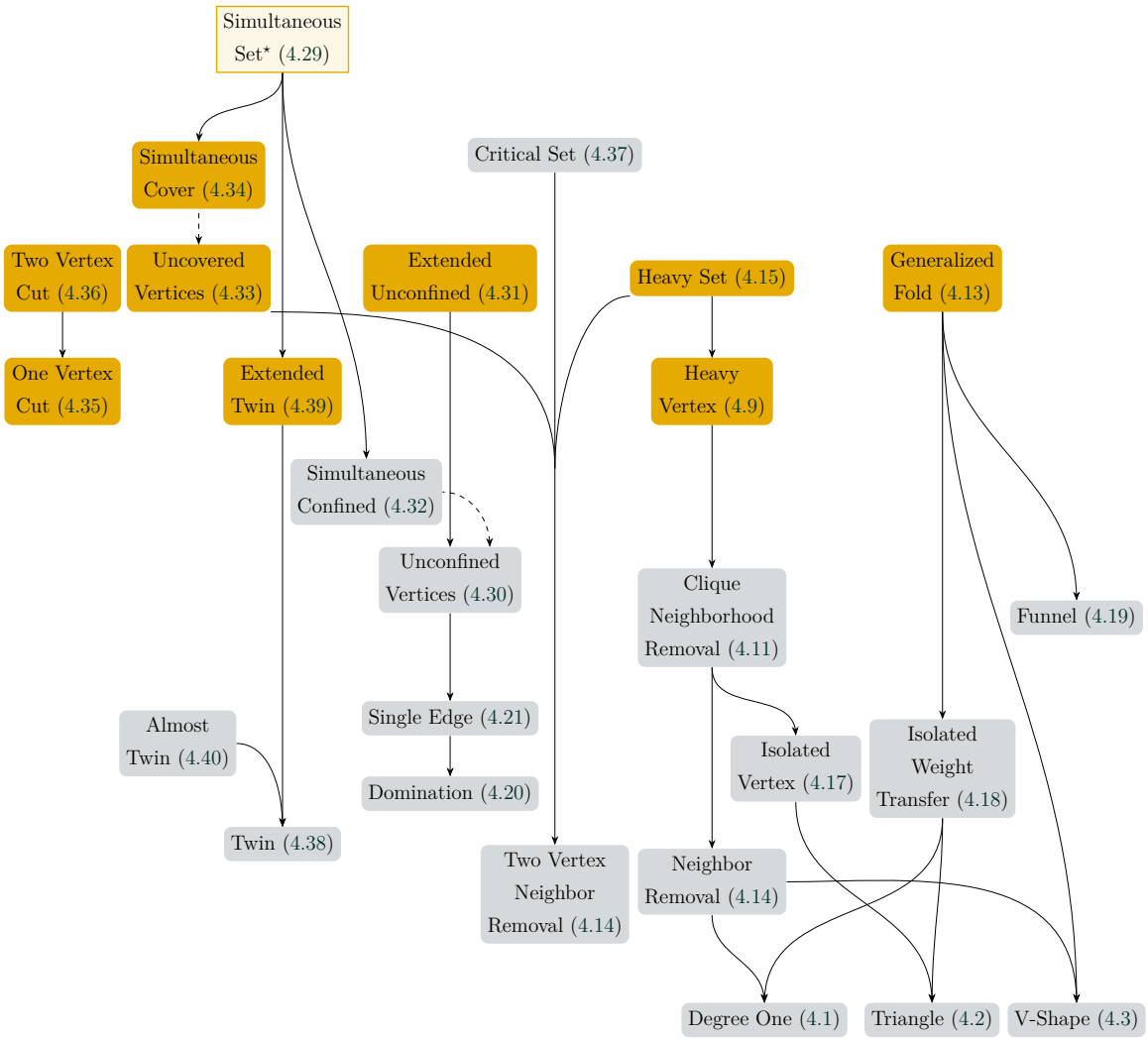


Figure 4.8: This figure gives an overview of the presented reduction rules and their relations. Two rules, A and B, are connected with an arrow from A → B if rule A is a more generalized form and can also reduce the patterns reduced by Reduction B. A dashed arrow indicates that the rule B is part of rule A. The rules are intuitively sorted by complexity, starting with the more computationally expensive rules at the top. Hence, more general rules are always above their special case rules. Note that even though the critical set rule can be applied in polynomial time, it has to be applied to the whole graph. That makes the rule more computationally expensive than other rules that have to solve the MWIS on a bounded subgraph, marked in yellow. The Simultaneous Set rule is only added as a meta reduction and is not implemented in the most general case. The struction-based rules are omitted since they transform the graph, so they are not easily comparable to the other reduction rules.

for this relation; therefore, some edges are omitted. For example, the Heavy Vertex rule covers the Clique Neighborhood Removal rule, which again covers the simpler Neighborhood Removal rule. Because of this, Neighborhood Removal is also a special

Table 4.2: Presentation of groups of data reduction rules used by different algorithms with implementations and experimental evaluation, sorted by year when the solver was first published. The reductions combined in the different groups are mentioned in the parenthesis after the group name.

Year	Algorithm	Degree One	Degree Two	Neighborhood	Domination	Clique	Twin	Unconfined	CWIS	Struction	Heavy Set	Cut
2018	NuMWVC [156]	✓	✓									
2019	KaMIS BnR [149]	✓	✓	✓	✓	✓	✓		✓			
2019	BMWVC [211]	✓	✓	✓								
2020	MAE-HTS [212]	✓		✓								
2020	DtTwo [224]			✓								
2021	Solve [219]		✓	✓			✓	✓	✓		✓	
2021	HtWIS [107]	✓	✓		✓							
2021	Struction [91]	✓	✓	✓	✓	✓	✓		✓	✓		
2022	GNN-VC [153]	✓	✓	✓	✓	✓	✓		✓			
2023	m^{2}wis [102]	✓	✓	✓	✓	✓	✓		✓	✓	✓	
2024	C-B&R/C-Search [160]							✓				
2024	HGLV [200]	✓		✓								
2024	LearnAndReduce [104]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Group	Reductions
Degree One	4.1
Degree Two	4.2-4.8
Neighborhood	4.10 -4.14
Domination	4.20, 4.21, 4.22
Clique	4.17, 4.18
Twin	4.38, 4.39, 4.40
Unconfined	4.30-4.34
CWIS	4.37
Struction	4.25-4.28
Heavy Set	4.15
Cut	4.35, 4.36

case of Heavy Vertex. However, this transitivity does not apply to dashed arrows. For example, the Uncovered Vertices rule is used to compute the covering sets in Simultaneous Cover, a special case of Simultaneous Set. However, the more general Simultaneous Set rule can not necessarily reduce the patterns that the Uncovered Vertices rule does. Furthermore, if a reduction rule has multiple cases, we add an arrow if only one is covered. For example, we need the Generalized Fold and the Neighborhood Removal rules to cover the V-Shape reduction fully. Looking at the relations between the different rules, we see that the Heavy Set and Generalized Fold reductions are very powerful and cover all the low degree, clique, and neighborhood-based reduction rules. Then, there are reductions derived from the simultaneous set meta reduction, cut reductions, and the reductions Unconfined Vertices and Uncovered Vertices, which are not special cases of other rules.

Practical Use of Data Reductions in Different Solvers

We now examine which data reduction rules are utilized and evaluated in practical implementations. In Table 4.2, we list all the algorithms that use data reduction rules for solving the Maximum Weight Independent Set or Minimum Weight Vertex Cover problems in practice. For each solver, we mark which rules are utilized.

The most commonly used reduction rules are the degree bounded rules (Degree One, V-Shape, and Triangle) and the Neighborhood Removal rule. These rules are fast to compute and effective in practice. After these simple rules, domination-based reductions are applied most often. These reduction rules are used in 5 of the 12 algorithms considered. The clique-based reductions and the Critical Weight Independent Set reductions are used in four different solvers.

The reduction rules Struction, Unconfined, and Heavy Set are used only in two algorithms. The cut-based rules, introduced in a theoretical paper [218], are not used in any practical solver. An explanation for why only a few implementations use these rules could be that they are more computationally expensive than the more popular reductions.

4.1.3 Experiments on Ordering of Reduction Rules

This section presents the experimental evaluation of the effect of different orderings in which data reductions are applied. We examine the impact of different orderings on solution size and running time. One outcome of this evaluation is robust orderings of reductions for exact reduction rules and a specific ordering that can improve the solution quality at the expense of computation time.

Different orderings of applying data reductions yield different sizes of the reduced graph. Additionally, the ordering impacts the running time of the reduction process. For example, this effect has been described by Figiel et al. [76]. In this section, we perform experiments to evaluate the impact different orderings may have. Therefore, we run the ExactReduce routine (see Algorithm 1), i. e., we apply the reductions in a given ordering exhaustively and report the results.

Methodology. We implemented our algorithm using C++11. The code is compiled using g++ version 12.2 and full optimizations turned on (-O3). For the experiments, we use Machine 1. We run each configuration with four different seeds and a time limit of ten hours. We always report the geometric mean results if not stated otherwise.

Instance Set 1 (Main)**.** *This set consists of 207 graphs from different sources. It contains large social networks from the Stanford Large Network Dataset Repository (`snap`) [155]. Additionally, real-world graphs from OpenStreetMaps (`osm`) [1, 20, 37]. Furthermore, as in Gu et al. [107] we took the same 6 graphs from the SuiteSparse Matrix Collection (`ssmc`) [53] where weights correspond to population data. Each weight was increased by one to avoid many vertices assigned zero weight. Additionally, we used instances from dual graphs of well-known triangle meshes (`mesh`) [185], as*

well as 3d meshes derived from simulations using the finite element method (`fe`) [210]. For unweighted graphs, we assigned each vertex a random weight uniformly distributed in the interval [1, 200]. We list all graphs in Table A.1 in the Appendix.

Initial Ordering as a Baseline

Our starting point is an intuitive ordering, which we constructed by the simplicity of the reductions, from simplest to most complex. Hereby, we orientate towards the ordering by Akiba and Iwata [4]. This initial ordering is given in Reduction List 4.1. The following experiments use it as a *baseline* for comparisons. Note that we did not use the full set of reduction rules introduced in Section 4.1.

Reduction List 4.1.

$$R_{4.1} \coloneqq [4.1, 4.10, 4.2, 4.3, 4.17, 4.21, 4.22, 4.38, 4.18, 4.37, 4.12, 4.15]$$

Orderings Based on Impact of Single Reductions

Since the space of all possible orderings of data reductions is too large to evaluate exhaustively, we make different restrictions. First, we start our evaluation by examining the impact of disabling single reductions from Reduction List 4.1. In detail, we run ExactReduce with Reduction List 4.1 and then build a reduction list where exactly one data reduction of the Reduction List 4.1 is disabled. The ordering of the remaining data reductions has not been changed. The time to apply all reductions exhaustively using our baseline ordering is denoted as $t_{R_{4.1}}$ and the size of the reduced instance by those reductions is denoted as ω_{all}. Then, we create different data reduction orderings from the baseline in which a single data reduction is disabled. For each of the available reductions r, we get a new time $t_{R_{4.1}\setminus r}$ and solution weight $\omega_{R_{4.1}\setminus r}$ which corresponds to running all reductions except r of the baseline (and in its order). Based on these values, we derive three orderings: a time-based ordering, a size-based ordering, and a combination of both.

Time-based Ordering. For the *time-based* ordering we rearranged the reductions such that the mean $\bar{t}_{R_{4.1}\setminus r}$ is decreasing. The intuition here is that if removing a reduction from the baseline yields an ordering that has an excessive running time, then this reduction is important for running time and should be applied before a reduction that has a smaller impact. Reductions with only small effects, where the mean solution quality $\overline{\omega_{R_{4.1}\setminus r}}$ is equal to the mean solution quality for $\overline{\omega_{R_{4.1}}}$, are disabled to reduce the running time further. This results in the ordering Reduction List 4.2.

Reduction List R_i	$t_{R_i}/t_{R_{4.1}}$	$n(\mathcal{K}_{R_i})/n(\mathcal{K}_{R_{4.1}})$	# best
$R_{4.1}$ (initial)	1.00	1.000	180/207
$R_{4.2}$ (time)	1.06	1.021	161/207
$R_{4.3}$ (size)	1.46	1.074	164/207
$R_{4.4}$ (time&size)	1.60	0.998	191/207

Table 4.3: Comparing the geometric mean of running times and reduced graph sizes for different orderings relative to the initial ordering, $n(\mathcal{K})$ is the number of vertices in the reduced graph. Additionally, we count how often an ordering found the smallest reduced graph compared to the other orderings (# best).

Reduction List 4.2.

$$R_{4.2} \coloneqq [4.3, 4.21, 4.38, 4.3(case3), 4.10, 4.22, 4.1, 4.17, 4.2, 4.37, 4.15]$$

Size-based Ordering. For the *size-based* ordering, the reductions are rearranged in decreasing order according to the mean value $\overline{\omega}_{R_{4.1}\setminus r}$ over all graphs. The intuition here is that if $\overline{\omega}_{R_{4.1}\setminus r}$ is large, then not using r has a large impact on the size of the reduced instance and hence should be applied before a reduction that has a smaller impact. The resulting ordering is given in Reduction List 4.3.

Reduction List 4.3.

$$R_{4.3} \coloneqq [4.3, 4.21, 4.37, 4.38, 4.3(case3), 4.15, 4.17, 4.2, 4.1, 4.10, 4.22, 4.11, 4.12]$$

Time and Size-based Ordering. Here we use a combination with $x_{R_{4.1}\setminus r} = \overline{t}_{R_{4.1}\setminus r} + 10 \cdot \overline{\omega}_{R_{4.1}\setminus r}$ decreasingly to order reductions. We use a factor of 10 here since solution quality is typically more important for applications than running time. This results in the ordering Reduction List 4.4.

Reduction List 4.4.

$$R_{4.4} \coloneqq [4.3, 4.21, 4.37, 4.38, 4.17, 4.3(case3), 4.15, 4.2, 4.1, 4.10, 4.22, 4.11, 4.12]$$

Discussion. The results for the previous orderings are presented in Table 4.3. Different orderings do not yield significant differences compared to the initial ordering. We observe that the Reduction List 4.2 (time) as well as the Reduction List 4.3 (size) are not able to improve either the size of the reduced instance or the computation

time. The Reduction List 4.4 (time&size) can compute smaller instances at the expense of an additional 60 % of running time. With this, we are able to find 191 out of 207 smallest reduced instances. However, the geometric mean size improvement compared to Reduction List 4.1 is less than 0.2 %. The experiments show that this initial ordering already yields a good trade-off for running time and reduced instance size.

When examining the positions for the reductions, we note that some reductions remain at approximately the same position, meaning they are either very important for solution size and quality or the opposite. For example Reductions 4.3 and 4.21 are applied at the beginning, whereas, for example, Reduction 4.11 is applied towards the end or removed. On the other hand, there also are reductions that are on completely different positions, e. g., Reduction 4.37. The findings of these first experiments are summarized in the following observation.

Observation 4.1.1: Time and Size-Based Orderings of Reductions. Our experiments show that the reduction order affects the size of the reduced instance and, especially, the running time. The Reduction List 4.1 with the initial ordering of the reductions is already robust w.r.t. the orderings considered in this section. The Reduction List 4.4 can be beneficial on some instances, however, it is also the most expensive regarding running time.

Orderings Based on Impact of Groups of Reductions

In the following experiments, we further want to examine the solution space more broadly. This we do by looking at the performance of different permutations. Since the total number of possible permutations is too high, we add some additional restrictions.

Therefore, we divide the reductions into three groups of roughly similar complexity. We examine each of these groups individually by only permuting the order in that group. The groups and their position relative to other groups always stay the same.

The first group contains Reduction 4.1 and Reduction 4.10, while the second group consists of reductions for vertices of degree two, which are Reductions 4.2 and 4.3. The third group contains all remaining reductions that are listed in Reduction List 4.1.

Permutations in the First and Second Group. We now examine all permutations in the first two groups. Since the groups always stay in a fixed order, Reductions 4.1 and 4.10 are always the first two reductions applied. Reductions of the third group are applied as in the Reduction List 4.1. Overall, our experiments show that the order of the reductions in the first two groups has a negligible effect on the run-

ning time and quality of a solution. Thus, we use the initial ordering for reductions in these groups for the remaining experiments.

Permutations in the Third Group. We now examine permutations in the third group of reductions. Here, we furthermore restrict the number of permutations by applying Reduction 4.12 always at the end and Reduction 4.21 always followed by Reduction 4.22. The best-performing permutation is given in Reduction List 4.5.

Reduction List 4.5.

$$R_{4.5} \coloneqq [4.1, 4.10, 4.2, 4.3, 4.3(case3), 4.17, 4.38, 4.37, 4.11, 4.21, 4.22, 4.12]$$

The geometric mean weight improvement found is $w_{R_{4.5}}/w_{R_{4.1}} = 1.002{,}3$ and the geometric mean time compared to the initial ordering is $t_{R_{4.5}}/t_{R_{4.1}} = 2.9$. We summarize the results of the ordering experiments in Observation 4.1.2.

Observation 4.1.2: Permutation Orderings for Reductions. Overall, some orderings perform better than the initial ordering by complexity. These improvements are only observed on a few instances and result in a significant increase in running time. In most cases, all orderings yielded similar results. Among those, the initial ordering remains one of the fastest.

We conclude that the Reduction List 4.1 with the initial ordering presents a very stable reduction ordering. Nevertheless, for some graphs, it might be worth trying multiple runs of ExactReduce using one of the other orderings we presented in this section as well.

4.2 GNN Guided Preprocessing LearnAndReduce

In this section, we introduce an advanced, exact preprocessing tool LearnAndReduce that employs Graph Neural Networks (GNNs) to decide where to apply data reduction rules. In Section 4.2.1, we introduce the different GNN models evaluated for this task. We then present a new dataset for supervised learning in Section 4.2.2. Finally, we evaluate the performance of the LearnAndReduce preprocessing in Section 4.2.4. Here, we evaluate first the GNN architectures and then the overall performance of the LearnAndReduce preprocessing.

References. *This section is based a publication which is joint work with Kenneth Langedal and Christian Schulz [104]. Large parts are copied verbatim from the paper.*

Our Results. With our GNN filtering, we can now apply reduction rules that were previously unused in other works due to the computational cost required to determine their applicability, as discussed in Section 4.1.2. In particular, we present a new dataset with labeled vertices for the problem of early reduction rule screening. The dataset contains two collections of graphs, one with unreduced instances and one after running a set of fast reductions. The second is the one we use for the early reduction rule screening, and it consists of more than one million labeled vertices. Using LearnAndReduce, we can reduce our instances to within one percent of what is possible using the full set of reduction rules while spending less than 23 % of the corresponding time. These results are adjusted for fast reduction rules that would always be applied.

Some of the reductions previously introduced need to solve additional MWIS problems on subgraphs, if these rules are not bounded, preprocessing using these rules could take exponential time. For practical use, the more costly reductions are limited in some way, especially for large instances. This is done, for example, by limiting the degree of the vertex to apply the reduction on, the subgraph size, or the time spent on each vertex. In Section 4.2.1, we introduce a new GNN application in the form of early data-reduction screening. We also provide a new and openly available supervised-learning dataset with graphs and labeled vertices for each reduction rule. At inference, for each expensive data reduction rule, a pre-trained GNN model decides what vertices should be checked for the applicability of the rule.

4.2.1 Graph Neural Network Models

In this section, we introduce a new screening method based on Graph Neural Networks (GNN). This method decides when to apply data reductions. We also provide a new and openly available supervised-learning dataset with graphs and labeled vertices for each reduction rule. For each expensive reduction rule used in the preprocessing, a pre-trained GNN model will limit the set of vertices that should be checked for using the reduction rule. GNNs are recent additions to the field of artificial intelligence that bring successful ideas from conventional deep learning to the irregular structure of graphs [217]. Where traditional deep learning focuses on structured input, such as the grid of pixels in an image, GNNs accept the unstructured data of a graph.

As a first stage of this task, we evaluate the most popular GNN architectures used in combinatorial optimization, namely Graph Convolutional Network (GCN) [139] and Graph Sample and Aggregate (GraphSAGE) [115]. In addition to GCN and GraphSAGE, we also introduce a slightly altered GNN architecture, which we name the LearnAndReduce-model (LR). At a high level, these GNN architectures combine

conventional neural networks with message passing. In this *message passing*, every vertex sends messages to its neighbors and aggregates the received messages. One iteration of message passing and a neural network transformation makes up a GNN *layer*. Several of these layers are stacked on top of each other to make up a GNN model. The goal is that after these message-passing layers, the final vertex embedding can be used to estimate how likely a reduction rule will succeed at this vertex.

An undirected graph $G = (V, E, H)$ is given as input to the model, where H is the initial feature representation for the vertices in the graph. Any number of vertex features can be used. Using the vertex weight is an obvious choice, but additional features such as vertex degrees and local clustering coefficients are common [154]. Note that computing complicated features for the vertices can add significant computational overhead to the model, which is important for our application since running the model should not take longer than checking the reduction rules. This feature representation will change after each layer in the model. For a vertex $u \in V$ at layer l, the feature representation is denoted by $H_u^{(l)}$. The length d of the feature vector at layer l is denoted by $d^{(l)}$. Stacking all the feature vectors at the l'th layer gives the matrix $H^{(l)} \in \mathbb{R}^{|V| \times d^{(l)}}$. Independent from any input graph, every layer in the GNN model has trainable parameters $W^{(l)}$, bias $b^{(l)}$, and a non-linear activation function σ. Every model uses the $\text{ReLU}(x) \coloneqq \max(0, x)$ activation function for internal layers and $\text{Sigmoid}(x) \coloneqq \frac{1}{1+e^{-x}}$ for the output layer. Note that these activation functions are applied element-wise when the input is a vector. With this, we define each model used for testing.

Graph Convolutional Network. The GCN architecture was the first successful extension of convolutional neural networks to work directly on graphs. At each layer in a GCN model, every node aggregates information from its immediate neighbors and combines it with its own data. After this, the information stored in each node is passed through the layer-specific neural network to create the node information for the next layer. The layer-wise propagation rule can be seen in Algorithm 7. Not that the GCN model assumes that there are self-edges added to the input graph. If not, a vertex will not include its own feature representation in the neighborhood aggregation.

Algorithm 7: GCN propagation rule. Self-edges are added to the input graph, and T_u is a temporary variable holding the aggregated feature vectors of the neighbors of u.

for $u \in V$ **do**
  $T_u \leftarrow \sum_{v \in N(u)} H_v^{(l)} / \sqrt{|N(v)|}$
  $H_u^{(l+1)} \leftarrow \sigma(W^{(l)} \cdot T_u + b^{(l)})$

Graph Sample and Aggregate. GraphSAGE expands on GCN in two notable ways. First, it can use any differentiable aggregation function, such as aggregating the neighborhood based on mean, max, sum, or even more complicated functions, such as a long short-term memory (LSTM) machine-learning model. Second, GraphSAGE concatenates the feature vector of a vertex with the aggregated information from its neighborhood. This is in contrast to the self-edges of the GCN and allows information to skip from one layer to the next without going through the neighborhood aggregation. The general layer-wise propagation rule can be seen in Algorithm 8.

Algorithm 8: GraphSAGE propagation rule. T_u is a temporary variable holding the aggregated feature vectors of the neighbors of u.

for $u \in V$ **do**
- $T_u \leftarrow \text{AGGREGATE}(\{H_v^{(l)} \mid v \in N(u)\})$
- $H_u^{(l+1)} \leftarrow \sigma(W^{(l)} \cdot \text{CONCAT}(H_u^{(l)}, T_u) + b^{(l)})$

Learn and Reduce. The proposed LearnAndReduce (LR) architecture differs slightly from the GCN and GraphSAGE architectures. Instead of applying the weighted transformation after the aggregation, it is applied during the neighborhood aggregation. Algorithm 9 gives the exact layer-wise propagation rule. To give an intuition for why this approach could learn reduction rules better than GCN and GraphSAGE, consider the case of the extended single-edge reduction. For this reduction rule, we are looking for two adjacent vertices $u, v \in V$ such that $\omega(v) \geq \omega(N(v)) - \omega(u)$. For more information on this reduction rule, see Table 4.4 or Gu et al. [107]. Assuming the node weights and neighborhood weights are given as input features, the LR architecture could conceivably detect this pattern during the first layer of the model since the concatenated feature representation of u and v would contain all the necessary information to decide if the rule can be applied. In contrast, GCN and GraphSAGE would aggregate the entire neighborhood before the first weighted transformation, potentially obscuring the necessary information required to make the correct prediction. This is not a novel GNN architecture and could arguably fit into the GraphSAGE framework.

Algorithm 9: LearnAndReduce propagation rule. The main difference with this architecture compared to GCN and GraphSAGE is the use of weighted transformation during neighborhood aggregation.

for $u \in V$ **do**
- $H_u^{(l+1)} \leftarrow \text{MEAN}(\{\sigma(W^{(l)} \cdot \text{CONCAT}(H_u^{(l)}, H_v^{(l)}) + b^{(l)}) \mid v \in N(u)\})$

4.2.2 Training Data Generation

The supervised-learning dataset contains each instance twice, first the original instances without any reductions applied, and second the same instance after applying a set of fast reductions. These fast reductions are not considered relevant for early screening because they are so computationally cheap that we would always check them exhaustively. Table 4.4 gives a summary of these reductions. We refer to these two versions of the same instance as ORIGINAL and REDUCED. The truth labels for each reduction rule are generated by testing the rule on each vertex without performing the reduction in the successful case. This way, each label is created using a clearly defined procedure without ambiguity. For the most costly reduction rules that rely on solving MWIS instances in subgraphs, a third class is used for a timeout event. This gives the following labeling for each combination of a vertex and reduction rule.

$$\text{LABEL}(v) = \begin{cases} 0 : \text{Unsuccessful reduction} \\ 1 : \text{Successful reduction} \\ 2 : \text{Timeout} \end{cases}$$

It is not always clear what vertex should be labeled as successful, especially for reductions where the rule starts from one vertex but ends up reducing another. It would be easiest if the starting vertex always received the successful label, given that the purpose of the trained model is to perform early screening. However, this would make the task of learning the rules unnecessarily hard for some reduction rules. The complete list of reduction rules used and how the labels are generated can be seen in Table 4.4. Note that the labeled vertices are always chosen to make the training as simple as possible. For example, the heavy set reduction starts from a source vertex and looks for two vertices in the neighborhood that can be included. If the successful labels were placed on the source vertices, the model would need to detect vertices adjacent to two vertices likely to be part of a solution. In contrast, if the labels are placed on the heavy vertices themselves, the pattern is simply to detect vertices that are likely part of a solution. We can still use the trained model for screening, but instead of deciding what vertices to start the reduction from, we only use suggested vertices from the neighborhood of the source vertex.

4.2.3 The LearnAndReduce Approach

In the following, we describe our LearnAndReduce approach in more detail. In addition to the new reduction rules introduced in this dissertation, we also use several other data reduction rules that where introduced in Section 4.1. The complete list

and a short description of each rule used is also given in Table 4.4. The most costly reduction rules are in the second part of this table, below the thick line. These reductions undergo early GNN screening before applying them. Each reduction rule has its own pre-trained GNN model for early screening. Since we use the Sigmoid activation function, the output from the GNN is a real number in the range from zero to one. If the GNN prediction for a vertex is greater than 0.5, that vertex is checked using the reduction rule for which the model was trained.

The LearnAndReduce reduction procedure maintains a queue of vertices to check for each reduction rule. The reduction rules are ordered based on complexity, and each rule is only checked when all the queues of easier reductions are empty. The ordering is the same as shown in Table 4.4, except for Reduction 4.24, which is applied before Reduction 4.13. Even if it is computationally more expensive, it yielded better results in our tests to place Reduction 4.37 before applying the struction approach introduced by Gellner et al. [91]. An outline of how the LearnAndReduce reduction procedure works is shown in Algorithm 10. Each queue is initialized with all the vertices from the graph. Whenever a successful reduction occurs, the adjacent vertices that saw a change in their neighborhood are queued up again for every reduction. This way, the reduction procedure always returns to the easier reductions as soon as possible. The vertices are not ordered within the queues beyond the first-in, first-out principle. Reduced vertices may still reside in other queues but are ignored when they are popped from the queue.

There are multiple ways to incorporate the GNN screening. Since the graph continuously changes during the reduction process, the initial predictions might not be relevant later. Therefore, we propose three GNN screening configurations.

Always. In this configuration, the queue for each expensive reduction is only checked to see if it is non-empty. If that is the case, the queue is always replaced by new suggestions from the GNN model. This means the GNN model could be evaluated multiple times during the reduction process.

Initial. Instead of always evaluating the GNN model, the Initial configuration only evaluates the GNN model the first time the rule is used. Recall that each queue is initialized with all the vertices in the graph, meaning that the first time the rule is used is also when the potential screening effect is the highest. After this initial screening, the reduction rule goes back to working as without GNN screening, meaning if further reductions occur and vertices are added back to the queue, then these are checked fully.

Algorithm 10: The LearnAndReduce reduction procedure.

Data: Graph $G = (V, E, \omega)$, set of reductions R, and set of GNN models M
Result: Reduced instance G

```
i ← 0
Q ← {{0, 1, ..., |V| − 1} · |R|}          // Initialize FIFO queues for each rule
while i < |R| do
    if Q[i] = ∅ then
        i ← i + 1
        // With the Initial/Initial-Tight configurations, the next If statement
         is changed accordingly
        if i < |R| and Q[i] ≠ ∅ and M[i] exists then
            Q[i] ← ∅
            P ← M[i](G)
            for v ∈ V such that P[v] > 0.5 do
                push(Q[i], v)
    else
        v = pop(Q[i])
        if v is not reduced then
            G′ ← R[i](G, v)                          // Try reduction on v
            if G′ ≠ G then
                G ← G′                               // Successful reduction
                i ← 0
                for each changed vertex u ∈ V and q ∈ Q do
                    if u ∉ q then
                        push(q, u)
```

Table 4.4: Full list of reductions used in LearnAndReduce along with a brief description. The last two columns connect the reduction rules to the supervised-learning dataset. We often combine similar reduction rules for the dataset. The bold vertical line in the center of the table indicates the distinction between computationally cheap and expensive reduction rules.

Id	Name	Brief Description	Data Name	Label
4.10	Neighborhood Removal [149]	Includes a vertex v if $\omega(v) \geq \omega(N(v))$	Neighborhood Removal	Included vertex
4.1	Degree One [107]	Reduces or folds all degree-one vertices	Degree-One	Degree-one vertex
4.2	Triangle [107]	Reduces degree-two vertices with adjacent neighbors	Degree-Two	Degree-two vertex
4.3	V-Shape [149, 107]	Reduces degree-two vertices with non-adjacent neighbors		
4.17	Simplicial Vertex [149]	Includes a vertex v if $N(v)$ forms a clique and $\omega(v) \geq \max_{u \in N(v)} \omega(u)$	Clique	Simplicial vertex
4.18	Simplicial Weight Transfer [149]	Folds a vertex v if $N(v)$ forms a clique and $\omega(v) \leq \max_{u \in N(v)} \omega(u)$		
4.20	Domination [149]	Exclude a vertex that dominates a lighter vertex	Domination	Excluded vertex
4.21	Basic Single Edge [107]	Exclude a vertex v if for some vertex $u \in N(v)$, $\omega(N(u) \setminus N(v)) \leq \omega(u)$	Single Edge	Excluded vertex
4.22	Extended Single Edge [107]	For $\{u, v\} \in E$, exclude $N(u) \cap N(v)$ if $\omega(v) \geq \omega(N(v)) - \omega(u)$	Extended Single Edge	Excluded vertices

Continued on next page

Table 4.4 – *Continued from previous page*

Id	Name	Brief Description	Data Name	Label
4.38	Twin [149]	Limited to independent neighborhood	Twin	Both twins
4.39	Extended Twin	Fold or reduce all twins		
4.40	Almost Twin	Includes a vertex v if for some non-adjacent u, $N(v) \subseteq N(u)$ and $\omega(v) + \omega(u) \geq \omega(N(u))$	Almost Twin	Included vertex
4.19	Weighted Funnel	For $\{u, v\} \in E$ with $\omega(v) = \max\{\omega(w) \mid w \in N(v) \setminus \{u\}\}$ and $N(v) \setminus \{u\}$ forms a clique, we can fold or reduce parts of $N(v)$	Funnel	Almost simplicial vertex
4.11	Clique Neighborhood Removal [149]	Extension of Reduction 4.10 for cliques in the neighborhood of v	—	—
4.23	Extended Domination (Reverse)	For $\{u, v\} \in E$ with $N[u] \subseteq N[v]$ and $\omega(u) < \omega(v)$, remove $\{u, v\}$ and set $\omega'(v) = \omega(v) - \omega(u)$	—	—
4.24	Extended Unconfined	Exclude a vertex v if a contradiction is obtained by assuming v is part of every MWIS	Unconfined	Excluded vertex
4.37	Critical Weight Independent Set [36]	Included the vertices of a (possibly empty) independent set $\mathcal{I}_c$ such that $\omega(\mathcal{I}_c) - \omega(N(\mathcal{I}_c)) = \max\{\omega(\mathcal{I}) - \omega(N(\mathcal{I})) \mid \mathcal{I} \text{ is an IS of } G\}$	Critical Set	Included vertices

Continued on next page

Table 4.4 – *Continued from previous page*

Id	Name	Brief Description	Data Name	Label
4.28	Extended Reduced Struction [91]	A vertex v and its neighborhood can be transformed into a clique where each vertex represents an independent set in $N[v]$	—	—
4.13	Generalized Neighborhood Folding [149, 150]	Extension of Reduction 4.10 by solving the MWIS in the neighborhood of v	Generalized Fold	Included vertex
4.15	Heavy Set [219]	Finds a pair of vertices that can be included based on their combined neighborhoods	Heavy Set	Included vertices
4.16	Heavy Set 3	Extension of Reduction 4.15 for three vertices instead of two	Heavy Set 3	Included vertices
4.35	Cut Vertex [218]	Fold or reduce an articulation point v along with the smaller component in $G - v$	Cut Vertex	Articulation points

Initial Tight. Again, the GNN model is only evaluated the first time the reduction rule is used. However, instead of returning to the normal queue-based application afterward, the queue cannot accept any additional vertices. This means that the initial suggestions from the GNN model are the only vertices that will be checked using that reduction rule. Intuitively, this configuration is the most aggressive form of early screening and will take the least amount of time.

For two reductions, namely CWIS by Butenko and Turkhanov [36] and One Vertex Cut by Xiao et al. [218], the queue-based approach does not fit so well. The Critical Set reduction needs to construct a new bipartite graph and compute the maximum flow, and the Cut Vertex reduction needs to find all articulation points. Therefore, it is natural to apply these reductions globally. Regarding the reduction queues, these two reductions still have the same queues as the other reductions. However, as long as the queue is non-empty, the reduction is applied globally, and the entire queue is cleared afterward. This also extends to the GNN screening, but we only apply the reduction globally if the GNN suggests applying the rule on more than 1 % of the graph.

4.2.4 Experimental Evaluation

This section presents the experimental evaluation of LearnAndReduce. First, we present training results and then how LearnAndReduce performs combined with an MWIS solver to compare different configurations.

Methodology. The code is implemented using C++11 and compiled using g++ version 12.2 and full optimizations turned on (-O3). To ensure fairness between the algorithms, the instances start in the same order for each code, and only one program is evaluated at a time. We evaluate each program once for each instance. The experiments for the LearnAndReduce preprocessing are performed on Machine 2 using the Instance Set 2.

Instance Set 2 (Main - hard to reduce)**.** *This set is a subset of Instance Set 1 and consists of 83 graphs. In this set, we remove all instances, which were fully reducible by simple and fast reduction rules*[1]*. This data set is presented in detail in Table A.2.*

[1]We excluded instances which were fully reduced by running the following reductions in the given order: Reduction 4.10, Reduction 4.1, Reduction 4.2, Reduction 4.3, Reduction 4.17, Reduction 4.20, Reduction 4.21, Reduction 4.22, 4.38, Reduction 4.39, Reduction 4.19, Reduction 4.11, Reduction 4.23, Reduction 4.28 (without blow-up)

Training Results

We evaluate three GNN architectures on our new supervised learning task: GCN, GraphSAGE, and LR. Each architecture is configured to have a similar number of layers, parameters, and running time. We use two message-passing layers, followed by two layers of only weighted transformation. The input to the final two layers is a concatenation of all intermediate feature vectors, including the input features. The size of each intermediate feature vector is 16, and the size of the input vector is 8. The activation function ReLU is used for internal layers, and Sigmoid is used for the output layer. After each message passing layer, we perform random dropout with a probability of 0.2 during training. We use weighted binary cross entropy loss for loss function to adjust for the rarity of vertices labeled as successful. The weight is set exactly to the ratio between successful and unsuccessful labels in the training set. Finally, the Adam optimizer [138] is used with its default hyperparameters and 0.001 learning rate. The input features for one vertex are as follows.

1. Vertex weight
2. Neighborhood weight
3. Minimum neighborhood weight
4. Maximum neighborhood weight
5. Vertex degree
6. Average neighborhood degree
7. Minimum neighborhood degree
8. Maximum neighborhood degree

It is important to note that these reduction rules can mostly be checked in polynomial time already. Therefore, it would defeat the purpose of early screening if the screening took more time than the ensuing reduction. For this reason, the models are kept as small as possible. The choice of 16 internal features is made based on the fact that BLAS kernels typically need at least 16 times 4 elements at single precision to utilize the CPU fully. In other words, going any lower than 16 would reduce the efficiency at which we can perform the necessary computations. For more information, we refer the reader to [95].

For the task of early reduction screening, we are only interested in the reduced graphs, i.e. the graph after applying the inexpensive reductions. There are 71 reduced graphs in total, with over 1 million labeled vertices combined. Each graph in the dataset is divided into three parts: (1) a training set containing 60 % of the

Metric	Description	Better
Accuracy	The probability that a suggested vertex can be reduced.	higher
Coverage	The fraction of reducible vertices suggested by the model.	higher
Screening	The fraction of the total vertices in the graph suggested.	lower

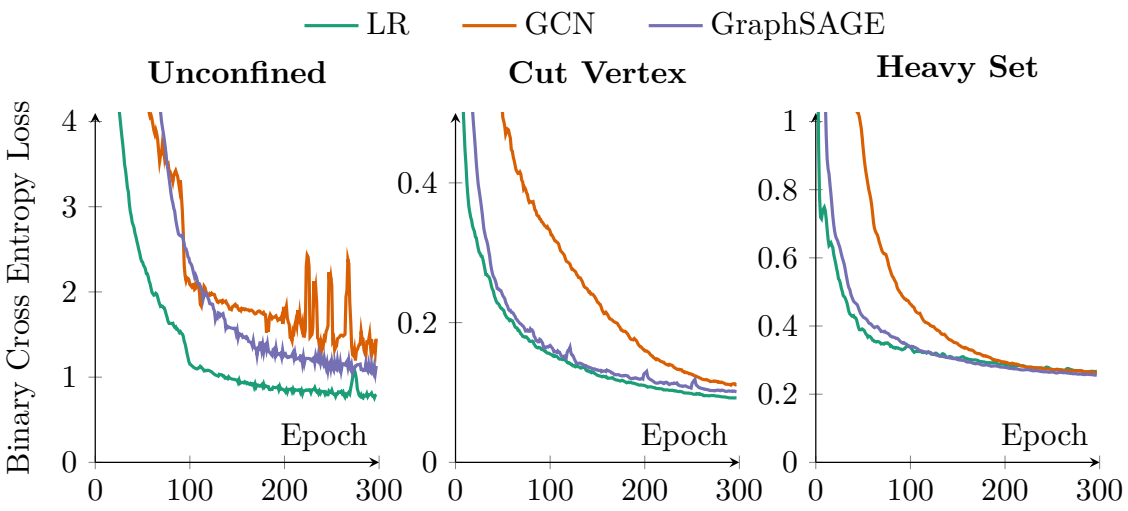
Table 4.5: Metrics used to evaluate the performance of the GNN models.

vertices, (2) a validation set containing 20 %, and (3) a test set containing the last 20 %. Vertices labeled as 2 (timeout) are not used during training or validation. Each model is trained for 300 iterations, where one iteration uses the entire dataset.

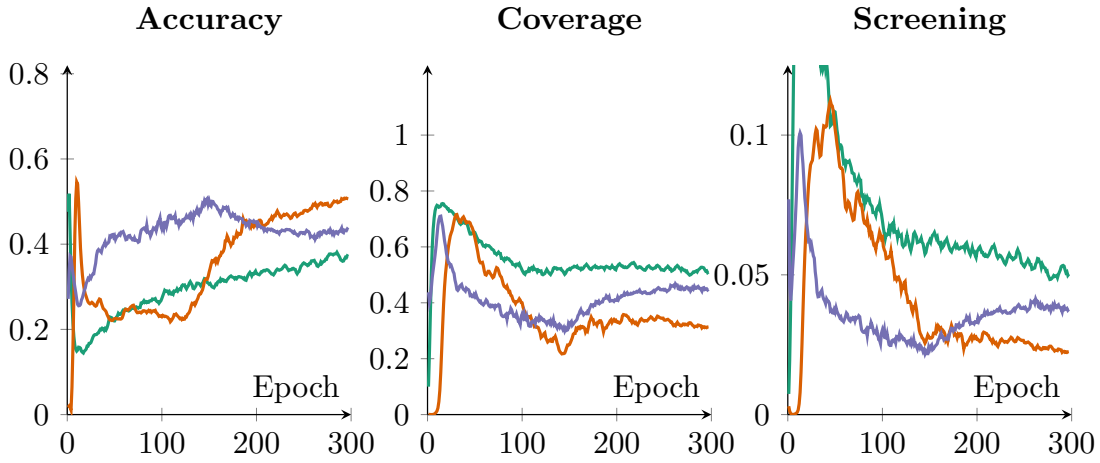
The training data can be considered a single large graph since we split the data in terms of vertices, not individual graphs. While this graph is derived from the same instances we use to evaluate LearnAndReduce later in the experiments, we argue that this is not an issue for the evaluation. First, the reduced graphs are snapshot images of how the instances looked after running a specific set of reductions. At inference, reductions are applied continuously as reducible vertices are found. This already means the inputs the GNN models will see at inference could be completely different from those seen during training. Furthermore, the reductions that lead to the reduced graphs are also not the same ones used in LearnAndReduce, and all the training results are given for the test set, which is a completely different set of vertices not seen during training.

In addition to the loss value, we also provide three additional metrics in Table 4.5 from the training procedure that indicate how well the trained model will perform at early reduction screening.

The detailed results for the Reductions Extended Unconfined Vertices, One Vertex Cut by Xiao et al. [218], and Heavy Set by Xiao et al. [219] can be seen in Figure 4.9a. The proposed LR architecture clearly outperforms the GCN and GraphSAGE for the unconfined rule, but the difference is less noticeable for the critical and heavy sets. Figure 4.9b provides accuracy, coverage, and the fraction of vertices removed by the screening for the heavy set reduction. Despite the similar-looking loss values for this reduction rule, it is clear from the other metrics that the models are discovering different things. GCN suggests the least amount of vertices. It also has the highest probability that a suggested vertex can actually be reduced. However, this is at the cost of catching less of the total reducible vertices. GraphSAGE and LR both suggest a larger fraction of the reducible vertices but at the cost of reduced suggestion accuracy and overall screening effect. Note that the models do not have any notion of accuracy or coverage during training, so reducing the loss of training data is the only thing



(a) Training loss for unconfined, critical set, and heavy set.



(b) Accuracy, coverage, and the fraction remaining after the screening for the heavy set rule.

Figure 4.9: Comparisons for all architectures on the test dataset. Note that the scaling of the y-axes differs.

being optimized for during training. To summarize the plots in Figure 4.9b for LR, it suggests approximately 5 % of the graph for the heavy set rule, and among the suggested vertices are approximately 50 % of the vertices that actually can be reduced using the heavy set rule.

The final results for all the expensive reduction rules on the test set are given in Table 4.6. Training all the models took approximately 24 hours on our test machine using PyTorch Geometric [75]. While PyTorch is a powerful framework for quickly developing and training models, several optimizations that could speed up the screening phase in our final program are not utilized. This includes combining the message passing and weighted transformation to reduce the number of times the data needs to pass through the cache hierarchy, loading feature vectors directly into AVX registers,

		GCN				GraphSAGE				LR			
Reduction	Red.	Scr.	Cov.	Acc.	Loss	Scr.	Cov.	Acc.	Loss	Scr.	Cov.	Acc.	Loss
Critical	6.19	7.03	**44.06**	38.81	0.45	**4.72**	31.73	41.63	0.44	6.52	43.99	**42.23**	**0.43**
Cut	0.53	0.03	2.89	55.91	0.11	**0.00**	0.16	24.25	0.10	0.06	**8.07**	**77.80**	**0.09**
Gen. fold	2.45	**0.14**	2.34	**42.17**	0.32	0.83	12.68	38.28	0.26	0.98	**14.33**	36.73	**0.25**
Heavy set	3.64	**2.25**	31.39	**50.65**	**0.26**	3.77	44.84	43.39	**0.26**	5.02	**51.03**	37.13	**0.26**
Heavy set 3	2.68	**0.47**	7.88	**45.45**	**0.29**	0.51	5.74	30.60	0.34	0.90	**9.35**	28.00	0.33
Unconfined	7.94	10.15	25.95	25.37	1.45	**5.65**	17.20	**27.07**	1.13	30.97	**53.36**	14.10	**0.80**

Table 4.6: Detailed results in percentage on the test data for all reduction rules used in the LearnAndReduce and GNN architectures. Red. refers to the fraction of vertices in the data that can be reduced using the reduction rule. The **best** numbers are shown in bold, where Scr., Cov., Acc., and Loss are compared individually.

and hard-coding the inner kernels for precisely the dimensions used by the models. To be clear, if the number of features had been higher, PyTorch would undoubtedly perform at a level close to the theoretical limit of the CPU. The optimizations listed above are useful because the number of features is low, and the main bottleneck is the memory bandwidth and latency. For this reason, we only continue with further experiments using one GNN architecture, and we do so using a manual implementation written in C.

Observation 4.2.1: Training Results for GNN Architectures. Even though the differences are small, LR archives the best training loss and coverage on 5/6 instances compared to GCN and GraphSAGE. Therefore, we proceeded with the LR architecture.

Experiments

For the cyclic struction in LearnAndReduce we have the two configurations *Fast* and *Strong* as introduced by Gellner et al. [91]. We run the five different screening approaches introduced in Section 4.2.3 for both of these configurations. In Table 4.7, we present the average results for all configurations. With the *no gnn red* screening, we can see that these configurations are the fastest but are computing the largest reduced instances. When we use the expensive rule without screening in *never*, we can see the potential gain in reduction size. In the *Fast* configuration, we can further reduce the instances by 13.22 % on average when using expensive reduction rules. However, this comes at the cost of more than doubling the preprocessing time on average. All of our *Fast* screening methods can reduce the preprocessing time compared to *never*.

Config	GNN	$n_K/n_{\tilde{K}}$	n_K/n_G	m_K/m_G	Offset	Time	Time exp.	$\#n_K = 0$
Fast	no gnn red	100.00%	5.50%	51.13%	13 179 456	**36.55**	0.00	49 / 83
Fast	never	86.78%	4.77%	50.85%	13 270 743	74.74	38.19	52 / 83
Fast	always	87.11%	4.79%	50.90%	13 269 029	65.51	28.96	52 / 83
Fast	initial	87.13%	4.79%	50.88%	13 268 824	58.59	22.04	52 / 83
Fast	initial tight	87.41%	4.81%	50.91%	13 267 958	45.12	8.57	51 / 83
Strong	no gnn red	98.74%	5.43%	50.81%	13 188 267	68.86	0.00	50 / 83
Strong	never	**85.06%**	**4.68%**	**50.51%**	**13 280 038**	101.56	32.70	53 / 83
Strong	always	85.56%	4.71%	50.55%	13 278 028	107.17	38.31	53 / 83
Strong	initial	85.66%	4.71%	50.56%	13 277 789	88.24	19.38	**54 / 83**
Strong	initial tight	85.65%	4.71%	50.58%	13 277 974	85.20	16.34	53 / 83

Table 4.7: Arithmetic mean reduction results for different configurations. Here, K is always the corresponding reduced instance, $\tilde{K}$ the reduced instances computed by *Fast - no gnn red* (for reference), and G the original graph. We refer to the number of nodes n and edges m of the corresponding graph in the index. Furthermore, we give the offset and reduction time. The configuration *no gnn red* does not use any of the computationally expensive reductions. In the column Time exp., we give the time used only for expensive reductions and reduction screening.

The fastest *initial tight* is on average a factor of 1.7 times faster and manages to reduce the instances by an additional 12.59 % compared to *no gnn red*. For the *Strong* configurations without screening, we can reduce the instances by 14.94 % on average when using the expensive reduction rules. However, the configuration without any expensive rules already takes more time than all the *Fast* configurations using our screening approach. Therefore, we only focused on the different *Fast* configurations in the following.

In Figure 4.10, we present two performance profiles comparing the different screening approaches for the *Fast* configuration with additionally running CHILS on the reduced instances afterward. On the left, we have the solution quality achieved using one hour per instance for preprocessing and CHILS. On the right, we see the performance profile for the time needed to find the best solution within this time limit. We can see that for all but five instances, all approaches achieve the same solution quality, and for the instances where the quality differs, the differences are minor. However, when considering the running times, we can see concrete differences. On more than 50 % of the instances, not using the expensive reduction rules and having more time for the local search is the best strategy. However, this approach can be multiple orders of magnitude slower on other instances. Not using the screening for

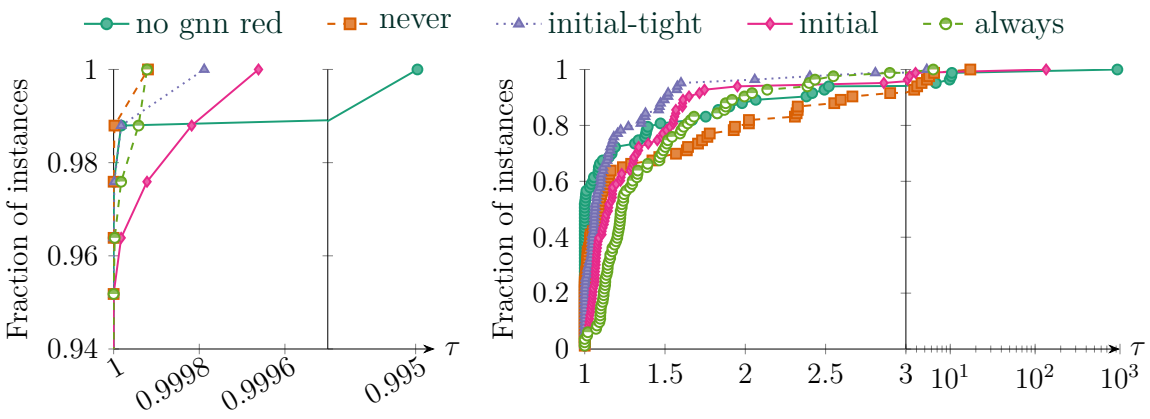


Figure 4.10: Performance profiles for different (*Fast*) LEARNANDREDUCE configurations. After reducing the instance, we run the CHILS heuristic to evaluate the practical impact of the reduction configurations. We present solution quality, including reduction offset (left) and running time including reduction time (right). The vertical line in each plot marks the change from one scale to another.

expensive rules, i. e., *never* is often more than twice as slow as other configurations. Overall, *initial tight* performed best on most instances regarding running time and is also very close to not using the screening regarding solution quality. There are only four instances where this approach took more than 1.6 times as long as the fastest solver on the respective instance. For all other configurations, this is true for more than 12 instances.

Observation 4.2.2: LearnAndReduce Configurations. The *Strong* configuration without any expensive rules already takes more time than all the *Fast* configurations using our screening approach. Therefore, we only focus on the different *Fast* configurations. The *initial tight* performs best overall, especially for more difficult instances. Regarding solution quality, the *initial tight* approach is very close to not using the screening. Therefore, we use the *Fast* - *initial tight* configuration for our LEARNANDREDUCE routine.

4.3 Memetic Algorithm m²wis

Memetic algorithms, as discussed in Section 2.2.4 combine evolutionary algorithms with other heuristics, such as local search to effectively explore (via global search) and exploit (via local search) the solution space. The general idea behind these algorithms is to use mechanisms inspired by biological evolution, such as selection, mutation, recombination, and survival of the fittest.

In this section, we develop an advanced memetic algorithm M²WIS to tackle the MWIS problem. We incorporate exact and heuristic reduction techniques to compute near-optimal weight-independent sets in huge, sparse graphs. More precisely, in addition to using exact data reductions, we use a memetic approach to heuristically choose vertices to include in the solution and further reduce the instances. This strategy is likely to open up the reduction space and remarkably speed up the computation of large-weight independent sets. Our experiments show that this approach is able to compete and outperform state-of-the-art algorithms. Our two configurations compute the best results among all competing algorithms for all instances tested. Therefore, this approach is a useful tool when large-weight independent sets in practice.

References. *This section is based on joined work with Sebastian Lamm, Christian Schulz and Darren Strash [102], and the extended journal paper, which is also joined work with Sebastian Lamm, Christian Schulz, and Darren Strash [103]. Large parts of this section are copied verbatim from these papers.*

Our Results. Our contribution is two-fold: First, we develop a state-of-the-art memetic algorithm based on recombination operations employing graph partitioning techniques. Our algorithm computes large-weight independent sets by incorporating a wide range of recently developed advanced reduction rules.

The algorithm may be viewed as performing two functions simultaneously: (1) reduction rules for the Maximum Weight Independent Set problem are used to boost the performance of the memetic algorithm *and* (2) the memetic approach opens up the opportunity for further reductions by selecting vertices that are likely to be in large-weight independent sets. In short, our method applies reduction rules to form a reduced instance, then computes vertices to insert into the final solution and removes these vertices and their neighbors from the graph. Then, further reductions can be applied. This process is repeated recursively until the graph is empty. This technique finds near-optimal weight independent sets much faster than existing local search algorithms. It is competitive with state-of-the-art exact algorithms for smaller graphs and allows us to compute large-weight independent sets on huge, sparse graphs. Overall, our algorithm configurations compute the best results among all competing algorithms for every instance and thus can be seen as the dominating tool when large weight independent sets need to be computed in practice.

4.3.1 The Algorithm Description

We now present our memetic algorithm for the MWIS problem, which we call MEMETIC MAXIMUM WEIGHT INDEPENDENT SET (M²WIS). This algorithm is inspired by REDUMIS [148] and is executed in rounds. Each round is split up into three parts. The exact data reductions are applied at the beginning of each round (EXACTREDUCE). Here, the graph is reduced as much as possible using a wide range of data reduction rules. On the resulting reduced graph, we apply the memetic part of the algorithm as the second step. We represent a solution, also called an individual, using bitvectors. The independent set $\mathcal{I}$ is represented as an array $s \in \{0,1\}^n$. For each array entry, it holds $s[v] = 1$ if and only if $v \in \mathcal{I}$. The memetic component itself is also round-based. Starting with an initial population $\mathcal{P}$, consisting of a set of individuals, this population evolves over several rounds until a stopping criterion is fulfilled. In the third part of each round, we select a subset of vertices to be included in the independent set by considering the resulting population. Here, we implement different strategies to select vertices for inclusion. Including these vertices in the independent set enables us to remove them and their neighbors from the instance. This opens up the reduction space, i. e., further reductions might be applicable after the removal process. The steps of exact reduction, memetic search, and heuristic reduction are repeated until the remaining graph is empty or another stopping criterion is fulfilled.

In Algorithm 11, we first apply the EXACTREDUCE routine described in Section 2.2.1. For the exact reductions, we use Reduction List 4.1. This is the initial reduction list, which performed overall the best in the ordering experiments presented in Section 4.1.3.

In the following, we describe the memetic routine EVOLVE, followed by the different vertex selection strategies used to heuristically open up the reduction space.

If, at some point, the resulting reduced instance is small enough, i. e., its number of vertices is less than a threshold n_K, we try to solve the instance exactly using STRUCTION by Gellner et al. [91] with the configuration CYCLICFAST and a time limit t_{exact}. If the instance is not solved optimally within this time, the computed solution is added to the population, and we continue.

The Memetic Components

After EXACTREDUCE, we apply the EVOLVE routine, which is described in Algorithm 12 on the reduced graph $\mathcal{K}$. It starts by generating the initial population of size $|\mathcal{P}|$, which is then evolved over several generational cycles (rounds). For the evolution of the population, two *individuals* from the population are selected and com-

Algorithm 11: M²WIS(G): High level overview.

```
Data: graph G = (V, E), reduction list R
Result: Best found Independent Set W
Procedure m²wis(G):
    W = ∅                                              // Best solution
    while not time limit exceeded do
        K, α ← ExactReduce(G, R)
        if K is empty then
            W ← Restore(K, α, W)
            return W
        if V(G) ≤ n_K then
            W* ← TryToSolveExact(K, t_exact)
            if W* optimal then
                return Restore(K, α, W*)
        P ← CreateInitialPopulation(K)                 // Adding W* if
         computed
        P ← Evolve(G, P)
        K̃, W ← HeuristicReduce(K, P, W)
        G ← K̃
    return W
```

bined to create an *offspring*. We also apply a mutation operation to this new solution by forcing new vertices into the solution and removing neighboring solution vertices. We look for good replacements to keep the population size constant and still add new offspring to the solution. In this process, we search for individuals in the population which have smaller weights than the new offspring. Among those, we look for the most similar solution by computing the intersection size of the new and existing individuals. We also add the possibility of forcing individuals into the population if it has not changed over a certain number of iterations and rejecting the offspring if the solution with the smallest weight is still better than the new offspring. Note that the size of the population $|\mathcal{P}|$ does not change during this process. Additionally, at any time, each individual in our population is an independent set. In the last step of the memetic algorithm, we improve the solution by the HILS algorithm [172]. The stopping criterion for the memetic procedure is either a specified number of unsuccessful combine operations or a time limit. In the following, we discuss each of these steps in detail.

Algorithm 12: EVOLVE$(G, \mathcal{P})$: High level overview.

```
Data: graph G = (V, E), current population P
Result: evolved population P
Procedure evolve(G, P):
    while not stopping criterion holds do
        randomly chose combine operation COMBINE
        IS ← ∅                    // Parent individuals
        OS ← ∅                    // Set of offspring
        IS ← TOURNAMENTSELECT(P)
        OS ← COMBINE(IS)
        if mutate with probability 10 % then
            OS ← MUTATE(OS)
        if suitable replacement then
            P ← REPLACE(P, OS)
    return P
```

We start by introducing the computation of the initial solution and then explain the combined operations for the evolutionary process as well as the mutation operation.

Initial Solutions. At the start of our memetic algorithm, we create an initial population of size $|\mathcal{P}|$. To diversify as much as possible, this population contains solutions computed in six different ways, which we choose uniformly at random. Before applying the strategies, we permute the order of the vertices such that different solutions are obtained for the same strategy by different tie-breaking.

RandomMWIS. The first approach works by starting with an empty solution and adding free vertices uniformly at random until the solution is maximal.

GreedyWeightMWIS. For the GREEDYDEGREEMWIS strategie, we start with an empty solution. This is extended to a maximal independent set by adding free vertices ordered by their weight. Starting with the largest weight, we include this vertex and exclude all its neighbors until all vertices are labeled either included or excluded.

GreedyDegreeMWIS. Via this greedy approach, we create initial solutions by successively choosing the next free vertex with the smallest residual degree. Each time a vertex is included, we label the neighboring vertices to be excluded.

GreedyWeightVC. In contrast to the previous approaches, the vertex cover problem is solved here. Therefore, an empty solution is extended by vertices of the smallest weight until a vertex cover is computed. As soon as the approach terminates, we compute the complement and have an initial solution to the MWIS problem.

GreedyDegreeVC. As in GreedyWeightVC, the complementary vertex cover problem is solved. However, for this approach, we choose those vertices to include in the solution, which cover the maximum number of currently uncovered edges.

Struction. We also add the possibility to compute an initial solution via the Struction algorithm by Gellner et al. [91]. We set a time limit of 60 seconds and use the configuration CyclicFast. If we also use Struction to compute initial solutions, we call the algorithm configuration M²WIS+S. Note that if the algorithm does not solve the instance within the time limit, it returns a non-optimal solution.

Combine Operations. The common idea of our combine operations, which are inspired by the work of Lamm et al. [148], is to combine whole blocks of independent set vertices. We use the graph partitioning framework KaHIP [187] to construct those blocks. To explain the combine operations, we first introduce the concepts of *partitioning* and *edge separators* used in the following.

The subdivision of the set of vertices V into disjoint blocks $V_1, ..., V_k$ such that $V_1 \cup ... \cup V_k = V$ is called a *k-way partition* [192, 41]. To ensure the blocks are roughly of the same size, the balancing constraint $|V_i| \leq L_{max} := (1 + \varepsilon) \left\lceil \frac{|V|}{k} \right\rceil$ with the imbalance parameter $\varepsilon > 0$ is introduced. While satisfying this balance constraint, the *edge separator* problem asks for minimizing the total cut, $\sum_{i<j} \omega(E_{ij})$, where E_{ij} is defined by $E_{ij} := \{\{u, v\} \in E : u \in V_i, v \in V_j\}$. The edge separator is the set of all edges in the cut. For the k-vertex separator problem, on the other hand, we look for a division of V into $k + 1$ blocks. In addition to the blocks $V_1, ..., V_k$ a separator S exists. This separator has to be chosen such that no edges between the blocks $V_1, ..., V_k$ exist, but there is no balancing constraint on the separator S. However, as for the edge separator problem, the balancing constraint on the blocks $|V_i| \leq L_{max} := (1 + \varepsilon) \left\lceil \frac{|V|}{k} \right\rceil$ has to hold. To solve the problem, the size of the separator $|S|$ has to be minimized. Removing the separator S from the graph results in at least k connected components since the different blocks V_i are not necessarily connected. Using KaHIP, we partition the graph into k blocks $V_1 \cup ... \cup V_k = V$. For $j = 1, ..., k$ the solution blocks $\mathcal{I}_j$ are defined by $\mathcal{I}_j = \mathcal{I} \cap V_j$. We created different offspring using the following combined operations on those solution blocks.

The parents for the first two combine operations are chosen by two runs of the tournament selection [166], where the fittest individual i. e., the solution with the highest weight gets selected out of two random individuals from the population. Then we perform one of the combine operations outlined below, and finally, after the combine operation, we use HILS [172] to improve the computed offspring.

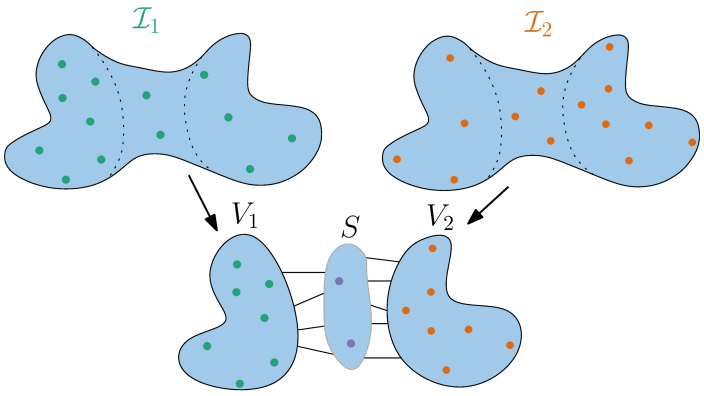


Figure 4.11: The vertex separator combine operation to create an offspring $\mathcal{O}$ out of two individuals $\mathcal{I}_1$ and $\mathcal{I}_2$.

Vertex Separator Combination. The first operator works with a vertex separator $V = V_1 \cup V_2 \cup S$. We use a vertex separator to exchange whole blocks of solutions without violating the independent set property. This can be done because no vertices belonging to different blocks are adjacent. Neighboring vertices would either be part of the same block, or one of them has to belong to the separator S. By this property, the combination of those blocks will always result in a valid solution to the independent set problem. The two individuals selected by the tournament $\mathcal{I}_1$ and $\mathcal{I}_2$ are split up according to these partitions and are then combined to generate two offspring $O_1 = (V_1 \cap \mathcal{I}_1) \cup (V_2 \cap \mathcal{I}_2)$ and $O_2 = (V_1 \cap \mathcal{I}_2) \cup (V_2 \cap \mathcal{I}_1)$. After that, we add as many free vertices greedily by weight until the solution is maximal. We get a local optimum via one iteration of the weighted local search. See Figure 4.11 for an illustration.

Multi-way Vertex Separator Combination. We extended the previously described operator to the multi-way vertex separator, where multiple solutions can be used and combined. Therefore, we compute a k-vertex separator $V = V_1 \cup ... \cup V_k \cup S$ and select k individuals. Then, a score is computed for every pair of partition V_i and individual $\mathcal{I}_j$ for $i, j \in \{1, ..., k\}$. This score is defined by $\sum_{v \in V_i \cap I_j} \omega(v)$. We start with the pair resulting in the highest score pair and then select pairs decreasingly. Once an individual or partition block is selected, we do not use it again. In contrast to the

previous operator, this combination only results in one offspring. We then maximize this offspring and compute a local maximum.

Edge Separator Combination. For this operator, we exploit the duality to the weighted vertex cover problem. Starting with a partition $V = V_1 \cup V_2$, the operator computes temporary offspring for the weighted vertex cover problem. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be the individuals selected by the tournament rule. Let $C_i = V \setminus \mathcal{I}_i$ be the solution to the weighted vertex cover problem for $i \in \{1, 2\}$. The new offspring are $O_1 = (V_1 \cap C_1) \cup (V_2 \cap C_2)$ and $O_2 = (V_1 \cap C_2) \cup (V_2 \cap C_1)$. However, these offspring can contain some non-covered edges, which are a subset of the cut edges between the two partitions. The graph induced by the non-covered cut edges is bipartite. In this graph, we compute a minimum-weight vertex cover using maximum flows.

Multi-way Edge Separator Combination. Similar to the vertex separator, the edge separator can also be extended to use multiple solutions. Therefore, a k-way-partition $V = V_1 \cup ... \cup V_k$ is computed. Equivalent to the multi-way vertex separator, we also select k individuals and compute a score for each pair V_j and $\mathcal{I}_i$. For the scoring function, the complement of an independent set inside the given block is used to sum up the weights of the vertices of the vertex cover in this block. For the offspring computation, each block is combined with the individual with the lowest score. As in the basic edge separator combine operator, there can be edges in the cut that are not covered. Since the induced graph here is not bipartite, we handle this problem using a simple greedy strategy. Afterward, the solution is transformed to get the offspring for the independent set individuals.

Mutation Operation. After each combine operation, a mutation operator can perturb the created offspring. This is done by forcing new vertices into the solution and removing the adjacent vertices to satisfy the independent set property. Those vertices are selected at random among all non-solution nodes in the graph. Afterward, we improve the solution using the HILS algorithm.

Heuristic Reductions and Recursion

After the memetic algorithm stops, we use a heuristic data reduction to open up the reduction space (and afterward, the next round of exact data reductions begins). We implemented different strategies to select vertices that we put into the solution. In each strategy, vertices are ordered by a rating function. The algorithm inserts

a fraction (from only one vertex to 100 %) of the vertices selected by the different strategies. We now explain the different selection strategies.

Vertex Selection by Weight. The first rating function is based on the weight $\omega(v)$ of a vertex v (higher is better). The intuition here is that by adding a vertex, we want to increase the weight of our solution as much as possible. More precisely, the fittest individual from the population evolved by the memetic algorithm is selected. The fitness of an individual is defined as the solution weight. From this individual, we select the x vertices from the independent set that have the highest weight and add them to our solution. Since we only consider vertices from one individual, x can be freely chosen without violating the independent set property of our solution. For example, we can only add the highest-weight vertex or select a fraction of those solution vertices to add.

Vertex Selection by Degree. Similar to the previous vertex selection strategy, we choose the fittest individual from which we add vertices to our solution. Here, the vertices are rated by their degree $\deg(v)$ (smaller is better). The intuition here is that adding vertices with a small degree to our solution will not remove too many other vertices from the graph that could be considered later.

Vertex Selection by Weight/Degree. For this selection strategy, we rate the vertices v of the fittest individual by the fraction $\frac{\omega(v)}{\deg(v)}$ (higher is better). This way, we combine both of the two previous ratings.

Hybrid Vertex Selection. In the hybrid case, the solution vertices $v \in V$ are rated by the weight difference between a vertex and its neighbors $\omega(v) - \sum_{u \in N(v)} \omega(u)$ (higher is better). This value describes the minimum gain in solution weight we can achieve by adding the vertex v to the solution. Note that Gu et al. [107] proposed this rule for their algorithm. The key difference here is that Gu et al. [107] use this function on all vertices, while our algorithm only considers solution vertices of the fittest solution of the memetic algorithm.

Vertex Selection by Solution Participation. In contrast to the previous methods, this strategy considers the *whole* population. Moreover, here, we consider *each vertex* in the graph. We check the population and assign each vertex a value according to the number of times it is part of a solution. Therefore, the maximum number a vertex can achieve is bounded by the population size $|\mathcal{P}|$. We can include all vertices

that are in every individual, i. e., vertices with a score equal to $|\mathcal{P}|$, as well as exclude all vertices that are in no solution at all, i. e., vertices with a zero score. Note that the total number of vertices selected with this strategy differs from the previous strategies, where we consider all the vertices from the best solution in the population. The vertices selected by solution participation are only a subset of these. This process is similar to the Merge Search introduced by Kenny et al. [135].

4.3.2 Experimental Evaluation

This section presents experiments analyzing different parameter values that highly influence the performance of our algorithm. Then, we present a detailed state-of-the-art comparison on a large set of instances.

Methodology. We implemented our algorithm using C++11. The code is compiled using g++ version 12.2 and full optimizations turned on (-O3). For the experiments, we use Machine 1. We run each configuration with four different seeds and a time limit of ten hours. We always report the geometric mean results if not stated otherwise. For all algorithms, we always report when the best solution is found within this time limit. The algorithms that might not use the whole ten-hour time are the exact algorithm KaMIS BnR, which terminates when found and proven an optimal solution, and the heuristic HtWIS. Note that this heuristic does not use any randomness, which would enable us to run it multiple times with different seeds for using the whole 10 hours, nor does the algorithm have any other parameters that would increase the solution quality by spending more time. If a solver exceeds a memory threshold of 100 GB during the time limit of ten hours for an instance, we mark this with a dash. In general, our algorithm does not test the time limit in the ExactReduce routine of M²WIS or during the calculation of the separator and partition pool. Hence, the time limit can be exceeded if the ten-hour mark is reached during these steps.

Parameter Configuration. We used the fast configuration of the KaHIP graph partitioning package [188, 189] for the computation of the graph partitions and vertex separators. In the ExactReduce we use the Reduction List 4.1 which is one of the best-performing orderings of these reductions, as shown in Section 4.1.3. Similar to Lamm et al. [148], we set the population size $|\mathcal{P}|$ to 250, the size of the partition and separator pool to 10 and the mutation rate to 10 %. Local search is limited to 15,000 iterations. Finally, for the multi-way combine operations, we bound the number of blocks used by 64. For the state-of-the-art experiments, we set the parameters as discussed in the following.

Data Sets. For the main experiments in this section, we use the Instance Set 1, also used in Section 4.1.3. It contains 207 instances from the Stanford Large Network Dataset Repository (`snap`) [155], OpenStreetMaps (`osm`) [1, 20, 37], the SuiteSparse Matrix Collection (`ssmc`) [53] as well as dual graphs of well-known triangle meshes (`mesh`) [185], and 3d meshes derived from simulations using the finite element method (`fe`) [210]. To compare with METAMIS, we use the Instance Set 3.

Instance Set 3 (Reduced OSM)**.** *This set consists of 17 OpenStreetMap [1, 20, 37] instances initially reduced using KaMIS BnR, as used by Dong et al. [59]. This set is called* `osmRed`*, and detailed graph properties are given in Table A.4 in the Appendix.*

We do not compare this approach on the Instance Set 4 of vehicle routing instances since data reductions do not work well on those instances [60] and the reduced graphs are still too large to explore sufficiently by our memetic algorithm.

Heuristic Data Reduction Rules

For the heuristic reduction, we perform multiple parameter tuning experiments on a subset of our dataset containing 15 graphs, marked in Table A.1 in the Appendix. For this, we took three large graphs from each class to evaluate the influence of our parameters on the performance. For each instance, we used four different seeds and a time limit of one hour.

We start by comparing the different vertex selection strategies presented in Section 4.3.1. Each strategy is evaluated for different fractions. Table 4.8 summarizes our results. For each configuration, we show the geometric mean quality and the geometric mean running time over the subset of 15 graphs. Note that the number of selectable vertices is different between *solution participation* and the other strategies. For *solution participation*, where vertices are only considered if they are in each or none solution within the *whole* population, the set of vertices selected is only a subset of the vertices selectable by the other strategies. Adding 100 % of the vertices does result in the same solution for these strategies since they are all based on the best solution, which is then always included completely. Furthermore, this explains the larger speedup for these other strategies compared to *solution participation* when the fraction is increased.

For all strategies, we can reduce the mean time by increasing the fraction of vertices added. The smallest speedup is observed for *solution participation*. This is due to the fact that, the total number of vertices that can be added (100 %) is less than for all the other strategies. With these other strategies, we can achieve a speedup of up to 10 by increasing the fraction parameter. However, the quality using *solution*

	t	ω	t	ω	t	ω	t	ω	t	ω
f	degree		hybrid		solution participation		weight		weight/degree	
1	781.48	4 052 070	753.95	4 052 039	654.53	4 054 053	785.94	4 052 026	610.46	4 052 120
5%	405.97	4 052 805	549.90	4 053 186	798.08	4 052 516	362.88	4 051 708	345.88	4 052 895
25%	208.75	4 052 056	252.43	4 052 409	685.24	4 053 765	135.34	4 051 263	199.10	4 051 819
75%	69.95	4 050 596	100.64	4 050 646	558.72	4 054 196	68.99	4 050 476	69.75	4 050 397
100%	59.60	4 050 453	60.17	4 050 453	515.11	**4 054 303**	59.82	4 050 453	60.41	4 050 453

Table 4.8: M²WIS geometric mean solution weight ω and time t (in seconds) required to compute ω for different vertex selection strategies and fractions f. Setting $f = 1$ is adding one vertex, while otherwise, f refers to the percentage of vertices that are added from all possible vertices for the corresponding strategy. The **best** result among all configurations is marked bold.

participation is increasing with increasing fractions until $f = 100\,\%$, while the other strategies perform best with $f = 5\,\%$. When further increasing the amount of vertices added in these strategies, the solution quality gets worse. The difference between the best and the worst mean result is around 0.1 %.

The overall best mean solution quality is achieved by using the *solution participation* with adding 100 % of the vertices possible. We fix this configuration for the next set of our experiments.

Observation 4.3.1: Heuristic Data Reduction. The strategy based on *solution participation* and fraction of 100 % performed best regarding solution quality. The fastest variants are all other strategies with a fraction of 100, %, which all only perform one round of the algorithm and then return the currently best individual.

Solving Small Reduced Graphs Exactly

With this experiment, we examine whether it is beneficial at some point to solve the reduced instance exactly. Therefore, we introduced two new parameters. First, we add a threshold n_K to determine when to start solving the reduced instance optimally. When the number of vertices in the reduced graph K is smaller than n_K, we apply STRUCTION. The second parameter is a time limit t_{exact}, which restricts this exact solver. If the instance is not solved within this time limit, we continue with M²WIS.

We present the summary of the results using different n_K from 0 to 15 000 and t_{exact} from 10 seconds to no time limit for M²WIS in Table 4.9 and for M²WIS+S in Table 4.10. We report the time when the best solution was found. Often, the exact solver finds a good solution very fast but is then not able to improve the solution. This is why, when increasing n_K, we sometimes see a decrease in the reported time.

In both tables, the solution quality is similar between the different configurations. These differences are all less than 0.01 %. However, we see in Table 4.9 that for M²WIS $n_K = 15\,000$, the mean time is reduced up to a factor of 15 for all time limits t_{exact} tested. The threshold $n_K = 10\,000$ yields the best result with a time limit of $t_{exact} = 100$. For M²WIS+S in Table 4.10 we see that the best parameter configuration is $n_K = 100$ and t_{exact} =1 000 seconds. With this configuration, we also achieved the overall best result compared to M²WIS.

Observation 4.3.2: Solving Small Reduced Instances. For the state-of-the-art comparison, we mainly focus on solution quality. For M²WIS, we use the configuration of parameters $n_K = 15\,000$ and $t_{exact} = 100$ seconds, since compared to the parameters yielding the best solution quality this configuration is only 0.000 02 % worse regarding solution quality, however, it is 2.5 times faster. For M²WIS+S we choose the configuration yielding the best solution quality is $n_K = 100$ and $t_{exact} = 1\,000$ seconds.

Limiting the Time for Evolve

Additionally to limiting the number of rounds of the evolution cycles, we also restrict the time used within the EVOLVE procedure. This is especially important for very complex instances where EVOLVE can take up all the time. We test different limits for *evotime* starting with 450 seconds up to 4 hours. We present the geometric mean over the results for both M²WIS and M²WIS+S in Table 4.11. Generally, we see that this parameter has a higher impact on the solution quality compared to the previous parameters tested.

	t	ω	t	ω	t	ω	t	ω
n_K	$t_{exact} = 10$		$t_{exact} = 100$		$t_{exact} = 1\,000$		no limit	
0	515.11	4 054 303	515.11	4 054 303	515.11	4 054 303	515.11	4 054 303
100	519.69	4 054 164	513.77	4 054 313	535.64	4 054 080	533.28	4 054 208
500	509.72	4 054 191	525.53	4 054 337	517.47	4 054 132	532.67	4 054 066
1 000	480.82	4 054 037	501.77	4 054 305	522.39	4 054 213	505.01	4 054 234
5 000	188.45	4 054 285	196.32	4 054 114	205.92	4 053 974	204.50	4 054 138
10 000	79.43	4 054 144	85.13	**4 054 367**	99.65	4 054 057	114.17	4 054 068
15 000	34.22	4 054 249	37.33	4 054 180	45.45	4 054 229	48.74	4 054 052

Table 4.9: M²WIS geometric mean solution weight ω and time t (in seconds) required to compute ω for different thresholds n_K and time limits t_{exact} to solve reduced instances exactly. The **best** result among all configurations are marked bold.

	t	ω	t	ω	t	ω	t	ω
n_K	$t_{exact} = 10$		$t_{exact} = 100$		$t_{exact} = 1\,000$		no limit	
0	15.58	4 054 480	15.58	4 054 480	15.58	4 054 480	15.58	4 054 480
100	15.37	4 054 599	15.70	4 054 336	15.55	**4 054 666**	15.90	4 054 579
500	15.31	4 054 530	15.90	4 054 537	15.52	4 054 546	15.60	4 054 439
1 000	15.52	4 054 467	15.52	4 054 520	15.48	4 054 386	15.34	4 054 445
5 000	13.21	4 054 461	12.92	4 054 571	14.18	4 054 628	15.69	4 054 481
10 000	13.23	4 054 538	14.39	4 054 409	18.25	4 054 463	20.90	4 054 377
15 000	13.04	4 054 569	14.04	4 054 492	17.85	4 054 543	20.58	4 054 520

Table 4.10: M²WIS+S geometric mean solution weight ω and time t (in seconds) required to compute ω for different thresholds n_K and time limits t_{exact} to solve reduced instances exactly. The **best** result among all configurations is marked bold.

	t	ω	t	ω
evotime	M²WIS+S		M²WIS	
450	15.36	4 055 174	37.55	4 055 028
900	15.85	**4 055 342**	46.90	4 055 206
1 800	17.59	4 055 150	59.16	**4 055 253**
3 600	18.11	4 054 872	66.08	4 055 179
7 200	18.22	4 054 202	75.86	4 054 664
14 400	18.70	4 053 447	81.22	4 053 511

Table 4.11: Geometric mean solution weight ω and time t (in seconds) required to compute ω for different time limits *evotime* (in seconds) for the EVOLVE procedure, see Algorithm 12. The **best** result among all configurations is marked bold.

Observation 4.3.3: Evolve Time Limit. For M²WIS+S setting *evotime* = 900 performed best regarding the geometric mean solution quality on our test set, while a higher parameter value of *evotime* = 1 800 worked best for M²WIS. We choose these configurations for the state-of-the-art experiments.

Comparison against the State of the Art

We now compare our algorithm M²WIS against a range of state-of-the-art algorithms. This includes all algorithms introduced in Section 3.2.3. These are the reduce-and-peel approach HTWIS by Gu et al. [107], both STRUCTION configurations by Gellner et al. [91] where we always report the better of the two results in the column named struction, the branch-and-reduce solver KAMIS BNR by Lamm et al. [149], and HILS by Nogueira et al. [172]. Additionally, we test against the vertex cover algorithms NUMWVC by Li et al. [156] and GNN-VC by Langedal et al. [153]. We

used the provided GNN-VC model, which was trained on instances from the SuiteSparse Matrix Collection [53]. We also include the variant of our algorithm, called M²WIS+S, using STRUCTION from Gellner et al. [91] with a time limit of 60 seconds to compute individuals for the initial population. We present a representative sample of our full experiments in Table 4.12. In the last part, we give a summary over all instances. This consists of the number of instances solved best, the geometric mean time and solution quality, respectively. Detailed per-instance results are presented in Table B.2 in the Appendix.

Overall, we see that M²WIS+S has the largest number of best solutions on Instance Set 1 containing 207 graphs. In particular, M²WIS+S is able to compute the best solution for all but three instances, i. e., kentucky-3, hawaii-3 (osm) and soc-p.rel. (snap). In these three cases M²WIS+S was outperformed by our other variant M²WIS. Additionally, M²WIS+S computes the best solutions for all graphs in the graph classes finite elemente, mesh and ssmc. Finally, for all but 18 of these 207 instances, M²WIS+S finds the best solution in less than 100 seconds. Our algorithm M²WIS is still able to compute the best solution for 198 instances, including hawaii-3, kentucky-3 and soc-p.rel.

When looking at the running times, we see that HTWIS achieves the smallest geometric mean running time. However, the quality is less or equal to the results of M²WIS and M²WIS+S on *all* the tested instances, with multiple instances having a significant difference in weight–larger than 10 000. If running time is crucial, the competitor HTWIS is noteworthy. However, our algorithm also computes high-quality initial solutions which are then enhanced over time. M²WIS+S for example needs a geometric mean time of 0.004 6 seconds to compute initial solutions which are on average 0.3 % better than the solutions computed by HTWIS which needs a geometric mean time of 0.039 seconds. The best improvement already after the initial computations is found for greenland-AM3, where we need 3 times as long as HTWIS while getting an improvement of more than 12 % over the result of HTWIS. The running time achieved by STRUCTION is also to be noted. It is, for example, the second fastest on ssmc instances, see tables B.2 and B.1. STRUCTION solves 47 instances faster than all competitors. However, it only finds 188 best solutions overall. The solver KAMIS BNR is able to compute 175 best solutions within the limitations of the experiment. It works especially well on snap and mesh instances, while it is not able to solve any of the ssmc instances optimally. When looking at the performance of the four competing heuristic algorithms, we see that HILS computes overall the highest number of best solutions, followed by GNN-VC. When comparing HILS and GNN-VC directly, the comparison is highly dependent on the graph class. On the mesh and snap instances for example, GNN-VC beats HILS on every instance regard-

ing solution quality. But since we have 148 `osm` instances, where GNN-VC was not trained on, overall HILS has the larger number of best-solved instances. Regarding running time, HILS needs almost 6 times as long as GNN-VC. The overall fastest competitor HtWIS computed on 97 instances the best solution, while NuMWVC never computed a best solution within our experimental setup.

In terms of memory requirement, when the Struction is able to solve the instances very fast, memory usage is usually below 1 GB and also a bit smaller than the memory required by our algorithm. For more difficult instances, as for example `body` (`finite elemente`) where M²WIS+S and M²WIS have a memory usage of less than 1 GB, Struction requires more than 100 GB. This high memory consumption compared to our solver can be explained since, particularly for challenging instances, Struction reaches a memory-intensive branching phase.

Comparison against the State of the Art on Reduced Instances

We now compare the same algorithms on the reduced instances computed with the set of reduction rules presented in Section 4.1. In this experiment, we show the impact of our algorithm apart from the reduction rules, which can be used as a preprocessing step in general. Table B.3 shows the detailed per-instance results. In Table 4.13, we show the same sample of these results. Especially for the branch and reduce algorithm by Lamm et al. [149] we see improvements over the results in Table 4.12. With additionally using the reductions, we are able to solve more of the `ssmc` instances optimally, as well as reduce memory consumption. For example, massachusetts-3 can now be solved without reaching the memory threshold. The other algorithms also benefit from using the reductions, which can be seen in higher solution qualities and less running time. Still, our two algorithm variants also perform best on the set of reduced instances. There is only one reduced snap instance (soc-pokec-relationships), see Table B.3, where our algorithms yield a smaller solution. On this instance, they are outperformed by HILS, but computed the second and third-best solution.

	Instance	KaMIS BnR		HtWIS		HILS		Struction		m²wis+s		m²wis	
		t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
fin. el.	body	-	-	0.04	1 645 650	1 259.67	1 678 510	-	-	112.85	**1 680 182**	901.26	1 680 179
	ocean	4.88	**7 248 581**	0.07	6 803 672	11 142.43	7 075 329	-	-	5.20	**7 248 581**	5.01	**7 248 581**
	pwt	-	-	0.03	1 153 600	761.52	1 175 437	-	-	3 846.33	**1 178 734**	5 131.59	1 178 583
mesh	blob	0.14	**855 547**	0.01	854 484	260.10	854 803	0.02	**855 547**	0.04	**855 547**	0.03	**855 547**
	dragon	2.90	**7 956 530**	0.04	7 950 526	9 026.60	7 947 535	0.17	**7 956 530**	0.31	**7 956 530**	0.33	**7 956 530**
	ecat	9.18	**36 650 298**	0.50	36 606 394	36 000.05	36 562 652	1.92	**36 650 298**	2.83	**36 650 298**	3.01	**36 650 298**
osm	florida-3	1 724.45	**237 333**	0.13	234 218	216.24	**237 333**	1.33	**237 333**	11.20	**237 333**	9.57	**237 333**
	massachusetts-3	-	-	1.77	144 381	355.93	**145 866**	-	-	77.14	**145 866**	104.87	**145 866**
	utah-3	239.50	**98 847**	0.04	97 754	72.21	**98 847**	0.08	**98 847**	2.34	**98 847**	1.94	**98 847**
snap	as-skitter	-	-	1.04	124 141 373	36 000.25	123 994 141	-	-	3 422.79	**124 157 729**	621.62	**124 157 729**
	ca-GrQc	<0.01	**287 919**	<0.01	287 850	144.49	**287 919**	<0.01	**287 919**	<0.01	**287 919**	<0.01	**287 919**
	web-BS.	36 000.12	43 891 206	9.94	43 889 843	36 000.10	43 888 267	6.52	**43 907 482**	8.75	**43 907 482**	9.32	**43 907 482**
ssmc	ga2010	36 000.10	4 644 324	0.47	4 639 891	29 522.41	4 642 807	0.62	**4 644 417**	1.64	**4 644 417**	1.85	**4 644 417**
	nh2010	36 000.00	581 637	0.03	587 059	2 163.80	588 797	0.11	**588 996**	0.47	**588 996**	0.47	**588 996**
	ri2010	36 000.00	447 427	0.02	457 108	782.49	458 489	0.09	**459 275**	0.62	**459 275**	0.49	**459 275**

overall	KaMIS BnR	HtWIS	HILS	Struction	m²wis+s	m²wis
# best	175/207	97/207	154/207	188/207	204/207	198/207
gmean ω	-	153 434	154 355	-	154 430	154 430
gmean t	-	<0.01	59.18	-	0.03	0.03

Table 4.12: Average solution weight ω and time t (in seconds) required to compute ω for a representative sample of our instances comparing the best-performing algorithms. The **best** solutions among all algorithms are marked in bold. Rows are colored gray if KaMIS BnR or Struction computed an optimal solution. See tables B.2 and B.1 for the results on the full set.

	Reduced	KaMIS BnR		HtWIS		HILS		Struction		m²wis+s		m²wis	
	Instance	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
fin.el.	body	-	-	0.01	222 320	74.81	224 550	-	-	60.30	**224 744**	100.23	**224 744**
	pwt	-	-	0.07	1 019 262	655.57	1 034 418	-	-	1 391.40	**1 038 139**	4 767.83	1 038 046
mesh	ecat	0.02	**31 254**	<0.01	**30 789**	8.17	**31 254**	<0.01	**31 254**	0.03	**31 254**	0.03	**31 254**
osm	florida-3	1 714.07	**25 992**	0.11	23 471	65.56	**25 992**	1.34	**25 992**	8.20	**25 992**	7.80	**25 992**
	mas.-3	36 000.45	14 610	1.79	13 285	247.66	**14 757**	-	-	74.22	**14 757**	102.60	**14 757**
	utah-3	200.33	**16 090**	0.03	15 186	41.05	**16 090**	0.07	**16 090**	1.77	**16 090**	1.64	**16 090**
snap	as-skitter	-	-	0.04	134 500	45.35	135 976	-	-	1 904.18	**135 998**	287.87	**135 998**
	web-BerkStan	28.05	**135 172**	<0.01	133 309	36.21	135 156	0.04	**135 172**	0.17	**135 172**	0.15	**135 172**
ssmc	ga2010	159.38	**76 316**	0.01	76 297	59.10	76 297	0.07	**76 316**	0.34	**76 316**	0.33	**76 316**
	nh2010	2.28	**26 770**	<0.01	26 477	17.20	26 768	0.01	**26 770**	0.07	**26 770**	0.05	**26 770**
	ri2010	6 477.22	**66 963**	<0.01	65 979	32.71	66 960	0.02	**66 963**	0.15	**66 963**	0.12	**66 963**

overall	KaMIS BnR	HtWIS	HILS	Struction	m²wis+s	m²wis
# best	67/93	14/93	75/93	75/93	88/93	88/93
gmean ω	-	15 964	16 683	-	16 688	16 688
gmean t	-	0.01	25.01	-	0.60	0.66

Table 4.13: Average solution weight ω and time t (in seconds) required to compute ω for a representative sample of our instances comparing the best-performing algorithms. The **best** solutions among all algorithms are marked in bold. Rows are colored gray if KaMIS BnR or Struction computed an optimal solution. See Table B.3 for more details.

		METAMIS		M²WIS+S		M²WIS	
	Instance	t	ω	t	ω	t	ω
redOsm	alabama-3	5.41	**45 449**	23.35	**45 449**	23.30	**45 449**
	district-of-columbia-2	58.38	**100 302**	128.11	**100 302**	105.50	**100 302**
	district-of-columbia-3	1 347.00	142 910	4 133.86	**143 056**	1 674.92	**143 056**
	florida-3	3.08	**46 132**	7.09	**46 132**	7.04	**46 132**
	greenland-3	28.02	**11 960**	2 556.12	**11 960**	363.89	**11 960**
	hawaii-3	1 207.00	58 819	5 415.14	58 869	4 595.13	**58 870**
	idaho-3	21.23	**9 224**	1 560.69	**9 224**	271.12	**9 224**
	kansas-3	3.38	**5 694**	62.60	**5 694**	102.45	**5 694**
	kentucky-3	1 387.00	30 789	9 125.55	**31 107**	6 457.95	**31 107**
	massachusetts-3	2.22	**17 224**	63.29	**17 224**	103.09	**17 224**
	north-carolina-3	0.38	**13 062**	61.20	**13 062**	116.08	**13 062**
	oregon-3	11.56	**34 471**	150.93	**34 471**	230.12	**34 471**
	rhode-island-2	0.27	**43 722**	0.68	**43 722**	0.64	**43 722**
	rhode-island-3	449.70	**81 013**	3 175.87	**81 013**	1 660.12	**81 013**
	vermont-3	9.33	**28 349**	1 238.25	**28 349**	135.12	**28 349**
	virginia-3	9.08	**97 873**	144.95	**97 873**	146.12	**97 873**
	washington-3	62.35	**118 196**	1 357.56	**118 196**	257.62	**118 196**
	overall	METAMIS		M²WIS+S		M²WIS	
	# best		14/17		16/17		17/17
	gmean ω		36 153		36 179		36 179
	gmean t		20.89		357.31		145.07

Table 4.14: Comparison with METAMIS [59] on Instance Set 3. We give the solution weight ω and time t (in seconds) required to compute ω. Bold numbers indicate the **best** solution among the algorithms. As done by Dong et al. [59] we reduced the `osm` instances in advance using KaMIS BnR [149]. For METAMIS, the best result out of five runs are reported; we report the best solution out of four runs, each with a ten-hour time limit.

Comparison to METAMIS

Recently, Dong et al. [59] presented a novel heuristic for MWIS called METAMIS. Since their code is not publicly available, we compare the solution quality of our algorithm against the results presented in their work. Detailed per-instance results can be found in Table 4.14. We use the same pre-reduced `osm` instances as Dong et al. [59]. In their experiments, the time limit for METAMIS is 1 500 seconds. However, as our algorithm unfolds its full potential over a long period, and our algorithm focused on higher-quality solutions and not fast running times, we stayed with a ten-hour time limit. Moreover, note that the results have been computed on different machines. In summary, our M²WIS configurations compute the same or better solutions on all graphs compared to METAMIS. In total, we were able to improve three solutions compared to the METAMIS results with our configurations. Especially for large

instances, our approaches outperform the results stated in [59]. However, it is not clear whether METAMIS would compute equally good solutions for the instances where M^{2}WIS performed better with a longer running time.

4.4 Concurrent Iterated Local Search CHILS

In this section, we present a new concurrent iterated local search heuristic CHILS for the MWIS problem. It uses a baseline local search which extends up on the HILS algorithm, see Section 3.2.3 and a special subgraph, called D-Core. This subgraph is constructed using multiple heuristic solutions and is used to focus the search on difficult parts of an instance. The algorithm is described in detail in Section 4.4.1. Our approach CHILS outperforms existing heuristics across a wide variety of real-world instances. We also show that CHILS can be used to solve the vehicle routing instances from [60] and that it outperforms the two other heuristics, METAMIS and BSA, specifically designed for these instances, as shown in Section 4.4.2.

References. *This section is based on a publication which is joint work with Kenneth Langedal and Christian Schulz [104].*

Our Results. We propose a new concurrent iterated local search heuristic CHILS to compute large-weight independent sets very fast. In particular, our heuristic works by alternating between the full graph and the Difference-Core (D-Core), a subgraph constructed using multiple heuristic solutions. With this method and the GNN preprocessing LearnAndReduce, introduced in Section 4.2, we are able to outperform existing heuristics across a wide variety of real-world instances, including Instance Set 4 of vehicle routing instances.

4.4.1 The Algorithm Description

The proposed heuristic consists of two parts: (1) a simple iterated local search procedure we refer to as BASELINE, and (2) a new, concurrent heuristic called CHILS (Concurrent Hybrid Iterated Local Search). In the following section, we first give a high-level overview of the proposed approach and then provide a detailed description of each component of our heuristic.

High-Level Idea

An uncomplicated heuristic implemented very efficiently is often better in practice than a complicated one that runs slowly, especially when the heuristic makes heavy use

of random decisions. This can be seen from the results of programming competitions such as PACE (Parameterized Algorithms and Computational Experiments), where fast randomized local search has become the default heuristic strategy among the winning solvers [19, 99, 134]. Note that the PACE competition setting restricts the time to solve the problem. We also used this approach for our baseline local search. First, randomly alter a local area in the current solution. Then, greedy improvements can be applied in random order until a local optimum is found. Finally, if the new local optimum is worse than the previous, backtrack the changes and repeat. Compared to more complicated heuristics, the main benefit of our BASELINE heuristic is that it is inherently local. Regardless of the graph size, one iteration of the search will typically only touch vertices a few edges away from where the random alteration started. Backtracking to the best solution is also local, as long as queues are used to track changes. Using queues also differentiates our BASELINE from other heuristics, such as HILS [172], that make a copy of the entire solution instead.

The high-level idea of CHILS is a new metaheuristic we call Concurrent Difference-Core Heuristic. Instead of only trying to improve one solution, we maintain several and update each concurrently. At fixed intervals, the solutions are used to create a Difference-Core (D-Core) instance based on where the solutions differ. In other words, if a vertex is part of all or none of the solutions, it will not be part of the D-Core. The intuition is that the intersection of the solutions is likely to be part of an optimal solution, and where they differ indicates areas where further improvements could be made. Since there is no guarantee that the intersection of the solutions is part of an optimal solution, the Concurrent Difference-Core Heuristic alternates between looking for improvements on the original instance and the D-Core, where the D-Core is always constructed according to the current solutions. While the general Concurrent Difference-Core Heuristic can work using any heuristic method, our CHILS heuristic uses our iterated local search BASELINE.

Baseline Local Search

The outline of the baseline local search is shown in Algorithm 13. Each search iteration starts by picking a random vertex u from the graph. If the vertex is already in the solution $u \in S$, or the tightness of u is one, i. e., $|N(u) \cap S| = 1$, then an alternating augmenting path (AAP) is used to perturb the current solution. If u is not currently in the solution, it is added along with a random number of further changes in its close proximity. The vertices that see a change in their neighborhood are considered in "close proximity" to u. These vertices are continuously queued up in Q to find

greedy improvements more efficiently later. We also use the size of Q to control the amount of random changes. We stop perturbing the solution once $|Q|$ exceeds m_q, where m_q is a hyperparameter for the BASELINE heuristic. The benefit of using Q and m_q is that it stabilizes the computational cost for one iteration across different graph densities. For instance, the number of changes needed to fill Q in a dense area will be higher than in a sparse area.

Algorithm 13: Baseline Local Search

Data: Graph $G = (V, E, w)$, independent set S, max queue size m_q, and time limit t
Result: Improved independent set S

```
Q ← ∅                    // Always filled with vertices that observe changes to S
while time spent < t do
    cost ← ω(S)
    u ← uniform random from [0, |V| − 1]
    if u ∈ S or |N(u) ∩ S| = 1 then
        AAW-moves(G, S, u)
    else
        S = {u} ∩ S \ N(u)
        while |Q| < m_q do
            v ← random element from Q
            if v ∈ S then
                S = S \ {v}
            else
                S = {v} ∩ S \ N(v)
    GREEDY(G, S, Q)
    if w(S) < cost then
        undo changes to S
return S
```

After perturbing the current solution, greedy improvements are made to find a new local optimum. Having stored the candidate vertices in Q speeds up the search for this new local optimum in sparse graphs. We incorporate three greedy improvement operators for GREEDY in BASELINE described in the following.

Neighborhood Swap. For a vertex $u \notin S$, if $\omega(u) > \omega(N(u) \cap S)$, then the independent set obtained by inserting the vertex u and removing all neighbors of u that

are currently in the solution, i.e. $S = \{u\} \cup (S \setminus N(u))$, leads to an independent set of higher weight. The total sum of the weights of each neighbor currently in the solution $\Sigma_{v \in N(u) \cap S} w(v)$ is maintained at all times. This additional information allows the neighborhood swap to be checked in $\mathcal{O}(1)$ time.

One-Two Swap. For a vertex in the current solution $u \in S$, consider the set $T = \{v \in N(u) \mid N(v) \cap S = \{u\}\}$ with one-tight vertices in the neighborhood of u. If there exists a pair of vertices $\{x, y\} \subseteq T$ such that $\{x, y\} \notin E$ and where $w(u) < w(x)+w(y)$, then swapping u for x and y leads to a better solution. Examining all one-two swaps can be done in $\mathcal{O}(m)$ time [7].

Alternating Augmenting Walk. We use a slightly modified version of the alternating augmenting path (AAP) introduced by Dong et al. [59]. An alternating augmenting walk (AAW) is a sequence of vertices that alternate between being in and out of the solution S. AAW moves are used both for perturbation and finding greedy improvements. When used for perturbation, the AAW is always extended in a random direction. After no more vertices can be added to the walk, the entire AAW is applied to the solution unless a strictly improving prefix of the walk exists. When searching for greedy improvements, the walk always starts from a one-tight vertex and extends in the best direction possible.

Let U be the set of vertices on the AAW in S, and $\overline{U}$ be the vertices on the AAW not in S. A valid AAW has the property that the set S' obtained by applying the AAW, i.e. $S' = (S \setminus U) \cup \overline{U}$ is also an independent set. This means $\overline{U}$ must also be an independent set where $N(\overline{U}) \cap S = U$. The main purpose of AAWs is to find improving walks where $\omega(\overline{U}) > \omega(U)$ and swap the vertices in U for those in $\overline{U}$ to obtain a heavier independent set. As a secondary use case, they can also perturb the current solution. The benefit of using AAWs instead of random perturbation is that the cardinality of the new solution can only be one less than the original. Constructing an AAW starts with a single vertex $u \in S$ or a pair of vertices $v \notin S, u \in S$, such that $N(v) \cap S = \{u\}$. The AAW is then extended from the last vertex u on the walk two vertices $x \in N(u), y \in N(x)$ at a time, such that the following three conditions are met.

1. x is not adjacent to any vertices in $\overline{U}$
2. x is not currently in $\overline{U}$
3. x is adjacent to precisely two vertices in the solution $N(x) \cap S = \{u, y\}$

The main difference to the definition of an AAP given for METAMIS [59] is that y is allowed to already be on the path. This results in a walk rather than a path. With

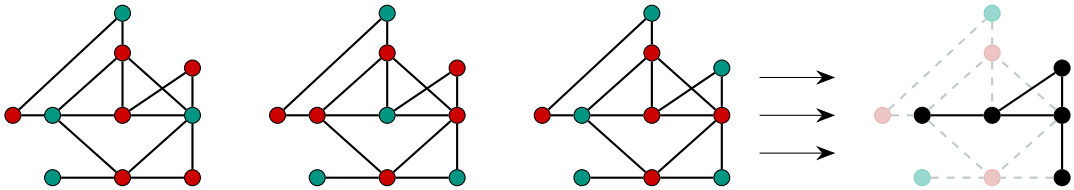


Figure 4.12: Illustration for the D-Core. Vertices that are part of all or none of the solutions are not part of the D-Core.

the path constraint as in METAMIS, an x, y pair that loops back to an earlier vertex on the path is not a valid extension of the AAP. After applying this AAP (without x, y), x would be a free vertex. In our relaxed definition, we can pick up these free vertices directly. A downside with our definition is that the walks could remain more local than a straight path away from the source vertex.

The CHILS Algorithm

We give an overview of our concurrent heuristic CHILS in Algorithm 14. First, we run BASELINE on P different solutions for the full graph with a time limit of t_G seconds. Each solution is assigned a different random seed and slightly modified max queue size (m_q) to increase their difference. We also assign each solution an *ID* and always keep track of the best solution found so far. Note that the ID of the best solution can change during the algorithm execution. After improving the solutions on the full graph, i. e., after $P \times t_G$ seconds, we construct the D-Core instance. This is done by removing vertices that are part of all or none of the P independent sets, see Figure 4.12 for an example.

On the D-Core, we again start our BASELINE local search P times with a time limit of t_C to generate new solutions. We extend an independent set on the D-Core to the full graph by adding the vertices in the intersection of all P solutions. This independent set can replace the previous solution with the same ID. However, for solutions with an even ID, as well as for the best solution found so far, replacements are only made if the new solution is of higher weight. Letting half of the solutions always accept the D-Core solution helps diversify the search.

Using the D-Core helps concentrate the local search on the more difficult regions of the graph, where our P solutions differ. However, since the areas where they agree are not necessarily part of an optimal solution, CHILS alternates between using the BASELINE on the original instance and the recomputed D-Core based on the current P solutions.

Algorithm 14: The CHILS Algorithm

Data: Graph $G = (V, E, \omega)$, number of solutions P, max queue size m_q, time limit for the full graph t_G, time limit for the D-Core t_C, and overall time limit t

Result: Independent set S

$C = \{S_1, S_2, \ldots, S_P\}$ // Using GREEDY$(G, \emptyset, V)$ with different seeds

while *time spent* $< t$ **do**

  parallel for $S_i \in C$ **do**

    $S_i = \text{BASELINE}(G, S_i, m_q + 4i, t_G)$

  $G' \leftarrow$ compute D-Core using C

  parallel for $S_i \in C$ **do**

    $S' = \text{BASELINE}(G', \emptyset, m_q + 4i, t_C)$

    if $\omega(S') + \mathit{offset} \geq \omega(S_i)$ ***or*** *(i is odd* ***and*** S_i *is not best)* **then**

      apply S' to S_i

  if $|V'| < 500$ **then**

    PARALLEL_PERTURBE(C)

return $S \in C$ *with largest weight*

If the number of vertices in the D-Core falls below some small constant value, it indicates that the P solutions are all quite similar. Since this reduces the benefit of our approach, we perturb all solutions with an odd id, except for the best solution, and without backtracking in the case where the new local optimum is worse. In Algorithm 14, this is shown as PARALLEL_PERTURB on Line 14, where perturbing one solution is done as described in lines 9-16 of Algorithm 13. As with accepting replacement solutions from the D-Core, perturbing only half the solutions here helps to diversify the search and escape local optima.

Parallel CHILS. Our CHILS approach is easily parallelizable, with a natural choice for the number of solutions being exactly the number of cores available on the machine, allowing each solution to be improved simultaneously. In this configuration, the worst-case scenario is similar to running the underlying BASELINE local search sequentially, at least when technical details such as memory bandwidth and dynamic clock speeds are ignored. For larger numbers of solutions, the parameter P should be divisible by the number of threads running to ensure that no threads are idle.

4.4.2 Experimental Evaluation

The following section introduces the experimental setup and establishes the dataset used for evaluating the proposed approaches. Then, we present state-of-the-art comparison for BASELINE and CHILS. Finally, we present results for the parallel scalability of CHILS.

Methodology. All the experiments were run on Machine 2. Both CHILS and BASELINE were implemented in C and compiled with GCC version 11.4.0 using the -O3 flag. OpenMP is used for the parallel implementations. We evaluate eight instances in parallel for the sequential experiments. To ensure fairness between the algorithms, the instances start in the same order for each code, and only one program is evaluated at a time. We evaluate each program once for each instance.

We compare our algorithms BASELINE and CHILS to the state-of-the-art heuristic algorithms HtWIS presented in [107] by Gu et al., the hybrid iterated local search HILS by Nogueira et al. [172], the memetic algorithm m^{2}wis+s by Großmann et al. [103], the new metaheuristic METAMIS by Dong et al. [59], and the Bregman-Sinkhorn algorithm BSA by Haller and Savchynskyy [113]. The first three, HtWIS, HILS, and HILS, were all used in their default configurations. The source code for METAMIS is not publicly available, and therefore, we had to use the numbers reported in [59]. They used the Amazon Web Service r3.4xlarge compute node running Intel Xeon Ivy Bridge Processors and wrote the implementation of their heuristic in Java. They also run their algorithm five times with different seeds and report the best solution found. For the Bregman-Sinkhorn algorithm BSA, we use the variation that produces integer solutions only after reaching 0.1 % relative duality gap for the LP relaxation by recommendation from the authors [114][2].

Datasets. For the experiments, we use the Instance Set 2, which is a subset of Instance Set 1 where easy-to-reduce instances have been removed. This set is also used in Section 4.2 to evaluate the LearnAndReduce preprocessing. It contains instances from the Stanford Large Network Dataset Repository (`snap`) [155], OpenStreetMaps (`osm`) [1, 20, 37], the SuiteSparse Matrix Collection (`ssmc`) [53] as well as dual graphs of well-known triangle meshes (`mesh`) [185], and 3d meshes derived from simulations using the finite element method (`fe`) [210]. Additionally, we perform experiments on Instance Set 4 of vehicle routing instances introduced recently by Dong et al. [60].

[2]The exact command we used was mwis_json −l temp_cont −B 50 −−initial−temperature 0.01 −g 50 −b 100000000 −t [seconds] [instance]

Instance Set 4 (Vehicle Routing)**.** *This instance set contains all 37 vehicle routing instances (`vr`) introduced by Dong et al. [60]. Detailed graph properties are given in Table A.3 in the Appendix.*

Initial warm-start solutions derived from the application and clique information for the graphs are also provided for these instances. The clique information is a clique cover of the graph, i. e., a collection of potentially overlapping cliques that cover the entire graph. The approaches METAMIS [59] and BSA [113] use this clique information.

Parameter Tuning

In this section, we perform experiments to find good choices for the number of concurrent solutions P and the time t spent improving them in the sequential version of CHILS. We choose a subset of our two sets of instances for parameter tuning. We took six graphs from each set. From each graph class in Instance Set 2, we select two graphs randomly, such that the reduced instance has more than 500 vertices. From the Instance Set 4, we took three graphs with more than 500 000 vertices and three with less than 500 000 vertices. Note that we did not include any *mesh* or *ssmc* graphs since these are all reduced to less than 500 vertices. The instances chosen for these experiments are marked with a $\star$ in Tables A.2 and A.3 in the Appendix.

We begin our experiments with the parameter defining the number of concurrent solutions, $P \in \{8, 16, 32, 48, 64\}$. The second parameter, highly correlating with P, is the time spent per local search run. For this experiment we set $t = t_G = t_C$ and test all $t \in \{0.1, 1, 2.5, 5, 10, 20, 30\}$ seconds. Recall that t_G is the time spent on the original graph and k_C is the time spent on the D-Core. We present the geometric mean solution weight after one hour for the different configurations of these two parameters in Table 4.15.

We can see that the best choice for t gets lower as the number of solutions P increases. Comparing the configurations with the most solutions, i. e., $P = 64$, the solution quality is worse than using fewer concurrent local search runs for all t values tested. The best configuration we found is $P = 16$ and $t = 10$ seconds, resulting in a geometric mean weight of 14 132 120. Note that these optimal choices are only for running CHILS sequentially.

Observation 4.4.1: Best Performing CHILS Configuration. The best parameter configuration we found experimentally was $P = 16$ and $t = 10$ for running CHILS sequentially with a time limit of one hour per instance.

	$P = 8$	$P = 16$	$P = 32$	$P = 64$
$t = 0.1$	14 119 825	14 119 386	14 119 516	14 100 937
$t = 1$	14 124 084	14 126 354	14 127 388	14 117 719
$t = 2.5$	14 127 282	14 128 074	14 130 823	14 118 203
$t = 5$	14 127 913	14 129 545	14 129 347	14 117 284
$t = 10$	14 127 684	**14 132 120**	14 124 680	14 116 758
$t = 20$	14 130 214	14 125 181	14 119 467	14 110 491
$t = 30$	14 126 896	14 118 547	14 115 472	14 112 364

Table 4.15: Geometric mean weight computed by the sequential CHILS after one hour for different configurations for the number of solutions P and time limits for the local search $t = t_G = t_C$. Note that the time t is spent per solution in P and is not divided between them.

State-of-the-Art Comparison

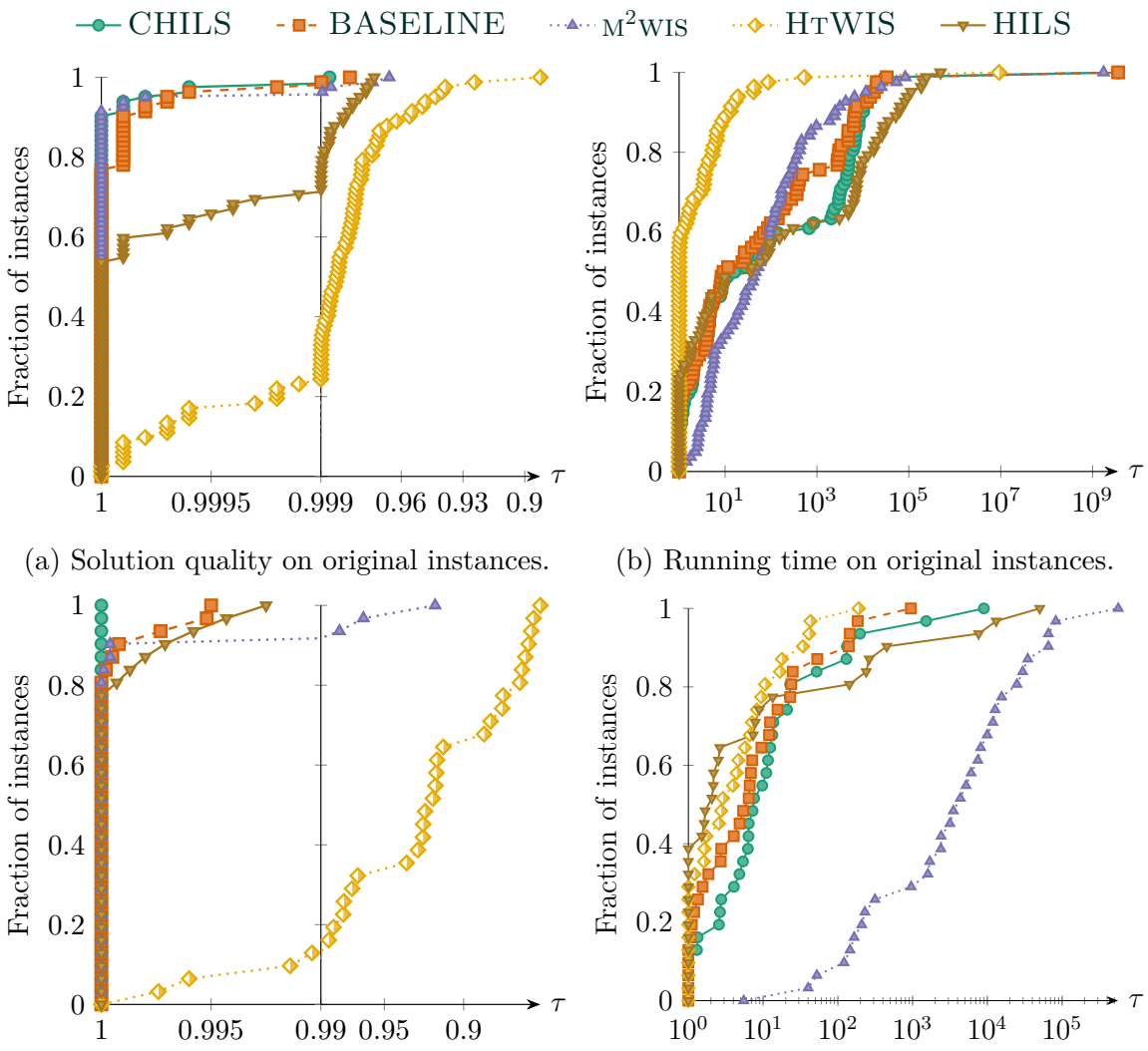
The state-of-the-art comparison in this section is divided into two parts. We start by discussing results for the Instance Set 2, a subset of our main set Instance Set 1, where we removed graphs that are simple to reduce. Here we compare our approaches BASELINE and CHILS to state-of-the-art heuristics HtWIS presented in [107] by Gu et al., the hybrid iterated local search HILS by Nogueira et al. [172] as well as the memetic algorithm m^{2}wis+s introduced in Section 4.3. The second part of this section is dedicated to the other algorithms and results for the vehicle routing instances, Instance Set 4. For these instances, we compare our heuristics with the results of METAMIS [59][3]. Additionally, we compare against the new Bregman-Sinkhorn algorithm BSA by Haller et al. [113]. We do not evaluate BSA on Instance Set 2 as BSA requires clique information that is only available for the vehicle routing instances. Similarly, the vehicle routing instances differ significantly from previously established testing instances. The older state-of-the-art heuristics are not designed or implemented for these instances and perform significantly worse. In splitting the state-of-the-art comparison as described, we evaluate each heuristic in spirit with what they were designed for while demonstrating that BASELINE and CHILS are competitive in both categories.

Table 4.16 presents results for a subset of graphs from Instance Set 2. We chose the largest four graphs from each graph class for this subset. The results on the full set can be found in Table B.4 in the Appendix. We can see that both our algorithms

[3]The code is not publicly available, therefore, we could not rerun the experiments.

BASELINE and CHILS outperform HtWIS and HILS in terms of solution quality on *every* instance. For large instances, such as the *osm* instances, we find these best solutions even faster than any competitor finding their best solution within the time limit. When comparing against m^{2}wis+s, we observe that especially for the *mesh* and *snap* instances, our approaches find slightly worse solutions and need more time. However, m^{2}wis+s already uses many of the reductions we incorporated in the LearnAndReduce preprocessing, which we did not use for BASELINE or CHILS. Comparing Table 4.16 to Table 4.17 clearly shows the importance of our preprocessing routine. In Table 4.17, we present the results on the same instances, which have already been reduced with LearnAndReduce. From this, it is clear that all evaluated methods benefit from the preprocessing. On these instances, CHILS finds the best solutions on *all* reduced instances.

In Figure 4.13, we show performance profiles comparing the different algorithms' solution quality and running time on the full set of original and reduced instances. Note that most of the solutions on these instances found by m^{2}wis+s are optimal [103]. On approximately 60 % of the original instances HtWIS finds the solution fastest, see Figures 4.13b. However, the fast running times come with a decrease in solution quality of up to 10 %, as seen in Figure 4.13a. On more than 90 % of the instances CHILS finds the best solution while having comparable running times as the other schemes except for HtWIS. The m^{2}wis+s algorithm performs very similarly to CHILS on the original instances. However, on the reduced instances, m^{2}wis+s has the longest running times; see Figure 4.13d. Note that the initial reductions used in m^{2}wis+s are also included in the LearnAndReduce framework. All other algorithms have similar running times on the reduced instances. From Figure 4.13c, it follows that all algorithms except for HtWIS find the same solution on almost 80 % of the reduced instances. On the other 20 % CHILS finds the best solutions which are up to 0.7 % better than the HILS solutions and more than 7 % better than m^{2}wis+s. On more than 85 % HtWIS computes solutions more than 1 % worse than the best solutions found, and for approximately 30 % of the instances these solutions are even more than 10 % worse.



(a) Solution quality on original instances.

(b) Running time on original instances.

(c) Solution quality on reduced instances.

(d) Running time on reduced instances.

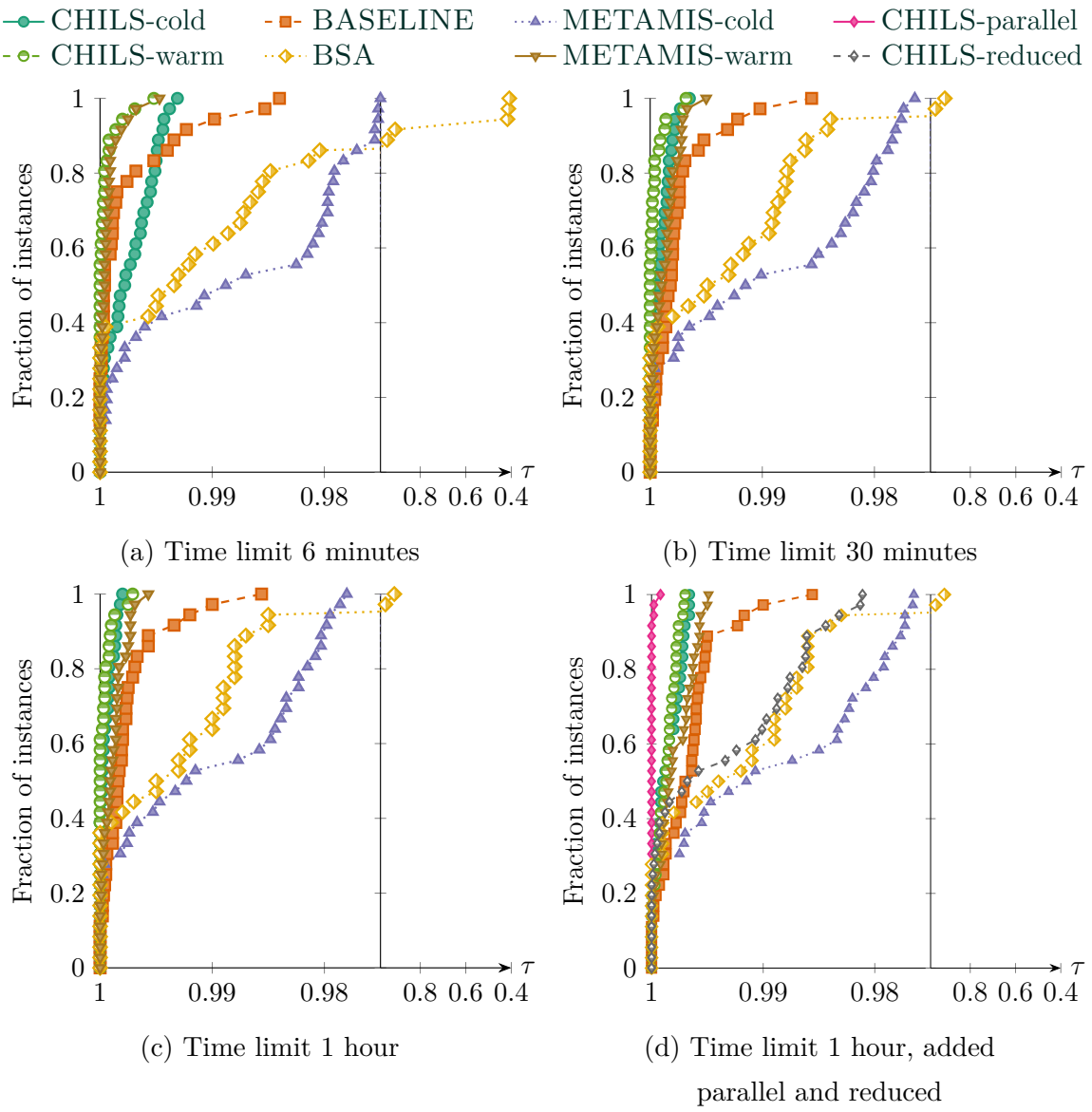
Figure 4.13: Performance profiles for solution quality and running time for original and reduced instances, not including the vehicle routing instances. The reduction offset and reduction time are not added for the reduced instances, and fully reduced instances are not included. The vertical line in the plots on the left indicates a change from one scale to another.

Observation 4.4.2: Effect of LearnAndReduce. All algorithms tested benefit from running the LearnAndReduce preprocessing. CHILS finds the best solution on *all* reduced instances, while performing similar to HILS regarding running time. It is slightly slower than HtWIS, which computes the worst solution quality on these instances. Without preprocessing, the improvement of CHILS over HtWIS and HILS regarding solution quality increases. Here, only M²WIS+S, which already uses reductions, can find better results than CHILS on some graphs.

Next, we investigate the results for the Instance Set 4. Note that the vertex weights for these instances are very large, which leads to only small percentage improvements despite significant differences between the solutions. Here, we only compare against the competitors METAMIS and BSA. In contrast to these competitors, CHILS and BASELINE are running on the full graph without exploiting the clique information also provided. On these instances, most of the solutions found are not optimal, evident by the significantly larger solutions we later find using our parallel CHILS. We add a warm-start configuration for CHILS where we start with the provided initial solution. The METAMIS numbers for cold and warm-starts are taken from [59]. In Figure 4.14, we present performance profiles to compare the solution quality achieved by the algorithms with different time limits. The first two show the state-of-the-art comparison of solution quality achieved with a 6 and 30 minutes time limit. As expected, the configurations with the warm start perform best with the shortest time limit, see Figure 4.14a. However, if no initial solution is known, our two variants, BASELINE and CHILS, significantly outperform all competitor configurations. Note that BASELINE is performing better than CHILS on around 80 % of the instances here, since the configuration for CHILS is optimized for running with a one-hour time limit, see Section 4.4.2. Compared to the second profile on the right, we can see that with more time, the CHILS solutions improve most, even surpassing the METAMIS-Warm results in some instances. Within this time limit CHILS-Warm finds the best solutions on almost 60 % of the instances. Furthermore, both CHILS configurations compute solutions that are at most 0.3 % worse than the best-found solution. On the other hand, on more than 38 % of the instances, BSA and METAMIS-Cold find solutions which are more than 1 % worse than the best solutions found on the respective instances.

In Figure 4.14c, we present the state-of-the-art comparison where all algorithms have an hour time limit and run on the original instances. Here, we can see that CHILS with and without the initial solution performs generally the best. The results shown here are very similar to those computed within the shorter time limit of 30 minutes, see Figure 4.14b.

However, this changes for the profile in Figure 4.14d. Here, we added a configuration called CHILS-Parallel, which utilizes the warm start solution and runs in parallel with the configuration of $P = 64$ and $t = 5$ seconds, using 16 parallel threads, i. e., without simultaneous multithreading. Additionally, we used LearnAndReduce and run CHILS on the reduced instances. With this configuration, CHILS-Reduced, we can not compete with the other configurations on most instances since, for these large instances, the reduction rules do not work well; see also Table A.3 in the



(a) Time limit 6 minutes

(b) Time limit 30 minutes

(c) Time limit 1 hour

(d) Time limit 1 hour, added parallel and reduced

Figure 4.14: Performance profiles for state-of-the-art comparison of solution quality on the vehicle routing instances with different time limits. In Figure 4.14d, results computed with parallel CHILS and CHILS on instances reduced with LearnAndReduce are added.

Appendix. Testing even the fast reduction rules on these graphs takes considerable time. However, on MR-W-FN or MW-D-01, for example, using reduction rules helps to find better solutions; see Table 4.18. Considering all of our variants, we outperform all other schemes in all but two instances; see Figure 4.14 and Table 4.18. Furthermore, in contrast to METAMIS, our approach does not depend on having good initial solutions. Nevertheless, using a warm start solution can improve the performance.

Observation 4.4.3: Performance on Vehicle Routing Instances. On the vehicle routing instances without initial solutions, even our BASELINE approach outperforms the competitors. The BASELINE approach is especially good with a low time limit. Overall, CHILS outperforms *all* the other heuristics regarding solution quality after 6 minutes, 30 minutes, and 1 hour.

Parallel Scalability

For the parallel results shown in Figure 4.14d, we used the same machine as the other heuristics to ensure fairness between each program with regard to solution quality. To gain more detailed insight into the parallel scalability of our approach, we utilize a larger machine with an AMD EPYC 9754 128-core processor for our scalability experiments. Furthermore, CHILS as defined in Section 4.4.1 relies on wall time to alternate between local search on the full graph and the D-Core. This introduces variance between runs and makes it difficult to conduct scalability experiments. Therefore, we change the implementation for this section to perform a fixed number of local search iterations instead, where one iteration refers to the while loop starting on Line 2 in Algorithm 13. We also set a fixed number of CHILS iterations, referring to the while loop starting on Line 2 in Algorithm 14. By removing the wall-clock measures and using the same random seed, we ensure that the parallel and sequential versions perform the exact same computations and reach the same solution in the end.

The configuration we use for these experiments consists of 1,000 local search iterations, 10 CHILS iterations, and $P = 128$. Depending on the instance, this ratio of local search on the full graph and D-Core is reasonably close to the wall-clock version with $t_G = t_C = 10$. Each run was repeated five times, and the best measurement is used. We use the best measurement because, in this configuration, there is no randomness between runs. Any difference we observe in execution time is solely due to factors outside our control, such as fluctuations in clock speed and other programs running on the machine. As such, the best measure is the closest to the true execution time.

The speedup numbers for each instance are shown in Figure 4.15. The best scaling instance reaches a speedup of 104, while the worst is still 28 times faster than the sequential program. Figure 4.15 also shows the amount of memory allocated for each instance. This includes the graph, which we store using the compressed sparse row format, and additional data structures required by the heuristic. All memory allocations are done upfront before starting the heuristic, and the appropriate thread initializes any thread-local data. The CPU we use has a combined L3 cache of 256 MiB. There is a clear drop in speedup around the point when the data no longer fits

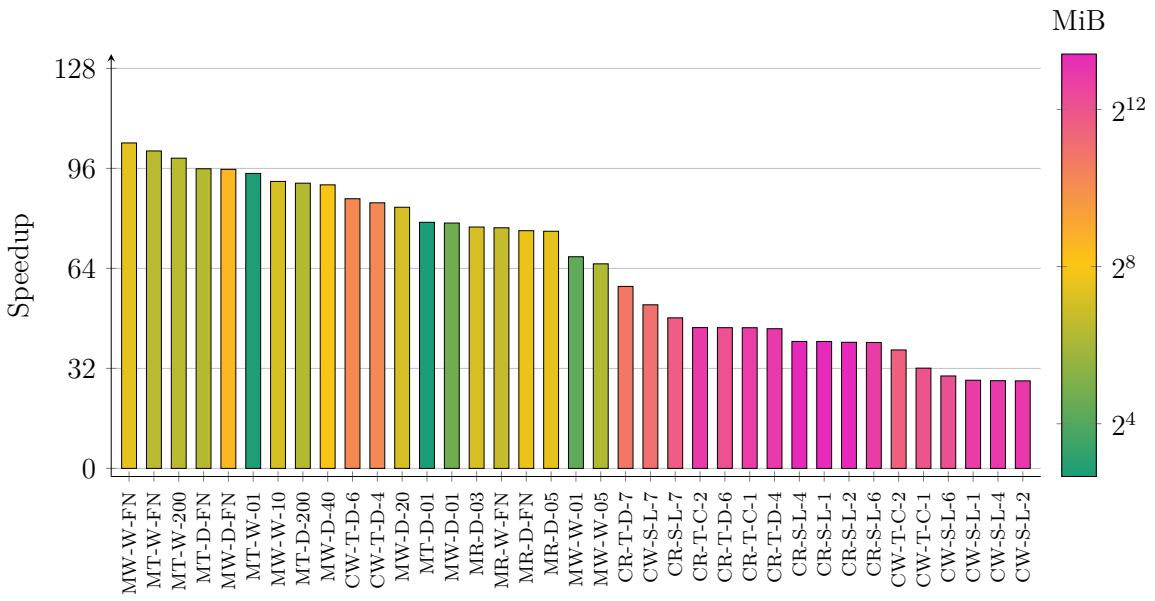


Figure 4.15: Bar chart showing the speedup on each instance using 128 threads compared to the sequential CHILS. The colors indicate the amount of allocated memory for each instance. The total amount of L3 cache for this machine is 256 MiB, and smaller instances fit entirely in the cache.

in the cache. This indicates that the memory bandwidth is the main bottleneck for larger instances. In terms of load balancing, there could be variations in how much work it takes to perform 1,000 local search iterations for each solution. This is because each solution has a different random seed and m_q parameter. Note that this is not an issue for the version presented in Section 4.4.1, since that version uses wall time instead of local search iterations. Table A.7 in the Appendix gives detailed execution time, speedup, and the amount of memory used for each instance.

Observation 4.4.4: Scalability Results. The parallel version of CHILS scales well with the number of threads. We achieve a speedup of up to 104 on the best instances, while the worst instances still show a speedup of 28. The experiments indicate, that the memory bandwidth is the main bottleneck for larger instances, as the speedup drops when the data no longer fits in the cache.

Table 4.16: Solution weight ω and time t (in seconds). The **best** solutions among all algorithms are marked in bold. Here, we give the results of the four largest, non-reduced graphs of each class from Instance Set 2. The results on all graphs are presented in Table B.4 in the Appendix.

		HtWIS		HILS		m²wis+s		BASELINE		CHILS	
Instance		ω	t	ω	t	ω	t	ω	t	ω	t
fe	body	1 645 650	0.06	1 678 383	3 471.02	**1 680 182**	2 958.88	**1 680 182**	1 824.22	**1 680 182**	165.03
	ocean	6 803 672	0.12	7 081 151	3 384.46	**7 248 581**	3.38	7 139 076	16.75	7 210 830	2 863.50
	pwt	1 153 600	0.04	1 175 710	3 580.37	1 176 685	3 601.03	1 178 479	49.24	**1 178 708**	537.13
	rotor	2 591 456	0.31	2 651 471	3 255.65	2 596 217	3 601.17	2 663 757	880.70	**2 665 805**	3 179.83
mesh	buddha	57 508 556	0.52	56 086 272	3 600.05	**57 555 880**	4.26	57 554 770	3 380.94	57 555 764	2 571.52
	dragonsub	32 163 872	0.28	31 833 085	3 599.67	**32 213 898**	1.62	32 212 574	3 409.68	32 213 769	2 857.15
	ecat	36 606 394	0.48	36 122 506	3 600.03	**36 650 298**	2.60	36 648 982	3 279.35	36 650 039	2 155.87
	turtle	14 247 883	0.14	14 238 437	3 597.64	**14 263 005**	0.58	14 262 790	2 753.32	14 262 993	1 665.35
osm	d.-of-c.-3	210 461	330.18	227 652	939.72	227 681	1 890.32	**227 683**	948.94	**227 683**	258.43
	hawaii-3	134 703	1 423.03	141 041	3 213.04	141 070	2 611.52	141 074	34.12	**141 095**	160.02
	kentucky-3	97 906	3 077.14	100 510	1 292.90	99 918	3 684.53	100 507	35.39	**100 511**	160.03
	rhode-i.-3	190 341	255.58	201 753	1 190.44	**201 771**	1 886.87	**201 771**	366.29	**201 771**	15.34
snap	as-skitter	124 141 373	1.17	123 207 288	3 597.32	**124 157 729**	2 828.19	124 141 585	3 545.45	124 111 689	3 491.32
	roadNet-CA	111 325 524	0.72	108 370 752	3 600.18	**111 360 828**	4.12	111 355 248	3 496.83	111 358 047	3 579.38
	soc-LiveJ.	283 922 214	9.22	279 386 267	< 0.01	**284 036 239**	1 737.39	283 951 703	3 599.46	283 924 551	3 562.85
	soc-p.-rel	83 920 370	39.06	82 984 397	3 598.22	81 112 447	3 637.19	83 971 405	3 369.22	**83 982 856**	3 520.91
ssmc	ca2010	16 792 827	0.55	16 591 401	3 597.70	16 867 553	3 601.26	16 867 987	2 650.77	**16 869 278**	2 613.13
	fl2010	8 719 272	0.40	8 694 303	3 599.61	**8 743 506**	183.77	8 742 384	2 941.69	8 743 309	2 832.06
	ga2010	4 639 891	0.21	4 640 168	3 595.16	**4 644 417**	61.28	4 644 398	579.13	**4 644 417**	650.90
	il2010	5 963 974	0.39	5 965 735	3 599.95	**5 998 539**	17.93	5 998 017	2 516.42	5 998 520	3 314.47

Table 4.17: Solution weight ω and time t (in seconds). The **best** solutions among all algorithms are marked in bold. Here, we give the results of the graphs from Table 4.16 reduced with LearnAndReduce. Fully reduced graphs are omitted, and the reduction offset and reduction time are included in the results. All algorithms are run for one hour on the reduced instances, where the LearnAndReduce time was not limited. The results for all graphs are presented in Table B.5 in the Appendix. Detailed per-instance results for the reduction time and offset are given in Table A.2.

		HtWIS		HILS		m^{2}wis+s		BASELINE		CHILS	
Instance		ω	t	ω	t	ω	t	ω	t	ω	t
fe	body	1 680 066	0.60	**1 680 182**	0.61	**1 680 182**	60.63	**1 680 182**	0.69	**1 680 182**	0.68
	pwt	1 169 768	2.21	1 178 273	1 946.51	**1 178 735**	3 196.24	**1 178 735**	9.31	**1 178 735**	9.84
	rotor	2 599 908	5.41	2 653 478	2 764.03	2 617 855	3 606.23	2 664 500	347.16	**2 666 222**	3 251.99
osm	d.-of-c.-3	217 052	395.81	227 586	734.53	227 622	3 450.79	227 205	350.73	**227 656**	450.50
	hawaii-3	135 846	1 659.20	140 963	1 068.11	141 076	4 427.01	140 976	1 023.18	**141 095**	1 785.05
	kentucky-3	99 310	3 421.08	100 475	2 582.95	100 193	5 125.45	**100 511**	2 167.95	**100 511**	2 385.14
	rhode-i.-3	197 126	306.81	201 654	184.23	201 767	3 171.02	201 480	167.51	**201 771**	333.07
snap	as-skitter	124 156 703	3.46	**124 157 729**	8.63	**124 157 729**	1 300.24	124 157 700	3.45	**124 157 729**	33.49
	soc-LiveJ.	284 035 711	23.49	**284 036 239**	23.65	**284 036 239**	1 291.80	**284 036 239**	23.55	**284 036 239**	23.55
	soc-p.-rel	83 913 297	669.24	83 762 908	4 243.55	81 480 804	4 311.48	83 986 271	4 175.34	**83 991 718**	3 975.30
ssmc	fl2010	8 742 642	3.62	**8 743 506**	37.95	**8 743 506**	1 479.03	**8 743 506**	3.65	**8 743 506**	3.65

Table 4.18: Results for vehicle routing instances by Dong et al. [60]. The results for METAMIS were taken from the paper by Dong et al. [59] since the code is not publicly available. Note that the authors of METAMIS used the best out of four runs, while the results for BASELINE, CHILS, and BSA were only run once. All the results show the best solution found after one hour.

	Cold Start				**Warm Start**		**Reduced**	**Parallel**
Instance	METAMIS	BSA	BASELINE	CHILS	METAMIS	CHILS	CHILS	CHILS
CR-S-L-1	5 588 489	5 639 292	5 694 508	5 698 608	5 692 891	5 701 015	5 610 201	**5 718 230**
CR-S-L-2	5 691 892	5 729 194	5 785 140	5 798 470	5 784 034	5 799 458	5 715 463	**5 813 362**
CR-S-L-4	5 681 336	5 725 525	5 781 519	5 791 447	5 777 081	5 792 758	5 698 425	**5 806 975**
CR-S-L-6	3 859 513	3 900 370	3 936 944	3 938 946	3 936 137	3 946 157	3 890 733	**3 952 205**
CR-S-L-7	1 989 879	2 006 810	2 017 624	2 021 238	2 019 428	2 022 339	2 006 857	**2 025 681**
CR-T-C-1	4 654 419	4 717 754	4 742 508	4 744 119	4 743 040	4 750 570	4 702 548	**4 760 428**
CR-T-C-2	4 874 346	4 934 488	4 969 512	4 975 069	4 968 952	4 976 613	4 918 257	**4 987 562**
CR-T-D-4	4 817 281	4 875 268	4 912 984	4 919 079	4 911 646	4 922 752	4 864 858	**4 932 826**
CR-T-D-6	2 970 011	3 009 020	3 024 044	3 027 211	3 024 523	3 029 782	2 999 231	**3 033 189**
CR-T-D-7	1 440 281	1 453 990	1 460 328	1 461 099	1 460 240	1 460 584	1 451 070	**1 462 239**
CW-S-L-1	1 634 950	1 645 459	1 660 063	1 660 472	1 660 815	1 662 580	1 646 548	**1 663 763**
CW-S-L-2	1 708 820	1 713 535	1 737 052	1 738 307	1 738 128	1 739 670	1 723 914	**1 743 532**
CW-S-L-4	1 725 591	1 730 641	1 751 204	1 754 409	1 753 803	1 756 602	1 735 633	**1 759 452**
CW-S-L-6	1 158 925	1 162 552	1 175 335	1 177 272	1 177 156	1 176 354	1 166 774	**1 178 467**
CW-S-L-7	587 288	588 279	594 312	594 598	593 891	593 807	590 834	**594 757**
CW-T-C-1	1 317 775	1 325 862	1 336 232	1 338 085	1 338 064	1 340 085	1 323 170	**1 341 888**
CW-T-C-2	931 802	935 238	944 902	946 217	945 886	945 913	935 907	**947 644**
CW-T-D-4	457 185	459 575	460 955	460 734	461 056	461 108	459 543	**461 475**
CW-T-D-6	457 790	459 238	461 088	461 354	461 312	461 101	460 227	**461 709**

Continued on next page

Table 4.18 continued from previous page.

	Cold Start				Warm Start		Reduced	Parallel
Instance	METAMIS	BSA	BASELINE	CHILS	METAMIS	CHILS	CHILS	CHILS
MR-D-03	1 754 110 286	1 757 141 345	1 752 894 190	1 758 762 733	1 757 227 519	1 758 429 721	1 758 775 738	**1 759 255 435**
MR-D-05	1 786 342 921	1 788 812 740	1 781 868 583	1 789 601 100	1 787 849 777	1 789 537 256	1 789 944 187	**1 790 776 639**
MR-D-FN	1 797 573 192	1 801 983 754	1 794 240 970	1 801 499 264	1 799 452 160	1 800 375 890	1 801 727 328	**1 802 925 854**
MR-W-FN	5 358 386 615	5 386 842 780	5 385 799 979	5 385 799 979	**5 386 842 781**	**5 386 842 781**	**5 386 842 781**	**5 386 842 781**
MT-D-01	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**	**238 166 485**
MT-D-200	287 048 909	**287 155 108**	287 042 596	287 086 442	287 048 081	287 042 596	287 086 442	287 086 442
MT-D-FN	**290 866 943**	**290 866 943**	**290 866 943**	**290 866 943**	290 771 450	290 771 450	**290 866 943**	**290 866 943**
MT-W-01	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**	**312 121 568**
MT-W-200	383 961 323	384 056 011	383 974 084	384 052 157	383 985 408	384 052 017	384 052 157	**384 056 012**
MT-W-FN	390 854 593	390 869 890	**390 869 891**	**390 869 891**	**390 869 891**	**390 869 891**	**390 869 891**	**390 869 891**
MW-D-01	476 334 711	476 298 607	475 180 516	476 328 138	475 987 082	476 120 423	**476 440 656**	476 360 408
MW-D-20	525 124 575	526 857 183	523 423 307	526 883 302	525 486 034	526 333 489	526 648 329	**527 498 481**
MW-D-40	536 520 199	**539 036 121**	533 671 730	538 302 190	536 735 155	537 485 389	538 409 586	538 596 069
MW-D-FN	541 918 916	545 554 192	541 193 604	544 451 893	543 857 187	544 205 239	544 246 489	**545 711 017**
MW-W-01	**1 270 305 952**	**1 270 305 952**	**1 270 305 952**	1 270 235 200	1 269 344 846	1 269 344 846	**1 270 305 952**	**1 270 305 952**
MW-W-05	**1 328 958 047**	1 326 236 043	1 327 478 708	1 328 043 785	**1 328 958 047**	**1 328 958 047**	1 328 043 785	**1 328 958 047**
MW-W-10	1 342 899 725	1 280 286 209	1 340 268 013	1 342 808 634	**1 342 915 691**	**1 342 915 691**	1 342 809 954	**1 342 915 691**
MW-W-FN	**1 350 818 543**	1 235 306 258	1 331 333 002	**1 350 818 543**	**1 350 818 543**	**1 350 818 543**	**1 350 818 543**	**1 350 818 543**

4.5 Dynamic Algorithms Based on Optimal Neighborhood Exploration

In this chapter, we introduce a novel local search technique called *optimal neighborhood exploration* to solve the M(W)IS problem in a dynamic setting. This technique, introduced in detail in Section 4.5.1, creates independent subproblems that are solved optimally, leading to improved overall solutions. In Section 4.5.3, we present an extensive experimental evaluation, where we assess the effectiveness of our approach and compare it with other state-of-the-art dynamic solvers. Our algorithm uses the subproblem size parameter that balances running time and solution quality.

References. *This section is based on joint work with Jannick Borowitz and Christian Schulz [32, 30]. Large parts are copied verbatim from the paper or the technical report.*

Complex graphs are useful in a wide range of applications, from technological networks to biological systems like the human brain. These graphs can contain billions of vertices and edges. In practice, the underlying graphs often change over time, i. e., vertices or edges are inserted or deleted as time passes. For example, users sign up or leave a social network, and relations between them may be created or removed over time, or in road networks, new roads are built. Terminology-wise, a problem is said to be *fully dynamic* if the update operations include both insertions *and* deletions of edges and *partially dynamic* if only one type of update operation is allowed. In this context, a problem is called *incremental* if only edge insertions occur, but no deletions, and *decremental* vice versa.

A (fully) dynamic graph algorithm is a data structure that supports edge insertions and edge deletions. It also answers certain queries that are specific to the problem under consideration. A graph-sequence $\mathbf{G} = (G_0, \ldots, G_t)$ for $t \in \mathbb{N}_0$ is an *edit-sequence of graphs* if for all $0 < i \leq t$ there exists exactly one update to the graph G_{i-1} resulting in the graph G_i. This update can be inserting a new edge $e \notin E(G_{i-1})$ such that $G_i = G_{i-1} + e$, or deleting an existing edge $e \in E(G_{i-1})$ such that $G_i = G_{i-1} - e$. Furthermore, a new vertex $v \notin V(G_{i-1})$ can be inserted yielding $G_i = G_{i-1} + v$, or an existing vertex $v \in V(G_{i-1})$ can be deleted, i. e., $G_i = G_{i-1} - v$, Additionally, an update can also change the weight of an existing vertex.

The most studied dynamic problems are graph problems such as connectivity, reachability, shortest paths, or matching (see [116]). However, while there is a large body of theoretical work on efficient dynamic graph algorithms, only recently has experimental work in the area been gaining momentum. For some classical dynamic algorithms, experimental studies have been performed, such as early works on (all

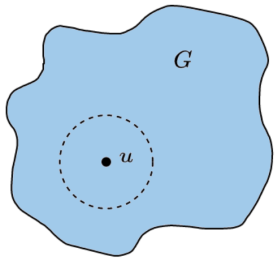


Figure 4.16: Starting from a vertex u, the algorithms find a locally independent subgraph $G[H]$ by exploring the neighborhood up to a certain depth. Then, adjacent vertices from the current solution are added to the subgraph. This subgraph $G[H]$ is solved by a solver of our choice.

pairs) shortest paths [82, 55] or transitive closure [143] and later contributions for fully dynamic graph clustering [58], fully dynamic approximation of betweenness centrality [24] as well as fully dynamic (weighted) matching [9, 119], fully dynamic delta orientations [29] and fully dynamic minimum cuts [120]. However, for other fundamental dynamic graph problems, the theoretical, algorithmic ideas have received very little attention from the practical perspective.

In this section, we tackle the Maximum (Weight) Independent Set problem in a fully dynamic setting. While quite a large amount of engineering work has been devoted to the computation of independent sets/vertex covers in static graphs, the amount of engineering work for the dynamic independent set problem is very limited with only three results by Zheng et al. [225, 226] with their algorithm DgOracleTwo as well as Gao et al. [90] with DyTwoSwap and one result by Bhore et al. [27] that is specialized to independent rectangles. In this dissertation, we extend the set of solvers for the Maximum (Weight) Independent Set problem in the dynamic model by introducing a technique that we call *optimal neighborhood exploration*.

4.5.1 Optimal Neighborhood Exploration

In this section, we explain the core concept of optimal neighborhood exploration. The term refers to optimally exploring a vertex's neighborhood in a graph to find the best possible solution locally. We use this key idea to improve a given (weighted) independent set (or handle updates in a dynamic graph). We first build an induced subgraph $G[H]$ with the following property. For a given independent set $\mathcal{I}$, we build the set of vertices H, such that all vertices $v \in H$ are non-adjacent to independent set vertices outside of H, i. e., $(N_G(H) \backslash H) \cap \mathcal{I} = \emptyset$.

This property enables us to exchange the independent set vertices $\mathcal{I} \cap H$ in G by any independent set of the induced subgraph $G[H]$. Hence, to improve a solution,

we solve the maximum weight/cardinality independent set problem in the induced subgraph $G[H]$. The running time of the algorithm depends crucially on the size of the induced subgraph. Therefore, we limit the subgraph size for our dynamic approach.

Building the Induced Subgraph

We now explain how we find the induced subgraphs for the optimal neighborhood exploration. We start with the following basic idea. Assume a maximal (weight) independent set $\mathcal{I}$ of G is given. We build the subgraph $G[H]$ around a seed vertex $u \in V$. To construct the subgraph, we perform a breadth-first search (BFS) of depth d for a given parameter $d \geq 0$. All vertices reached by the BFS are added to a set H. Afterward, we add all vertices adjacent to vertices in H and the given independent set $\mathcal{I}$. More precisely, we set $H \leftarrow H \cup (N_G(H) \cap \mathcal{I})$. This construction ensures that $(N_G(H) \backslash H) \cap \mathcal{I} = \emptyset$ and therefore we can exchange the independent set vertices of $\mathcal{I}$ in H by any independent set of the induced subgraph $G[H]$.

If we only solve one local problem in an induced subgraph around u, swapping the solution vertices can result in a non-maximal solution. This can happen even if the solution in $G[H]$ has been optimal. Figure 4.17 gives an example of this situation. To deal with this, we add vertices in $N_G(H) \backslash H$ for which all adjacent independent set vertices are in H, i. e., we add the H tight vertices from $N_G(H) \backslash H$ to our subproblem.

Limiting the Subgraph Size

The basic strategy described above can yield large subgraphs $G[H]$ if the expansion of the graph is high. Consequently, we do not only limit the depth d of the BFS but also introduce a second parameter, $\nu_{\max} > 0$, to bound the total number of vertices in H. In order to limit the size of the induced subgraph, we perform a modified breadth-first search. We start the BFS at a seed vertex u and add it into the BFS queue Q. Whenever the BFS reaches a vertex v, we know the current size of H and its extended set via bookkeeping. If adding v to H would imply that the final problem size is larger than $\nu_{\max}$ (when adding respective independent set vertices and H tight vertices of $N(H) \backslash H$), we do not add this vertex to H and also do not continue the BFS at that vertex, i. e., v is not added to the BFS queue Q. The BFS continues until all vertices have been processed, the depth d is reached, or the queue Q is empty.

Pruning Large Degree Vertices (Pinching)

Large-degree vertices in sparse graphs are often not part of optimum solutions. For the weighted case, vertices, where $\omega(N(v))/\omega(v)$ is high, are often not part of optimum

Algorithm 15: DegGreedy Algorithm

Procedure `insertion(`u, v`)`**:**

- $\textsc{Adj}[u] := \textsc{Adj}[u] \cup \{v\}$
- $\textsc{Adj}[v] := \textsc{Adj}[v] \cup \{u\}$
- **if** $u \in IS$ ***and*** $v \in IS$ **then**
  - $\phi_u \leftarrow w(u)/w(N(u))$
  - $\phi_v \leftarrow w(v)/w(N(v))$
  - $\mathcal{R} \leftarrow u$ // $\mathcal{R}$ stores vertex to be removed
  - **if** $\phi_u > \phi_v$ **then**
    - $\mathcal{R} \leftarrow v$
  - **else if** $\phi_u = \phi_v$ **then**
    - $\mathcal{R} \leftarrow \textsc{Random}(u \text{ or } v)$
  - $\mathcal{I} \coloneqq \mathcal{I} \setminus \{\mathcal{R}\}$
  - **for** $w \in N(\mathcal{R})$ **do**
    - **if** $\textsc{free}(w)$ **then**
      - $\mathcal{I} := \mathcal{I} \cup \{w\}$

Procedure `deletion(`u, v`)`**:**

- $\textsc{Adj}[u] := \textsc{Adj}[u] \setminus \{v\}$
- $\textsc{Adj}[v] := \textsc{Adj}[v] \setminus \{u\}$
- **if** $u \notin \mathcal{I}$ **then**
  - **if** $\textsc{free}(u)$ **then**
    - $\mathcal{I} := \mathcal{I} \cup \{u\}$
- **if** $v \notin \mathcal{I}$ **then**
  - **if** $\textsc{free}(v)$ **then**
    - $\mathcal{I} := \mathcal{I} \cup \{v\}$

Procedure `free(`u`)`**:**

- **return** $N(u) \cap \mathcal{I} = \emptyset$

solutions. To address this, we implement a pruning strategy that selectively removes these vertices from the subproblem $G[H]$ before solving it. By excluding these vertices, we reduce the complexity of the subproblem, allowing the solver to concentrate on more promising candidate vertices with lower degrees or lower $\omega(N(v))/\omega(v)$ for the weighted case. This approach not only speeds up the solution process but also helps maintain a more manageable subproblem size, ensuring the solver can operate within practical time limits. This technique is frequently employed in static (heur-

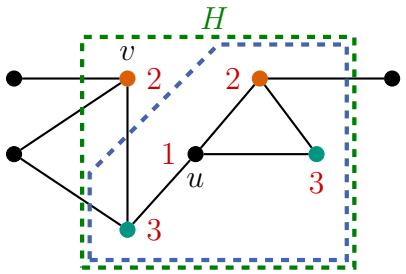


Figure 4.17: Induced subgraph with BFS of depth 1 starting at vertex u. The current solution vertices are orange. The set $H = N[u] \cup \{v\}$. The new optimal solution on the subgraph is green. When changing to this solution, the independent set is not maximal.

istic) algorithms using reduce-and-peel strategies, as M^{2}WIS introduced in Section 4.3. For our approach here, we check the current subgraph $G[H]$ and find the independent set vertex in $G[H]$ with the largest degree. Let the degree of that vertex be denoted by $\Delta_{\mathcal{I},H}$. Then, before solving $G[H]$, we remove vertices from $H \setminus \mathcal{I}$, having a degree larger than $\delta \cdot \Delta_{\mathcal{I},H}$ where $\delta > 1$ is a tuning parameter. The smaller δ, the more this process reduces the subproblem we want to solve. We use $\delta = 1.25$, however, the algorithm is not too sensitive about the precise choice of the parameter.

Solving Local Induced Subgraphs

To solve the local MWIS problem optimally or heuristically, any (exact) solver can be used. We choose the exact solver KaMIS BnR introduced by Lamm et al. [149] since it is state-of-the-art among the exact algorithms and it can solve the MWIS as well as the MIS problem. The KaMIS BnR solver is introduced in detail in Section 3.2.3. In general, the algorithm exhaustively applies a wide range of data reductions to the input instance and runs a sophisticated branch-and-reduce algorithm afterward. It can solve many (large) static instances optimally while outperforming other approaches. Even compared to iterated local-search approaches like HILS, KaMIS BnR can often compete in running time, e. g., for `osm` instances (street networks), as the experiments by Lamm et al. [149] show. We run KaMIS BnR with a time limit t_{BnR}. If the algorithm can not find an optimal solution within the time limit, it returns the best weighted independent set $\mathcal{I}_H$ for $G[H]$ that it has found. Since every independent set for $G[H]$ yields a modified and feasible independent set in G, we update the old solution $\mathcal{I}$ in G if $\mathcal{I}_H$ improves solution quality, i. e., if $\omega(\mathcal{I}_H) > \omega(\mathcal{I} \cap H)$ or in the cardinality case, $|\mathcal{I}_H| > |\mathcal{I} \cap H|$.

Algorithm 16: DynamicONE Algorithm

Procedure `insertion(`u, v`):`

- $\textsc{Adj}[u] := \textsc{Adj}[u] \cup \{v\}$
- $\textsc{Adj}[v] := \textsc{Adj}[v] \cup \{u\}$
- $\textsc{DegGreedy}::\textsc{insertion}(u, v)$
- **if** DegGreedy cannot add a vertex to $\mathcal{I}$ **then**
  - $H \leftarrow \text{BFS-SEARCH}(u, v, d, \nu_{\max})$
  - $\textsc{PruneLargeDegreeVertices}(H)$
  - $H \leftarrow H \cup (N_G(H) \cap \mathcal{I})$
  - **if** $\textsc{SolveIS}(G[H], \mathcal{I} \cap H)$ **then**
    - $\textsc{UpdateSolution}()$

Procedure `deletion(`u, v`):`

- $\textsc{Adj}[u] := \textsc{Adj}[u] \setminus \{v\}$
- $\textsc{Adj}[v] := \textsc{Adj}[v] \setminus \{u\}$
- $\textsc{DegGreedy}::\textsc{deletion}(u, v)$
- **if** DegGreedy cannot add a vertex to $\mathcal{I}$ **then**
  - $H \leftarrow \text{BFS-SEARCH}(u, v, d, \nu_{\max})$
  - $\textsc{PruneLargeDegreeVertices}(H)$
  - $H \leftarrow H \cup (N_G(H) \cap \mathcal{I})$
  - **if** $\textsc{SolveIS}(G[H])$ **then**
    - $\textsc{UpdateSolution}()$

Procedure `PruneLargeDegreeVertices(`H`):`

- **for** each $v \in H$ **do**
  - **if** $d(v) > \delta \cdot \Delta_{\mathcal{I},H}$ and $v \notin \mathcal{I}$ **then**
    - Remove v from H

Procedure `SolveIS(`G_H, $\mathcal{I}_{\mathcal{H}}$`):`

- Solve (weighted) independent set problem in G_H
- **return** true if solution improved, false otherwise

Procedure `UpdateSolution():`

- Update $\mathcal{I}$ in G with solution of $G[H]$

4.5.2 Dynamic Methods

We now leverage the core concept of optimal neighborhood exploration described above to define a fully dynamic algorithm for the MWIS and the MIS problem. Next, we explain our approach to handling various update operations. Before we describe how we use the dynamic optimum neighborhood exploration, we give simple dynamic and fast algorithms. Later, we combine these fast algorithms with our (more expensive) dynamic optimum neighborhood exploration technique to obtain a fast, high-quality, fully dynamic algorithm.

Fast and Simple Greedy Updates

When a new edge $\{u, v\}$ is *inserted*, we first check if the newly created edge induces a conflict with our current solution. Specifically, we verify whether both u and v are currently part of the independent set. If they are, the current solution is no longer independent. To resolve this conflict, we remove the vertex which minimizes $w(v)/w(N(v))$. In case of a tie, we remove a random vertex from the solution. Note that in the cardinality case, this reduces to removing the vertex with the larger degree from the current solution. The intuition here is that we want to remove the vertex that blocks most of the other vertices/weights in the graph. Let the vertex that we removed from the solution be u. Since u is no longer an independent set vertex, we check if its neighbors can be added to the solution. Checking each neighbor takes $O(\Delta)$ time. Thus, the overall update takes time $O(\Delta^2)$. Note that by using proper data structures, i. e., storing and updating for each vertex the number of adjacent independent set vertices, one can reduce the update time to $O(\Delta)$. If an edge $\{u, v\}$ is *deleted* from the graph, we check in $O(\Delta)$ time if u and v can be added to the solution (if u or v are not already part of the solution). We call this fast and simple algorithm DegGreedy and give pseudocode for it in Algorithm 15.

Dynamic Optimum Neighborhood Exploration

We now explain our approach to performing dynamic optimum neighborhood exploration. We integrate this with fast and simple greedy updates to optimize running time. We call this technique *pruning updates*. Initially, we execute the aforementioned fast and simple greedy algorithm. No further action is taken if this algorithm can add a vertex to the solution after updating the graph data structure. However, if it cannot, we can use the more resource-intensive dynamic optimum neighborhood exploration to attempt to improve the solution. Pruning the expensive updates using the fast and simple update algorithm significantly improves the running time of the overall

algorithm. We refer to this algorithm as DynamicONE. The pseudocode for the algorithm is given in Algorithm 16.

Edge Insertion. We construct the subgraph H by performing a breadth-first search using the two parameters d and $\nu_{\max}$. We initialize the BFS with both vertices u and v. We then prune the large-degree vertices of the subgraph (see Section 4.5.1). Afterward, we solve the (weighted) independent set problem in the induced subgraph and update the solution accordingly if we find an improvement.

Edge Deletion. Deleting an edge can not lead to any conflict between independent set vertices. Hence, the current independent set does not need any fixing. After unsuccessfully running the fast and greedy algorithm, we construct the subgraph H based on the two parameters d and $\nu_{\max}$ to obtain the subgraph H. As in the insertion case, we initialize the breadth-first search with both vertices u and v. As before, we prune the large-degree vertices of the subgraph. We then solve the (weighted) independent set problem in the induced subgraph and update the solution accordingly.

Miscellaneous. To save running time, we limit the time of the local solver to 10 s and define *rare updates*. Rare updates only perform expensive updates (solving subgraphs) every $x = 3$ update. In the other two cases, only greedy updates are performed. The parameter x controls a trade-off between running time and solution quality. A smaller value of x will lead to better solution quality but also a higher running time. Other update operations, such as vertex insertion (or weight update) and deletion of a vertex u, can be done by mapping those operations to edge insertion and deletions. However, this can yield many subproblems that need to be solved, for example, if the degree of the inserted vertex is large. One can improve vertex insertion (or weight update) by solving just one subgraph H starting from the seed vertex u with its edges already inserted in the graph. When deleting a vertex u, we initialize the dynamic BFS with the former neighbors $N(u)$ to obtain a subgraph and hence a subproblem.

4.5.3 Experimental Evaluation

In this section, we present the experimental evaluation of our approaches. First, we present experiments on the unweighted instances for the different parameters in our algorithms. Then, we compare our methods with the state-of-the-art. Finally, we evaluate our approaches on weighted instances.

Methodology. We implemented our algorithm using C++17. The code is compiled using g++ version 11.4.0, and full optimizations are turned on (-O3). In the unweighted dynamic case, we compare against the dynamic implementations of Zheng et al. [225] (DgOracleTwo) as well as Gao et al. [90] (DyTwoSwap). These algorithms are also implemented in C++ and compiled with the same compiler and optimization settings as our algorithm. In general, we compare the results of the algorithms after all updates have been performed. We generally run each algorithm/configuration with ten different seeds on Machine 3. We compare our proposed approaches on the Instance Set 5 of dynamic graphs. These consist of 33 instances from various sources.

Instance Set 5 (Dynamic)**.** *This set consists of 33 instances from various sources [15, 53, 83, 116, 123, 145, 155, 177, 180]. Table A.5 in the Appendix summarizes the main properties of this dynamic benchmark set. This set includes several graphs from numeric simulations, road networks, and complex networks/social networks. Our set includes static graphs as well as real dynamic graphs. In the case of instances that have originally been static, we create dynamic instances by starting with an empty graph and inserting all edges in order of their appearance of the static graph. As our algorithms only handle undirected simple graphs, we ignore possible edge directions in the input graphs and remove self-loops and parallel edges. For the cardinality case, we treat all graphs as unweighted graphs.*

Fully Dynamic Maximum Cardinality Independent Set Algorithms

We now present an evaluation of our methods for the maximum cardinality independent set problem. Due to space constraints, our focus is on assessing the impact of the BFS depth and the effectiveness of various algorithmic components in accelerating performance. Finally, we compare our algorithm against the state-of-the-art.

Depth d of BFS. We start with the depth of the BFS, which is denoted by d. Note that this parameter is such that increasing the parameter increases the size of the local problems and thus will (likely) increase the running time of the algorithm but also improve the quality of the solutions. Hence, we look at the performance of the parameter on all instances. The same parameter is used by the respective algorithm on all instances, and there is no instance-specific tuning of the parameter.

In this experiment, we set $\nu_{\max} = 2\,500$ and use $d \in \{0, 1, 2, 3, 4, 6, 8, 10\}$. Figure 4.18 provides a performance profile for solution quality and a running time box plot for the different algorithms as a function of depth d. The algorithms behave as expected, with the quality of the solution improving as the depth of the BFS search

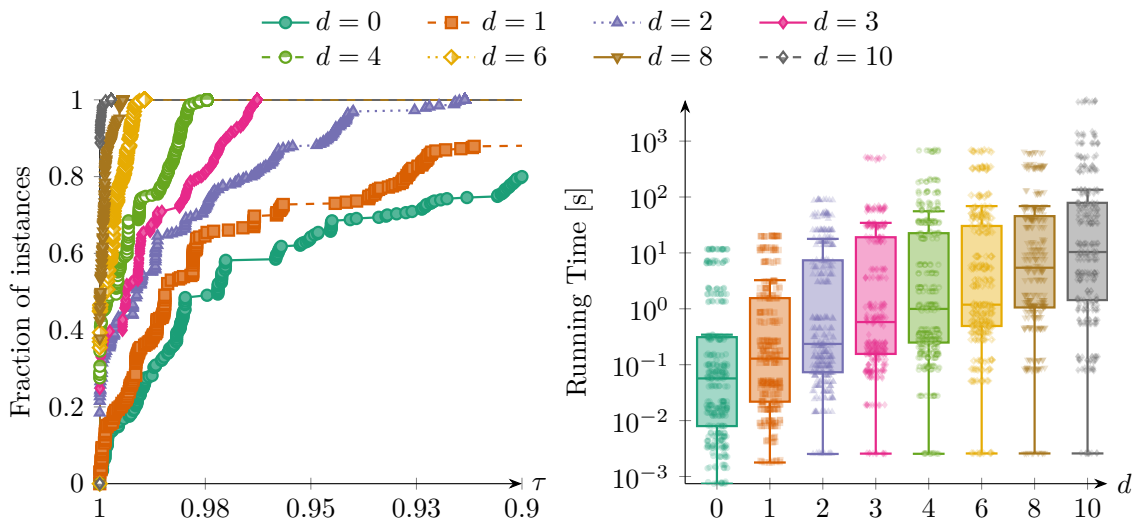


Figure 4.18: Performance profile for solution quality (left) and running times in total for all updates (right) for various depths d with $\nu_{\max} = 2\,500$.

increases. However, this improvement comes with a significant increase in running time. For example, increasing the depth from $d = 0$ to 10 results in an average solution improvement of 6 %, with the largest observed improvement being 22 % on the wing instance. However, this depth also makes the algorithm 181 times slower. The plots further show that initial improvements in quality are more substantial, such as when increasing depth from $d = 0$ to 4. At $d = 4$, our algorithm yields 5 % better solutions than the algorithm using $d = 0$ on average, which is 25 times slower. Thus, in applications, this parameter can effectively control the quality of solutions. However, further techniques (as will follow) are required to speed up the algorithm.

Pruning Updates. Pruned updates help avoid the expensive solving steps of the induced subgraph by performing a simpler and faster local search augmentation step if successful. To evaluate the impact of the pruning updates on the algorithm's performance, we tested all values for d used previously with pruned updates enabled. Overall, pruning updates significantly improve the algorithm's running time. The speed-up does not heavily depend on d, as the pruning step filters out unnecessary updates across all algorithm configurations. On average, pruning reduces the algorithm's running time by 21 %. Additionally, pruning slightly improves solution quality (by a very small margin), which may be attributed to minor variations in the solutions found by the algorithm throughout the various update steps. Hence, we conclude that pruning updates are a crucial component of our algorithm, as they significantly reduce the running time without compromising solution quality.

Rare Updates. With rare updates, we introduce another technique to reduce the number of computationally expensive updates that need to be performed. We tested the impact of rare updates on the algorithm's performance for different depth values d. For this, we run the same configurations as above (without pruning), as well as the configurations from above with rare updates enabled. The results show that rare updates can significantly reduce the algorithm's running time. Here, the impact varies for different values of d. For small values of d, the running time without and with pruning are closer, but as d increases, the benefit of using rare updates becomes more pronounced. For instance, when $d = 1$, the running time improved by 35 % with rare updates (but the size of solutions also decreased by 17 %). As d increases to 10, the running time is improved by a factor of 3.05 while solutions are worse by 0.6 % on average. The small decrease in solution quality is because ignored updates likely take part in a later update as the local subgraphs can be quite large for large values of d. In summary, rare updates prove to be an effective technique for reducing running time, especially for larger values of d, with a small impact on the solution quality. Still, this option may not be useful if solution quality is paramount. Note that, in principle, different values of x yield different trade-offs.

Pinching. Pinching removes vertices from the local subgraph that are unlikely to be in an independent set, reducing the size of the local subgraph. In addition to reducing the size, removing these vertices also makes data reductions more effective. We tested the impact of pinching on the algorithm's performance by varying this parameter, running the same configurations as above (without pruning and rare updates), and enabling pinching on the configurations from above. The results show that pinching can significantly reduce the running time. As expected, for small values of d, pinching does not have a high impact on solution quality or running time. However, for larger values of $d \geq 3$ solution quality is not impacted (< 0.001 % on average) while running time is reduced by 12 %, 15 % 18 %, 22 % and 16 % for $d = 3, 4, 6, 8, 10$ respectively. Hence, we conclude that pinching is a useful technique for reducing the running time of the algorithm without compromising solution quality for values of $d \geq 3$.

We define a strong and fast configuration of our algorithm based on these observations. In both configurations, we set the depth of the BFS to $d = 10$. We use the variant with pinching and pruning, but without rare updates, as our strong configuration DynamicOneStrong, since this resulted in the best solution quality. In our faster configuration, DynamicOneFast, we use pinching, pruning, and rare updates. Futhermore, we decreasing the limit on the local subproblem size to $\nu_{\max} = 200$, which further speeds up this configuration.

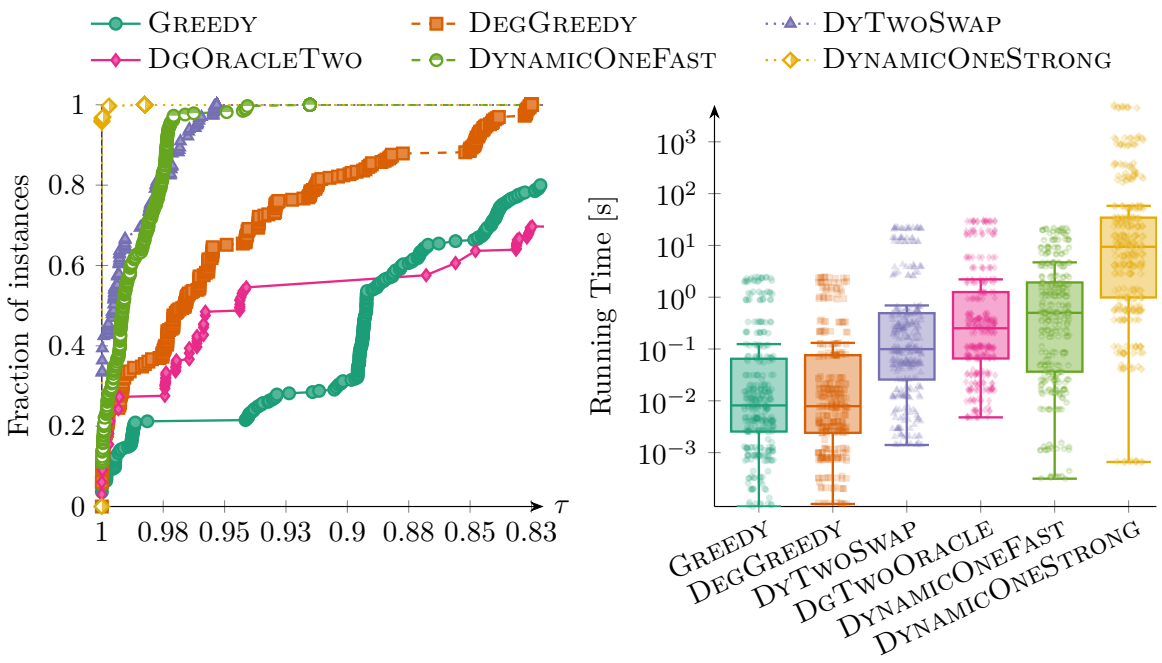


Figure 4.19: State-of-the-art comparison on unweighted instances. Performance profile for solution quality (left) and box plot for total update time (right).

Comparison with the State-of-the-Art

We now compare our algorithms against two state-of-the-art algorithms, DyTwoSwap and DgOracleTwo, as well as two greedy strategies. The Greedy strategy is similar to the DegGreedy algorithm described in Section 4.5.2, with a key difference in how it handles the initial removal of conflicted vertices. Instead of basing this removal on $w(v)/w(N(v))$, it just uses the weights of the vertices $\omega(v)$. Specifically, if an edge is inserted between two independent set vertices, the lighter one is removed from the independent set. In the cardinality case, where all weights are one, this translates to removing a random vertex from the two newly adjacent vertices, followed by a simple augmentation step. This tries to add adjacent vertices of the vertex that we just removed from the independent set vertex. Figure 4.19 gives a performance profile and a running time box plot for the various algorithms under consideration. Detailed per instance results are given in Table B.6. As DgOracleTwo computes worse results than DyTwoSwap and is slower, we only discuss results compared to DyTwoSwap.

First, we see that our strong method DynamicOneStrong outperforms all other algorithms in terms of solution quality. On 31 out of 33 instances, our algorithm computes the best (or equal to the best) result. Improvements vary largely on the type of instances that we consider. Our method performs particularly well on mesh-type networks, which are generally known to be hard instances for the independent set problem. The largest observed improvement is 3.6 % on the whitaker3 instance. The

good performance is due to the fact that the local subgraphs/problems align closely with the algorithm's intuition, specifically forming a ball around the recently inserted or deleted edge, and the fact that this is a powerful local search operation for the value of d we are using. Similarly, our algorithm performs well on road networks. Our algorithm computes a strictly better result than all of the competitors. The largest observed improvement of DynamicOneStrong over DyTwoSwap is 2.4 % on the uk instance. On social networks, DyTwoSwap and our strong algorithm DynamicOneStrong mostly compute the same results. Initially, this looks surprising, however, the authors already show in their paper that on this type of (easy) instances, their algorithm computes solutions very close to optimal or even optimal solutions. We also compare algorithms under consideration to the optimum result, see Table B.6 in the Appendix. We obtained the optimum result after all updates have been performed using the optimum KaMIS BnR; see Section 3.2.3. On the instances that the static branch-and-reduce algorithm could solve within a two-day time limit, our DynamicOneStrong algorithm is at most 0.4 % worse.

Based on the performance profile, the performance of the DynamicOneFast algorithm in terms of solution quality is comparable to, albeit slightly worse than, the DyTwoSwap algorithm. In terms of running time, it is important to note, however, that DynamicOneFast is slower because it employs a more generic approach for updating the independent sets. Moreover, our DynamicOneStrong algorithm, although significantly slower than DyTwoSwap, remains orders of magnitude faster than computing independent sets from scratch. To illustrate this, we relate the running time per update of DynamicOneStrong with the time required by the iterated local search algorithm in the KaMIS framework. This comparison is made on the static counterpart of the final instance after all updates. On average (geometric mean), the time per update for DynamicOneStrong is 1 776 times lower than computing the independent set from scratch using weighted iterated local search on the final instance. Note, however, that this is only a rough estimate as the instances throughout the update sequence increase in size. We sum up our experimental results in the following observations.

Observation 4.5.1: Solution Quality Focus. If solution quality is paramount and the structure of the networks is similar to road networks or mesh-type networks, DynamicOneStrong is the best choice. If even better quality is required in the respective applications, larger values of d and $\nu_{\max}$ may be feasible. Since our algorithm computes the same results as DyTwoSwap on social networks but is much slower, DyTwoSwap is still the way to go for this type of instances.

The simple greedy strategies Greedy and DegGreedy are the fastest algorithms, but they are also worse in terms of solution quality. The DegGreedy algorithm performs significantly better than the simpler Greedy algorithm while having a similar running time. On average, the DegGreedy algorithm computes solutions that are 4.2 % smaller (worse) than DyTwoSwap. However, it is also a factor of 11.7 faster. Surprisingly, the simple DegGreedy strategy already outperforms the DgOracleTwo algorithm significantly. It computes 70.6 % larger independent sets on average while also being a factor of 21 faster. The large improvement mostly stems from the mesh type networks, where the DgOracleTwo algorithm performs particularly badly. In their original paper, the authors did not consider mesh-type networks. Overall, the DegGreedy algorithm is a good choice if the running time is paramount and solution quality is less important.

Observation 4.5.2: Running Time Focus. The simple greedy strategies Greedy and DegGreedy are the fastest methods. These can compute solutions up to 11.7 times faster than the current state-of-the-art heuristics. The DynamicOne approaches are slower because they employ a more generic approach for updating the independent sets. Moreover, our DynamicOneStrong algorithm, although significantly slower than DyTwoSwap, remains orders of magnitude faster than computing independent sets from scratch.

Fully Dynamic Maximum Weight Independent Set Algorithms

Another advantage of our algorithm is that it also works with weighted graphs. We are not aware of any other algorithm that can handle general weighted graphs in the fully dynamic setting (DyTwoSwap and DgOracleTwo can not handle weights). Hence, we compare DynamicOne against the simple greedy strategies incorporating weights as well as optimum results. For this experiment, we run all algorithms on all instances, assigning each vertex a random weight uniformly distributed within the interval [1, 100].

Figure 4.20 gives a performance profile as well as a running time box plot for the various algorithms under consideration. Detailed per instance results can be found in Appendix Table B.7. Overall, the results are in line with the cardinality case. Our DynamicOne algorithms compute much better solutions than the greedy strategies but are also slower. Both of our algorithms outperform Greedy and DegGreedy across all instances. On average, DynamicOneFast and DynamicOneStrong compute solutions that are 8.2 % and 9 % better than DegGreedy, respectively. In

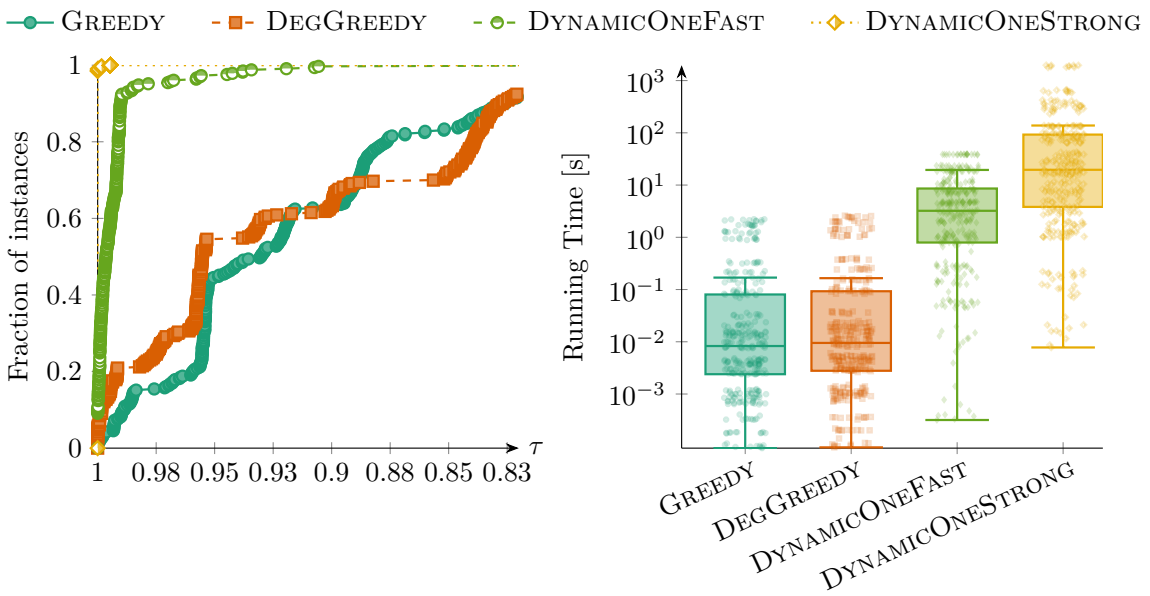


Figure 4.20: Performance profile for solution quality (left) and total update time comparison (right) against greedy algorithms on weighted instances.

contrast to the cardinality case, Greedy and DegGreedy compute mostly similar results. The difference in solution quality is less than 0.1 % on average.

As in the cardinality case, we relate the running time per update of DynamicOne with the time required by the iterated local search algorithm in the KaMIS framework. This comparison is made on the static counterpart of the final instance after all updates. On average (geometric mean), the time per update for DynamicOneStrong and DynamicOneFast is 655 and 6 070 times lower (better), respectively, than computing the independent set from scratch using iterated local search on the final instance. Similar to the cardinality case, we compare algorithms under consideration to the optimum result in Table B.8 in the Appendix.

Observation 4.5.3: Comparison to Optimal Solutions. On the instances that the exact algorithm KaMIS BnR could solve to optimality within a two-day time limit, DynamicOneStrong computes solutions that are at most 0.06 % worse.

4.6 Conclusion

We draw an intermediate conclusion for the chapter on the Maximum Weight Independent Set problem. We start with the conclusion on the reduction rules, followed by the LearnAndReduce method, the m^{2}wis algorithm, the CHILS approach, and end with the conclusion to the optimal neighborhood exploration technique.

Data Reduction Rules

In addition to providing a comprehensive overview of all known data reduction rules for the Maximum Weight Independent Set problem, we introduce new ones and show their use in different solvers. Additionally, we experimentally analyze different reduction orderings. We find that the order of the rules impacts the reduction time and quality, and intuitively ordering the reductions from simple to more complex rules is a good strategy, resulting in stable performance over a wide range of instances.

GNN Guided Preprocessing

We present LearnAndReduce, a preprocessing algorithm for the MWIS problem that combines Graph Neural Networks (GNNs) with a large collection of reduction rules to reduce further and faster than previously possible. The GNNs are trained to predict where we can apply costly reduction rules to speed up the reduction process. Combined, this strikes a good balance between speed and quality at the preprocessing stage.

We introduce a supervised learning dataset for MWIS reductions. The models we use in LearnAndReduce are trained on this dataset. In this work, we only considered the most common GNN architectures and only the application of reduction rule screening. Besides trying more complicated architectures, there are other directions for further work starting from this dataset. One promising direction is to use GNN models to reduce the graph directly. Even though this would no longer be exact preprocessing, it could lead to a powerful heuristic.

Memetic Maximum Weight Independent Set

In this section, we introduced a novel memetic algorithm for the MWIS problem. It repeatedly reduces the graph until a high-quality solution to the MWIS problem is found. After applying exact reductions, we use the best solution computed by the evolutionary procedure on the reduced graph to identify vertices likely to be in an MWIS. These are removed from the graph, which further opens the reduction space and creates the possibility of applying this process repeatedly.

Overall, our two algorithm configurations compute the same or better results than the state-of-the-art. For most instances, these results are probably close to the optimum, and even small improvements in solution quality can yield substantial cost reduction for some applications [60].

For future work, we are interested in an island-based approach to obtain a parallelization of our evolutionary approach. The ExactReduce, and the heuristicRe-

DUCE routine can result in a disconnected reduced graph. We are interested in solving the problem on each of the resulting connected components separately, which also enables new parallelization possibilities. The code of this work is publicly available under https://github.com/KarlsruheMIS.

Concurrent Local Search CHILS

We introduce a new heuristic called CHILS (Concurrent Hybrid Iterated Local Search) that expands on the HILS heuristic. This new heuristic outperforms all known heuristics across a wide range of test instances in a sequential environment. As an added benefit, CHILS can also leverage the power offered by multicore processors. Letting CHILS use all 16 cores available on our test machine significantly improves the solution quality on the hardest instances in our dataset.

The vehicle routing instances by Dong et al. [60] offer a significant challenge for practical MWIS algorithms. Our result marks the third iteration of improvements to this dataset after METAMIS [59] and BSA [113], and yet, we are still far away from optimal solutions on these instances. This is evident by the significant uplift in solution quality by running CHILS in parallel.

There are several directions for future research, including finding efficient data reductions that work on these hard instances or using the clique information in the CHILS heuristic. If the use of clique information leads to improvements, then a natural continuation would be to compute clique covers for other hard instances.

CHILS is based on the proposed metaheuristic Concurrent Difference-Core Heuristic. This metaheuristic could lead to improvements in heuristics for other problems as well. As a metaheuristic, it only requires that solutions can be compared to find the Difference-Core; otherwise, any heuristic method can be used internally. Examples of problems to try include Vertex Cover, Dominating set, Graph Coloring, and Connectivity Augmentation. As an added benefit, the Concurrent Difference-Core Heuristic is trivially parallelizable, which could enable improvements in the parallel setting too.

Dynamic Algorithms

With optimal neighborhood exploration, we present a novel approach for solving the dynamic Maximum Cardinality and Maximum Weight Independent Set problem. Our method efficiently handles large-scale dynamic graphs typical in real-world applications. We developed a local search technique that forms and solves independent subproblems optimally, significantly improving solution quality. Experiments show that our algorithm, featuring a tunable subproblem size parameter, outperforms

existing state-of-the-art solvers. Adjusting this parameter allows balancing between running time and solution quality, with increased values leading to better solutions. In summary, our dynamic optimal neighborhood exploration technique advances the state-of-the-art for computing independent sets in a dynamic setting, providing a practical and scalable solution for large, dynamic graphs. Future work will focus on further optimization, exploring additional applications in other dynamic graph problems, as well as using the technique as a general (parallelizable) local search technique in the static case.

Chapter 5

Maximum (Weight) 2-Packing Set

A 2-packing set for an undirected graph $G = (V, E)$ is defined as a subset $\mathcal{S} \subseteq V$ such that for each pair of vertices $v_1 \neq v_2 \in \mathcal{S}$, the shortest path between v_1 and v_2 has at least length three. Finding a 2-packing set of maximum cardinality or maximum weight is an **NP**-hard problem. We develop new approaches to solve these problems on arbitrary graphs, i. e., without restriction to a special graph class. We do this using new data reductions for the Maximum (Weight) 2-Packing Set (M2PS/MW2PS) problem combined with a reduction to the Maximum (Weight) Independent Set (MIS/MWIS) problem.

We start this chapter with the introduction of our general link-graph framework for the data reduction process in Section 5.1 and then the reduction to the Maximum (Weight) Independent Set problem via a graph transformation in Section 5.2. This reduction to the independent set problem is slightly different for a graph and our link-graph data structure. Afterward, we introduce the problem-specific components of our algorithms for the cardinality variant in Section 5.3 and for the weighted generalization in Section 5.4. In Section 5.5, we conclude this chapter.

References. *This chapter is based on joined work with Jannick Borowitz, Christian Schulz, and Dominik Schweisgut [33], and another joined work with Jannick Borowitz and Christian Schulz [31]. Both papers are currently in submission and large parts are copied verbatim from these papers.*

Our Results. For both the cardinality and the weighted version of the problem, we introduce in total 24 new exact data reduction rules, as well as approaches that use these rules and a reduction to independent set to solve the Maximum (Weight) 2-Packing Set problem. For the cardinality case, we contribute a new exact algorithm red2pack_b&r as well as a heuristic red2pack_heuristic. Our experiments indicate that our algorithms outperform current state-of-the-art algorithms for arbitrary

graphs regarding solution quality and running time. For instance, we can compute optimal solutions for 63 % of our graphs in under a second, whereas the competing method for arbitrary graphs achieves this only for 5 % of the graphs, even with a ten-hour time limit. Furthermore, we outperform a specialized algorithm for planar graphs with respect to solution quality. Lastly, our method solves many large instances that remained unsolved before.

For the weighted case, we evaluate our reduction approach using several exact and heuristic independent set methods. Our experiments show that our new data reduction rules are highly effective and can speed up the process of finding high-quality solutions by multiple orders of magnitude, showing that our algorithms are competitive with state-of-the-art solvers. Moreover, we propose a new method based on the metaheuristic Concurrent Difference Core Heuristic introduced in Section 4.4. Our heuristic excels especially for large graphs, where the transformation uses much memory. Our experiments indicate that our approach can keep up with the state-of-the-art independent set solvers. Additionally, it can find the best solution quality on the biggest instances in our data set, outperforming all other approaches.

Applications. An important application for the Maximum 2-Packing Set problem is in distributed algorithms. In contrast to the Maximum Independent Set problem, which, given a solution vertex, only conflicts with the direct neighborhood, the 2-packing set provides information about a larger area around the vertex. This is important for self-stabilizing algorithms [87, 88, 163, 193, 203, 205, 207]. In particular, computing large 2-packing sets can be used as a subroutine to ensure mutual exclusion of vertices with overlapping neighborhoods. An example is finding a minimal $\{k\}$-dominating function [87] which has various applications itself as presented by Gairing et al. [89]. Bacciu et al. [13] use large k-packing sets to develop a down-sampling approach for graph data. This is particularly useful for deep neural networks. Furthermore, Soto et al. [94] show that the knowledge of the size of a maximum 2-packing set in special graphs can be used for error correcting codes, and Hale et al. [111] indicate that large 2-packing sets can be used to model interference issues for frequency assignment. This can be done by looking at the frequency-constrained co-channel assignment problem. In this application, the vertex set consists of locations of radio transmitters, and two vertices share an edge if the frequencies are mutually perceptible. We want to assign a channel to as many radio transmitters as possible to conserve spectrum and avoid interference. Therefore, vertices assigned to the same channel must have a certain distance. A weighted extension to this application is to assign different importance to the locations modeled by the vertex weights. The

MW2PS is a set of most important locations for radio transmitters, with a distance of three, to avoid interference. The M2PS problem is also of theoretical interest since it is used to compute neighborhood-restricted [≤ 2]-achromatic colorings [42] and the roman domination function [47].

5.1 The Link-Graph

We define the link-graph $\mathcal{G}$ as a tuple $\mathcal{G} = (G, \mathcal{L})$, where $G = (V, E)$ is a graph, and $\mathcal{L}$ is the link set with $\mathcal{L} \subseteq \binom{V}{2} \setminus E$. The link set $\mathcal{L}$ can link vertices in V that are non-adjacent. Every link in $\mathcal{G}$ adds two to the length of a path containing it. We define the *induced link-subgraph* of a set of vertices $V' \subseteq V$ as $\mathcal{G}[V'] = (G[V'], \mathcal{L}[V'])$ with $\mathcal{L}[V'] = \{\{u, v\} \in \mathcal{L} \mid u, v \in V'\}$. The *link-neighborhood* of a vertex $v \in V$ is defined as $L(v) = \{u \in V \mid \{u, v\} \in \mathcal{L} \vee u \in N(N[v])\}$. In the link-graph, the closed 2-neighborhood is defined as $N_2[v] = N[v] \dot\cup L(v)$ and the open 2-neighborhood by $N_2(v) = N_2[v] \setminus \{v\}$. By this definition, for all vertices $u \in L(v)$, the shortest path from u to v is of length two. The *link-degree* of a vertex is defined by the size of its link-neighborhood $\deg_{\mathrm{L}}(v) = |L(v)|$. For a vertex $v \in V$, we define the induced link set $\mathit{link}(v) \coloneqq \{\{x, y\} \in \binom{N(v)}{2} \setminus E\}$. This set contains all links between two non-adjacent vertices with v as a common neighbor. This notation is extended to a set of vertices $V' \subseteq V$ by $\mathit{link}(V') \coloneqq \{\{x, y\} \in \binom{N(V')}{2} \setminus E \mid (N(x) \cap N(y)) \cap V' \neq \emptyset\}$, such that $\mathit{link}(V')$ only contains links connecting vertices in $N(V')$ that have a common neighbor in V'. An important part of the correctness of our data reduction rules introduced in the following sections is extending the set $\mathcal{L}$ by these links for removed vertices.

In this section, we motivate the importance of links and give a first overview of the structure of the reductions. We use the M2PS problem here, but all results are directly transferable to the weighted case. The reduction rules introduced in this chapter follow a similar scheme to those in Chapter 4. We describe this scheme using the example Reduction 5.0.

Reduction 5.0 (Example)

Description of the pattern that can be reduced.

Link Set	*How to extend the link set* $\mathcal{L}$
Reduced Graph	*How to build the reduced link-graph* $\mathcal{G}'$ *from* $\mathcal{G}$ *containing the extended link set*
Offset	*Which size can be added to the offset*
Reconstruction	*How to reconstruct the solution* $\mathcal{S}$ *for the original link-graph given the solution* $\mathcal{S}'$ *on the reduced link-graph* $\mathcal{G}'$

First, we define the pattern that the reduction rule can reduce, followed by details on how to perform the reduction. This information consists of four parts. We start with information on how to extend the link set $\mathcal{L}$. Then, we describe the construction of the reduced link-graph $\mathcal{G}'$ using $\mathcal{G}$ with the already extended link set. Then, the *offset* describes the difference between the size of an M2PS on the reduced graph $\alpha^2(\mathcal{G}')$ and the size of an M2PS on the original graph $\alpha^2(\mathcal{G})$. Lastly, the information on how the solution $\mathcal{S}'$ on the reduced instance can be lifted to a solution $\mathcal{S}$ on the original link-graph is provided. Note that the reduction process with input G starts with the link-graph $\mathcal{G} = (G, \emptyset)$.

In general, our reductions allow us to identify vertices as (1) part of a solution (included) and (2) non-solution vertices (excluded). Before presenting the reductions, we give a lemma that generally describes how to reduce the link-graph to include or exclude a vertex.

Lemma 5.1. *Let $G = (G, E)$ be a graph, $\mathcal{G} = (G, \mathcal{L})$ be a link-graph and $v \in V$.*

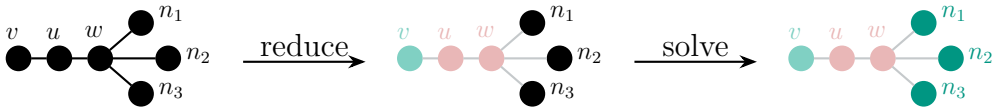
1. *If there is an M2PS not containing v, then we can exclude v and reduce by*

Link Set	$\mathcal{L} = \mathcal{L} \cup link(v)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - v$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

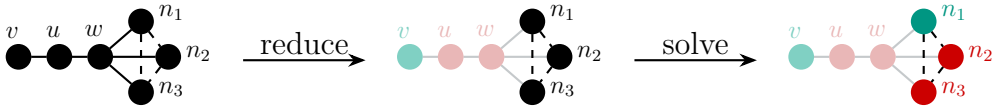
2. *If there is an M2PS containing v, then we can include v and reduce by*

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let $v \in V$ be a vertex in a link-graph $\mathcal{G} = (G, \mathcal{L})$, with $G = (V, E)$. First, assume there is an M2PS not containing v, described in 1. Then, we can safely remove the vertex v from the instance. To maintain the correctness, we have to reduce the graph in the following way. By extending the set $\mathcal{L}$ with links between vertices in the neighborhood of v, i. e., $\mathcal{L} = \mathcal{L} \cup link(v)$, we maintain the 2-neighborhood information for these vertices with $\mathcal{L}$. Adding these links to the link-graph $\mathcal{G}$ does not change the 2-packing set properties of $\mathcal{G}$. The reduced graph $\mathcal{G}'$ is then obtained by removing the vertex v along with all to v incident edges and links. This can be done, since the links $link(v)$ are in $\mathcal{G}$. An M2PS $\mathcal{S}'$ in $\mathcal{G}'$ can then at most contain one vertex from the previous neighbors of v and therefore is also an M2PS in $\mathcal{G}$ and we get $\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}')$.



(a) Including v without extending the link set $\mathcal{L}$, thereby loosing the 2-neighborhood information yielding an invalid solution.



(b) Including v with extending $\mathcal{L}$ by linking all vertices in $\mathcal{L}$ resulting in a valid M2PS.

Figure 5.1: We demonstrate the vital role of the induced links of excluded vertices. With adding these links, we maintain necessary information for the non-reduced neighbors of excluded vertices. Reduced vertices and edges are light gray shaded. Green vertices are included, and red vertices are excluded from the solution.

For the second case, assume there is an M2PS $\mathcal{S}$ containing v, as described in 2. In this case it is safe to include the vertex v in the solution resulting in all vertices in $N_2(v)$ being excluded. To maintain the correctness, we have to apply the same link-graph reduction as in the first case to all vertices in $N_2[v]$. First, note that we do not need to consider vertices in $N(v)$ since all their neighbors are excluded. Applying the excluding reduction to all vertices in $L(v)$ can be combined to extend the link set by $\{\{x, y\} \in \binom{N(N_2[v])}{2} \mid N(x) \cap N(y) \cap N_2[v] \neq \emptyset\}$ and removing all vertices in $N_2[v]$ along with their incident edges and links from $\mathcal{G}$. This, results in the described approach to compute the reduced link-graph $\mathcal{G}'$. An M2PS $\mathcal{S}'$ in $\mathcal{G}'$ can be extended by v to an M2PS in $\mathcal{G}$, i. e., $\mathcal{S} = \mathcal{S}' \cup \{v\}$, and we get $\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$. □

With the reduction, we potentially add links between two vertices that still have a common neighbor in $\mathcal{G}'$, not affected by the reduction. We do not filter out these cases since the correctness is not affected by adding these links and we can save additional computations.

In Figure 5.1a, we perform a reduction that includes the vertex v without extending the link set $\mathcal{L}$. Here, we do not get a valid solution since the information that n_1, n_2, and n_3 belong to the same 2-neighborhood is lost. When we apply the reduction with the additional links in $\mathcal{L}$ for the excluded vertex w, we obtain a valid solution, illustrated in Figure 5.1b.

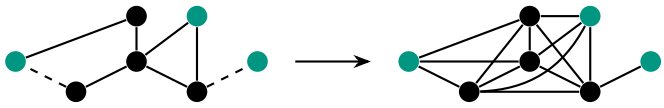


Figure 5.2: The graph $\mathcal{G} = ((V, E), \mathcal{L})$ on the left with links $\mathcal{L}$ marked as dashed lines. This link-graph is transformed to the MIS instance on the right by building the square graph for (V, E) and then adding edges between all linked vertices. The green vertices present an M2PS on the left link-graph and an MIS on the right graph.

5.2 Reduction to Independent Set

In this section, we again use the M2PS problem, however, everything discussed also applies to the weighted case. The maximum 2-packing set in a graph is equivalent to the maximum independent set in the square graph. This equivalence is stated in the following theorem and used in our framework.

Theorem 5.1. *Let $G = (V, E)$ be a given graph and $G^2 = (V, E^2)$ the square graph. Then, an optimal solution to the MIS problem on G^2 is an optimal solution for the M2PS problem on the original graph G [112].*

This simple graph transformation allows us to solve the M2PS problem with well-studied maximum independent set solvers. The basic transformation starts with a given graph $G = (V, E)$ and builds the square graph $G^2 = (V, E^2)$ by connecting all non-adjacent vertices in V with a common neighbor. Starting from a link graph $\mathcal{G} = (G, \mathcal{L})$, we first transform G to the square graph G^2. In the next step, we add edges between two non-adjacent vertices in G^2 that are linked in $\mathcal{G}$. Figure 5.2 illustrates this transformation.

We apply well-studied independent set solvers to find (optimal) solutions on this transformed graph. During the transformation, we increase the number of edges in the graph. Since building the square graph results in more dense instances, the direct transformation becomes quite slow and requires substantial memory.

The reduce&transform Framework

Before transforming the input graph to the square graph, we add a preprocessing routine REDUCE&TRANSFORM. With this routine, we can reduce the input and then transform the reduced instance, resulting in less dense graphs. This can lead to shorter running times and reduced memory consumption. In REDUCE&TRANSFORM, we exhaustively apply our new data reductions to the instance before the transformation to obtain a reduced instance $\mathcal{K}$. We then apply the transformation on $\mathcal{K}$, resulting in a significantly smaller transformed graph. On this graph, an MIS solver is applied to

Algorithm 17: RED2PACK: The REDUCE&TRANSFORM framework combined with independent set solvers.

Data: graph $G = (V, E, \omega)$
Result: M2PS solution
Procedure RED2PACK(G):
  // R ordered list of reduction rules, $\mathcal{K}$ reduced link-graph
  $\mathcal{K}$, $\alpha \leftarrow$ExactReduce(G, R)
  $\mathcal{K}^2 \leftarrow$Transform($\mathcal{K}$)
  $S \leftarrow$ MISSolve($\mathcal{K}^2$)
  return *restore*($\mathcal{K}$, α, S)

obtain an (optimum) solution. In the end, the solution is transformed into an (optimum) solution to the input instance. Combining REDUCE&TRANSFORM with MIS solvers results in the algorithm RED2PACK presented in Algorithm 17.

5.3 Maximum 2-Packing Set

As intruduced in the previous section, the MAXIMUM 2-PACKING SET (M2PS) problem can be reduced to the MAXIMUM INDEPENDENT SET (MIS) problem by a graph transformation to the square graph. Using our new data reductions and the reduction to MIS, we can utilize well-studied independent set sovlers. We introduce these new data reduction rules for the M2PS problem in Section 5.3.1. In Section 5.3.2, we cover the MIS solvers we use, and Section 5.3.3 presents the evaluation of the reductions and the state-of-the-art comparison.

5.3.1 Exact Data Reduction Rules

To reduce the problem size, especially for large instances exact data reductions are very useful tools. Following the scheme introduced in the previous section, we present the data reduction rules for the M2PS problem.

Main Reduction Rules

We first introduce our two main reductions. Afterward, we present more efficient special cases of these rules. An example for the Domination Reduction is illustrated in Figure 5.3a.

Reduction 5.1 (Domination)

Let $u, v \in V$ be vertices such that $N_2[v] \subseteq N_2[u]$, then exclude u.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(u)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - \{u\}$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

Proof. Let $v, u \in V$ and $N_2[v] \subseteq N_2[u]$, yielding v and u are adjacent. Further assume $\mathcal{S}$ is an M2PS in $\mathcal{G}$ containing u. Since $N_2[v] \subseteq N_2[u]$, it holds for all vertices $x \in N_2[v] \setminus \{u\}$ that $x \notin \mathcal{S}$. We define $\mathcal{S}' = (\mathcal{S} \setminus \{u\}) \cup \{v\}$ and it follows that $|\mathcal{S}| = |\mathcal{S}'|$. Moreover, $\mathcal{S}'$ is still a valid 2-packing set since there is no vertex in $N_2[v] \setminus \{v\}$ that is also an element of $\mathcal{S}'$. By construction $\mathcal{S}'$ has the same size and, therefore, is an equivalent solution to M2PS not containing u. Using Lemma 5.1, we get the desired reduction. □

We define a *distance-2-clique* as a set of vertices in $\mathcal{G}$ whose vertices are pairwise connected by a path of length at most 2. A vertex v is *distance-2-simplicial* if the vertices of $N_2(v)$ form a distance-2-clique.

Reduction 5.2 (Distance-2-Clique)

Let $v \in V$ be distance-2-isolated in $\mathcal{G}$, then include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let $v \in V$ be distance-2-isolated and $\mathcal{S} \subseteq V$ be an M2PS in $\mathcal{G}$. Then, at least one vertex $w \in N_2[v]$ is contained in $\mathcal{S}$, otherwise $\mathcal{S}$ is not maximal. It holds $u \notin \mathcal{S}$ for all $u \in N_2[v] \setminus \{w\}$. Additionally, since v is distance-2-isolated $N_2[v] \subseteq N_2[w]$. Therefore, a new solution $\mathcal{S}' = (\mathcal{S} \setminus \{w\}) \cup \{v\}$ of the same size containing v can be constructed. This way, there is always an equivalent or better solution when including v, and therefore, the vertex v is in some M2PS of G. Reducing the graph by including v results in $\alpha^2(\mathcal{G}) = 1 + \alpha^2(\mathcal{G}[V \setminus N_2[v]])$. Using Lemma 5.1, we get the desired reduction. □

These two introduced reductions require knowledge about the 2-neighborhood as a prerequisite. The 2-neighborhood of a vertex v can become quite large, and verifying these conditions is computationally expensive. Hence, we have sought out different

special cases where it suffices to consider only the direct neighborhood and constraints on the link-degree, which we maintain.

Efficient Special Case Reduction Rules

The following lemma helps us to show that these special cases are instances of the more general case.

Lemma 5.1. *Let $u, v \in V$ be neighbors in $\mathcal{G}$ with $N[v] \subseteq N[u]$ such that $\deg_L(v) + \deg(v) \leq \deg(u)$. Then, all link-neighbors of the vertex v, are also neighbors of the vertex u, i. e., $L(v) = N[u] \setminus N[v]$.*

Proof. Let $u, v \in V$ be neighbors with $N[v] \subseteq N[u]$ such that $\deg_L(v) + \deg(v) \leq \deg(u)$. By definition of the link-neighborhood and since u and v are neighbors, we know that $N[u] \setminus N[v] \subseteq L(v)$. Therefore, it holds that $\deg(u) + 1 - (\deg(v) + 1) \leq \deg_L(v)$ which is equivalent to $\deg(u) \leq \deg_L(v) + \deg(v)$. Due to the assumption that $\deg_L(v) + \deg(v) \leq \deg(u)$ it follows equality, i. e., $\deg_L(v) + \deg(v) = \deg(u)$. Because of $N[u] \subseteq N[v] \cup L(v)$ the two sets $N[u] \setminus N[v]$ and $L(v)$ must be equal. Using Lemma 5.1, we get the desired reduction. □

Note that, during the reduction process, all direct neighbors of a vertex v can be removed, resulting in $\deg(v) = 0$. However, there can still be links to the vertex v remaining, yielding $\deg_L(v) > 0$. This case is considered in the following reduction.

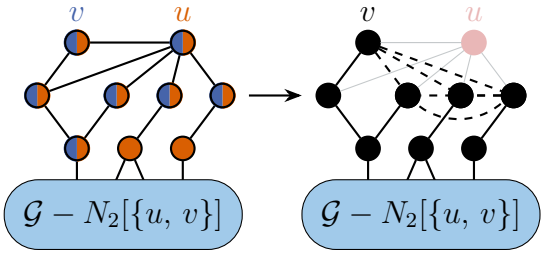
Reduction 5.3 (Degree Zero Reduction)
Let $v \in V$ be a degree zero vertex with $\deg_L(v) \leq 1$, then include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

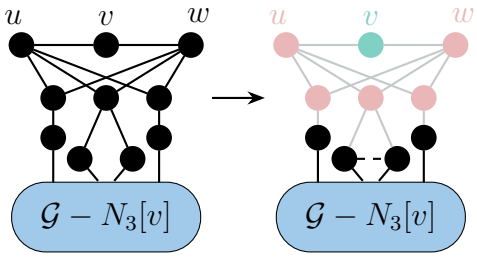
Proof. Let $v \in V$ be a vertex with $\deg(v) = 0$ and $\deg_L(v) \leq 1$. For the case of $\deg_L(v) = 0$, there is no vertex in the 2-neighborhood of v. Therefore, v can be included in the solution. In the case of $\deg_L(v) = 1$, let the link-neighbor of v be $u \in L(v)$. If u is not in the solution, the vertex v can be safely included. In the case of u being part of an M2PS $\mathcal{S}$ on the original instance, we show there always exists another M2PS $\tilde{\mathcal{S}}$ containing v. We can create $\tilde{\mathcal{S}}$ by swapping the vertex u for v, i. e., $\tilde{\mathcal{S}} = \mathcal{S} \setminus \{u\} \cup \{v\}$. It follows that $\tilde{\mathcal{S}} \cup N_2[v] = \{v\}$, since u is the only 2-neighbor of v in G. Furthermore, the solution is the same size and is, therefore, also an M2PS. We showed that an M2PS containing v always exists, and therefore, the vertex can be included in the solution. We get the described reduction using Lemma 5.1. □

Figure 5.3: Original link-graphs on the left are reduced to the link-graphs on the right. Gray edges, and green (included) or red (excluded) vertices are removed from $\mathcal{G}$. The added links are marked with dashed lines.

(a) Reduction 5.1 (Domination): $N_2[u]$ is marked orange and $N_2[v]$ blue. The vertex u dominates v and can be excluded.



(b) Reduction 5.10 (Twin): The neighbors $u, w \in V$ of v are twins. We can include v into the solution and exclude $N_2(v)$.



Reduction 5.4 (Degree Zero Triangle)

Let $v \in V$ be a degree zero vertex. Furthermore, let $\deg_L(v) = 2$ with link-neighbors $L(v) = \{u, w\}$ that are also adjacent or linked, that is $u \in N_2[w]$, then include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let $v \in V$ be a vertex of $\deg(v) = 0$ and $\{u, w\} = L(v)$ its link-neighbors adjacent or linked, i. e., $u \in N_2[w]$. Vertices u and w dominate vertex v and can therefore be excluded by Reduction 5.1. Now, Reduction 5.3 is applicable, and vertex v can be included in the solution. With Lemma 5.1, we get the graph and link set described. □

Reduction 5.5 (Degree One)

Let $v \in V$ be a degree one vertex with $N(v) = \{u\}$. Furthermore, let $\deg_L(v) \leq \deg(u) - 1$, then include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

.

Proof. Let $u, v \in V$ be vertices such that $\deg(v) = 1$ and $N(v) = \{u\}$ and additionally $\deg_L(v) \leq \deg(u) - 1$. Here, we can apply Lemma 5.1, and it holds that $N_2[v] = N[u]$.

Therefore, this represents a special case of the distance-2-clique, and Reduction 5.2 can be applied. □

Reduction 5.6 (Degree Two V-Shape)

Let $v \in V$ be a vertex of $\deg(v) = 2$ *with* $N(v) = \{u, w\}$ *and* $\deg_L(v) = 0$*, then include* v*.*

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let the above-stated assumptions hold. The closed 2-neighborhood $N_2[v]$ only contains the vertices v, and its neighbors u and w. Furthermore, since $N(v) = \{u, w\}$ we know $u \in N_2[w]$ and $w \in N_2[u]$. Therefore, $N_2[v] \subseteq N_2[u]$ and $N_2[v] \subseteq N_2[w]$ the vertices w and u can be excluded by Reduction 5.1. Since $|L(v)| = 0$, vertex v can safely be included. □

Reduction 5.7 (Degree Two Triangle)

Let $v \in V$ be a vertex of $\deg(v) = 2$ *with* $N(v) = \{u, w\}$ *and* $\deg(u) = \deg(w) = 2$*. Furthermore, let* $\deg_L(v) = 0$*, then include* v*.*

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let the vertices $v, u, w \in V$ all have degree two and $N(v) = \{u, w\}$ and $\deg_L(v) = 0$. In this case, the vertices u, v, and w form a triangle. Since $N_2[v] \subseteq N_2[u]$ and $N_2[v] \subseteq N_2[w]$ the vertices w and u can be excluded by Reduction 5.1. Since $|L(v)| = 0$, vertex v can safely be included. □

Reduction 5.8 (Degree Two 4-Cycle)

Let $u, v, w, x \in V$ be vertices with $\deg(v) = \deg(u) = \deg(w) = 2$*, $N(v) = \{u, w\}$ and $L(v) = \{x\}$. Furthermore, let $x \in N(u)$ and $x \in N(w)$. Then, the vertices build a 4-cycle, and v can be included.*

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Assuming the above-stated assumptions such that the vertices u, v, w and the one link-neighbor $x \in L(v)$ form a 4-cycle. It holds that $N_2[v] = \{u, v, w, x\} \subseteq N_2[u]$, therefore, we can exclude the vertex u by Reduction 5.1. Similarly, we can exclude vertex w. Assume x is part of an M2PS $\mathcal{S}$. Then, we can create a new solution $\mathcal{S}' = \mathcal{S} \setminus \{x\} \cup \{v\}$ of same size. This way, we always find an M2PS including v. By applying Lemma 5.1, we get the desired reduction. □

Reduction 5.9 (Fast Domination)

Let $u, v \in V$ be vertices such that $N[v] \subseteq N[u]$ and $\deg_L(v) + \deg(v) \leq \deg(u)$, then exclude u.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(u)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - u$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

Proof. Let $u, v \in V$ and $N[v] \subseteq N[u]$ with $\deg_L(v) + \deg(v) \leq \deg(u)$. Since $N[v] \subseteq N[u]$, u and v are adjacent and we can use Lemma 5.1. Therefore, it holds $L(v) = N[u] \setminus N[v]$ and $N_2[v] \subseteq N_2[u]$ resulting in Reduction 5.1 being applicable. □

Figure 5.3 gives an example for the Twin Reduction.

Reduction 5.10 (Twin)

Let $v \in V$ be a vertex with $\deg(v) = 2$ and $u, w \in V$ be its neighbors with $N(u) = N(w)$. Furthermore, let $\deg_L(v) \leq \deg(u) - 1$. Then, u and w are twins, and v is included.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha^2(\mathcal{G}) = \alpha^2(\mathcal{G}') + 1$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let $v \in V$ be a vertex with $\deg(v) = 2$ and $u, w \in V$ be its neighbors with $N(u) = N(w)$. Since $u \notin N(u) = N(w)$, the vertices u and w are non-adjacent. We now show that all link-neighbors of v are adjacent to the vertex u and w. By definition of the link-neighborhood and since $w \notin N(u)$, we know $N(u) \setminus \{v\} \subseteq L(v)$. This yields $\deg(u) - 1 = |N(u) \setminus \{v\}| \leq |L(v)| = \deg_L(v)$. The additional assumption $\deg_L(v) \leq \deg(u) - 1$ ensures that the sets are equal and therefore $\mathcal{G}[N_2[v]]$ forms a distance-2-clique. Consequently, this reduction is a special case of Reduction 5.2. □

5.3.2 Independent Set Solvers for the 2-Packing Set

To solve the Maximum Independent Set problem on the transformed graph, we use the state-of-the-art solver KaMIS BnR by Lamm et al. [149], introduced in detail in Section 3.2.3. However, it is also possible to integrate any other exact solver. Note that we did not choose the branch and reduce solver for the unweighted problem [148] since, with it, we are restricted to smaller graphs. While our focus is on an optimal solution, we also combine our reduce&transform approach with OnlineMIS [52], resulting in our heuristic red2pack_heuristic. For a brief description of the main components of the solver OnlineMIS by Dahlum et al. [52] we refer the reader to Section 3.2.3.

5.3.3 Experimental Evaluation

We now present the experimental evaluation of our algorithms for the Maximum 2-Packing Set problem. We first investigate the impact of our data reduction rules. Then, we compare our algorithms against the state-of-the-art.

Methodology. We implemented our algorithm using C++17. The code is compiled using g++ version 12.2 and full optimizations turned on (-O3). Every experiment is run on Machine 1 repeated four times with different random seeds. We report geometric mean values unless mentioned otherwise. The time limit for all algorithms is set to ten hours. If a solver exceeds a memory threshold of 100 GB during execution, we stop the solver and mark this with m.o. If the algorithm terminated due to the time limit, we mark it with t.o. in the results. In both cases, we report the best solution found until this point.

We always report the best solution found until the time limit is reached and the time it takes to find this solution. If the time limit is reached, this can result in reported solutions from exact solvers not being optimal and smaller than reported heuristic solutions.

Overview/Competing Algorithms. We perform a wide range of experiments. First, we perform experiments to investigate the influence of the data reduction rules in Section 5.3.3. Therefore, we define three reduction list configurations for our reductions. The first is called 2pack and does not include any of our proposed reductions. Then, in main, we only use Reduction List 5.1 containing the clique and domination reduction. In preliminary experiments, this order of reductions worked best.

Reduction List 5.1 (main)**.** $R_{5.1} \coloneqq [5.2, 5.1]$

For the last variant FAST, we apply the full set of all special case reductions as well as the Domination and Clique Reduction. The order of reductions for this variant is given in Reduction List 5.2.

Reduction List 5.2 (FAST)**.** $R_{5.2} \coloneqq$ [5.3, 5.4, 5.5, 5.7, 5.8, 5.6, 5.10, 5.9, 5.1, 5.2]

For this ordering, we put our special case reductions before the Domination and Clique Reduction in FAST to test efficient reductions first. They may reduce a significant number of vertices before we apply more computationally costly reductions. Note that we did not experiment with different orderings for the reductions as done in Section 4.1.3 for the MWIS problem. Since an intuitive ordering worked best in this experiment and small changes did not significantly affect the solution quality and running time, we also chose a similar approach for ordering Reduction List 5.2.

We now compare our methods against the state-of-the-art for the problem in Section 5.3.3. In particular, we compare against the genetic algorithm GEN2PACK by Trejo-Sánchez et al. [202] as well as the APX-2P+IMP2P algorithm by Trejo-Sánchez et al. [206] which only works for planar graphs. We use two configurations of APX-2P+IMP2P. The configurations differ in the parameter h, which specifies the number of vertices in the subgraphs. With increasing h, the solution quality improves, but the slower the algorithm performs. We chose the default configuration with $h = 50$ and $h = 100$ to improve the solution and give a fairer comparison with our ten-hour time limit. We could not perform experiments with MAXIMUM-2-PACK-CACTUS [78] since the code is not available [77] and the data in the paper itself is presented such that a direct comparison is not possible.

Data Sets. We use Instance Set 6, consisting of small and large social networks (`small/large social`), which have not been solvable for this problem before, cactus (`cactur`) and Erdős–Rényi (`erdos`) graphs to compare against GEN2PACK, and planar graphs (`planar`) used for the comparison with APX-2P+IMP2P.

Instance Set 6 (2-Packing)**.** *This set contains a wide range of 60 instances from different sources. First, we use a set of social networks in our benchmark, which are also part of Instance Set 1. More precisely, we use forty social networks from [16] and [184]. We divided this class into two subsets, each containing twenty graphs, based on the number of vertices: graphs with fewer than 50,000 vertices (`small social`) and those with more (`large social`). We only added instances to the benchmark where the number of edges did not exceed the 32-bit limit when constructing the square graph. Note that this only happened for large graphs and not for any instances used in previous experiments for the maximum 2-packing set problem. The set furthermore includes*

	2PACK		MAIN		FAST	
	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$
`planar`	100	212.11	99.91	211.67	99.91	211.67
`social (s)`	100	3 154.00	14.48	70.25	14.48	70.25
`social (l)`	100	4 327.38	17.12	274.56	17.12	274.58
overall	100	2 564.50	43.84	185.49	43.84	185.50

Table 5.1: Effect of graph transformation: Arithmetic mean of percentage of number of vertices $\tilde{n} = 100 \cdot n(\mathcal{K}^2)/n(G^2)$ and edges $\tilde{m} = 100 \cdot m(\mathcal{K}^2)/m(G^2)$, where $\mathcal{K}$ is the reduced and transformed link-graph computed with the different configurations. Since 2PACK does not apply reductions, this column shows the mean values for the square graphs G^2. Detailed results are presented in Table B.9 in the Appendix.

20 cactus graphs (`cactus`) from [78] with at least 37 vertices, as well as a random selection of 20 of the 1,050 Erdős–Rényi graphs (`erdos`) from [202]. Additionally, this set includes planar graphs (`planar`) from [206]. For an overview of the graph properties, see Table A.6 in the Appendix.

Impact of Data Reductions

We first investigate the effectiveness of the data reductions. To do so, we use the three reduction configurations for the exact algorithm RED2PACK_B&R. To evaluate the effectiveness, we do not investigate the influence on `erdos` and `cactus` graphs, as they are already very small. We compare the impact on the size of the reduced transformed graphs as well as solution quality and running time. Details are presented in Tables B.9 and B.10. We summarize these results in Table 5.1 and Figure 5.4.

First, we look at the effectiveness of our reductions on the size of the reduced transformed graph $\mathcal{K}^2$. When applying the graph transformation on the original input, i.e. without applying any reduction (2PACK), the resulting instance has the same amount of vertices and on average $25.65\,m(G)$ edges, whereas MAIN and FAST both yield on average $0.44\,n(G)$ vertices and $1.86\,m(G)$ edges. Thus, our data reductions help to decrease the size of the transformed graph by more than a factor of ten on average. As expected, the approaches MAIN and FAST compute overall the same reduced link-graph sizes. However, on five large social instances, FAST reduces further, but only with a very small difference. Similar to the independent set problem, also here, our reduction rules are not working well on planar graphs since they have a mesh-like structure. The number of vertices in the reduced instances is only reduced

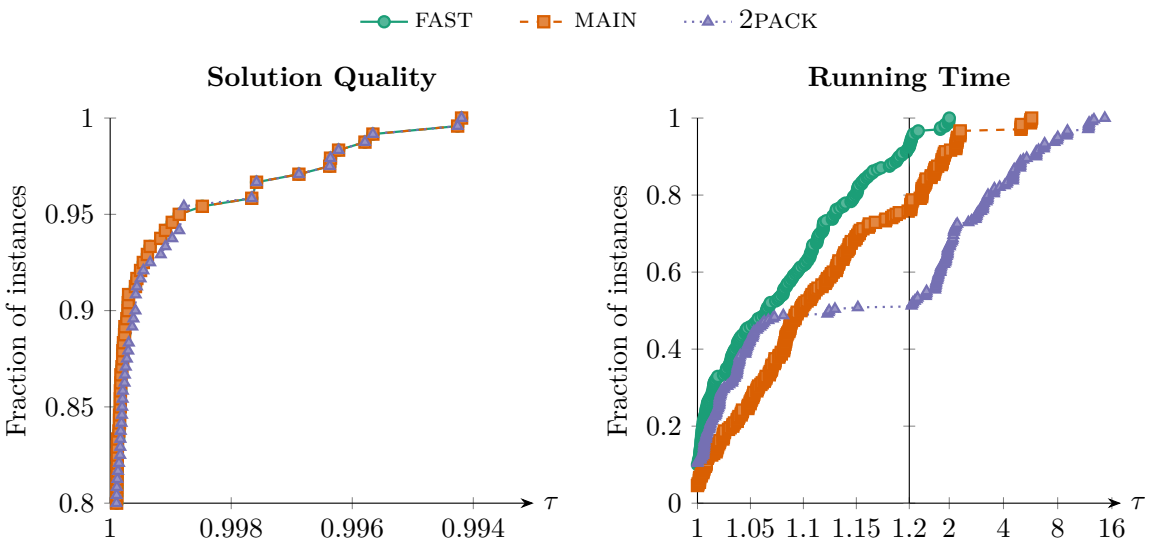


Figure 5.4: Solution quality (left) and running time (right) for our exact algorithm RED2PACK_B&R evaluated on `planar` and `social` graphs comparing our different reduction variants. Since `cactus` and `erdos` graphs are too small to show differences, we omitted these classes.

by 0.1% compared to the original graph. Altogether, we are able to fully reduce 15 out of 60 instances and thereby solve them optimally solely by our data reductions. Hence, we conclude that the reductions are highly effective in reducing the graph size and especially reduce the size for social networks.

Our variant FAST performs slightly better than the other variants regarding solution quality. However, especially on the `small social` and `planar`, a difference can hardly be seen. When considering `large social` graphs, however, we can find no instance on which 2PACK outperforms MAIN or FAST. Overall, we achieve an improvement through FAST on this graph class of 0.05 % compared to 2PACK and MAIN. The instance with the largest difference in solution quality is road_usa. Here, FAST achieves an improvement of 0.93 % over the other two strategies. On the instance amazon-2008 FAST performs worse compared to MAIN. On this, the solution quality of MAIN is improved by 0.10 % compared to the solution of FAST. For all of the 6 instances, on which FAST was outperformed by 2PACK the improvement over FAST is always smaller than 0.01 %.

Figure 5.4 also shows that using our different data reduction rules as a preprocessing step (MAIN and FAST) especially improves the running time compared to 2PACK. Here, we see that, in general, our reductions are improving the performance, and our approach FAST works best. In the detailed results in Table B.10 in the Appendix, we see that especially for `large social` graphs FAST yields a speedup of 2.7

compared to 2PACK. On `planar` graphs, where our reductions are not very effective in reducing the initial input size, the performance is very similar for all our variants.

Observation 5.3.1: Reduction Configurations. The solution quality does not change much between the three different configurations. However, the time spent during this process differs. With our variant FAST we find the best performence, showing that it is important to apply fast, special case reductions before the more general ones. The naive transformation approach 2PACK is up to 16 times slower compared to using our reductions.

Due to the good performance, we choose the FAST reduction variant in the following state-of-the-art comparisons for both RED2PACK_B&R and RED2PACK_HEURISTIC.

Comparison with State-of-the-Art

We compare our algorithms RED2PACK_B&R and RED2PACK_HEURISTIC using the best reduction variant FAST. The comparison includes the state-of-the-art algorithms GEN2PACK by Trejo-Sánchez et al. [202] as well as two configurations of APX-2P+IMP2P by Trejo-Sánchez et al. [206].

GEN2PACK: For the comparison with GEN2PACK, we use `cactus` graphs and Erdös–Rényi (`erdos`) networks, i. e., the instances used in their paper, as GEN2PACK is not able to solve any of the other, larger graphs within the given time limit. This can be explained by the initial computations containing matrix multiplication used in GEN2PACK. This does not finish within the ten-hour limit, so the algorithm could not compute any solution. Detailed per instance results for this comparison can be found in Table B.12 in the Appendix.

In Figure 5.5, we give performance profiles for running time and solution quality. In Table 5.2, we present the geometric mean running times and solution qualities for these results. Our algorithm RED2PACK_B&R as well as RED2PACK_HEURISTIC find overall the optimal solution in the classes `cactus` and `erdos` within a few milliseconds. Our algorithms dominate GEN2PACK in terms of both solution quality as well as running time. Especially the differences in running time are very large. On all graphs, our two algorithms are multiple orders of magnitude faster than GEN2PACK. It can only find optimal solutions for 6 out of these 40 graphs, see Table B.12. On these two graph classes *both* of our algorithms always compute the optimum solution, which results in an average solution quality improvement of more than 20% and a speedup of more than 10^5. Among all instances under consideration, on Erdos37-2 GEN2PACK needed the least time to compute an optimum solution. For this instance, we achieve with RED2PACK_B&R a speedup of more than

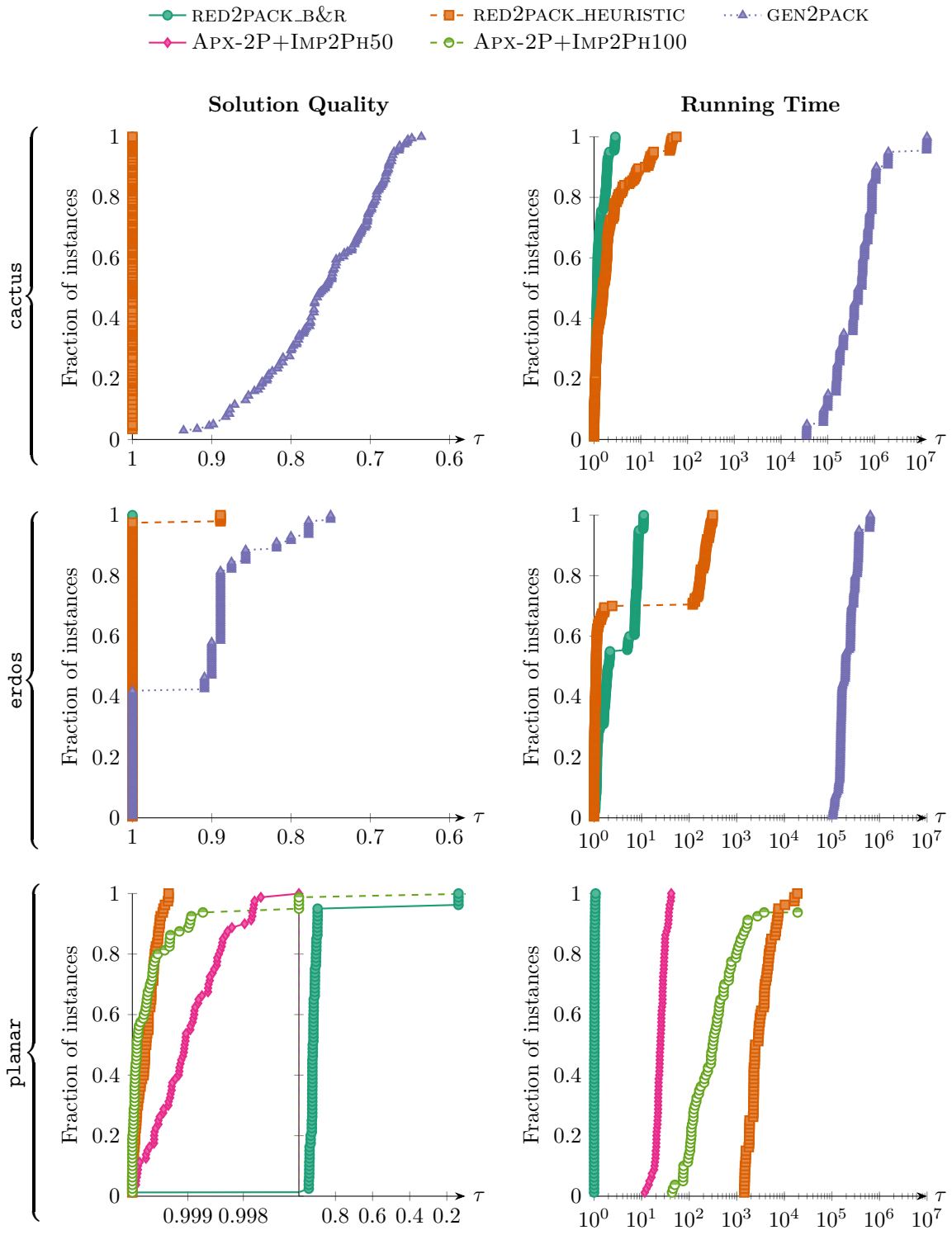


Figure 5.5: State-of-the-art comparison on solution quality (left) and running time (right) for different graph classes. For `planar` graphs (bottom), GEN2PACK is not able to solve the instances. APX-2P+IMP2P with $h = 50$ and $h = 100$ is restricted to `planar` graphs.

	gen2pack		Apx-2P+Imp2P ($h = 50$)		red2pack b&r		red2pack heuristic	
Class	$\|S\|$	t	$\|S\|$	t	$\|S\|$	t	$\|S\|$	t
`cactus`	104	1 384 007.11		-	**137**	**4.26**	**137**	7.30
`erdos`	8	21 679.67		-	**9**	**0.31**	**9**	0.53
`planar`		-	110 009	255.04	92 135	10.41	**110 095**	**31 706.65**
`social (s)`		-		-	**159**	**11.32**	**159**	13.53
`social (l)`		-		-	30 066	6 442.49	**30 756**	**26 377.56**

Table 5.2: Summary comparison of state-of-the-art results. Geometric mean over different graph classes of solution size $|S|$ and time t (in seconds) to find it. **Best** results are emphasized in bold. Apx-2P+Imp2P with $h = 100$ cannot solve all planar instances within the experimental setup. Detailed results are presented in Tables B.11 to B.13 in the Appendix.

300 000 and more than 350 000 with red2pack_heuristic. The instance on which gen2pack needs the most time is `cac1000`. On this instance, red2pack_b&r and red2pack_heuristic again have similar speedups in the range of 10^5 over gen2pack and an improvement in solution quality of roughly 32%. Considering the overall data set, our approach red2pack_b&r can solve 63 out of 100 graphs to optimality within less than one second and 71 within the ten hour time limit and 100 GB restriction. For instances that we could not solve to optimality due to experimental restrictions, we give the solution found until this point, see Tables B.11 to B.13 in the Appendix. In Table B.11, we present results to compare red2pack_b&r and red2pack_heuristic on `social` graphs, which are not solvable with the other approaches. On these instances, we can achieve an average improvement in solution quality of around 1 % with red2pack_heuristic compared to red2pack_b&r. The heuristic can find better solutions, especially for large graphs, where the exact solver meets the memory limit.

Apx-2P+Imp2P: Detailed results for these experiments are given in Table B.13 and Figure 5.5 (bottom). The method gen2pack for general graphs cannot solve any of these instances, so we omitted it in the corresponding tables and performance profiles. Since our reductions do not perform well on planar graphs, we cannot solve them optimally with red2pack_b&r and exceed the memory threshold quite fast. This results in the fast running times and bad solution quality reported. Overall, the solution quality we achieve with red2pack_b&r for the planar graphs is 84 % of the solution quality that Apx-2P+Imp2P ($h = 50$) computes, but we only use 4 % of its

time needed. With RED2PACK_HEURISTIC, on the other hand, we outperform APX-2P+IMP2P ($h = 50$) on all but one instance regarding solution quality, see Table B.13. We achieve an average solution quality that is on par in terms of solution quality. Note that the authors experimentally show in [206] that the 2-packing set computed by APX-2P+IMP2P are already at least 99 % of the optimum solution. However, on those instances, RED2PACK_HEURISTIC needs roughly two orders of magnitude more running time than APX-2P+IMP2P ($h = 50$). APX-2P+IMP2P with $h = 100$ compared to $h = 50$ can improve all but one solution. However, the running time increase is up to multiple orders of magnitude, and some instances were not solved within the ten-hour time limit. RED2PACK_HEURISTIC can find better solutions than APX-2P+IMP2P ($h = 100$) on 9 out of 20 instances, while the differences in solution quality are very small. We summarize our main findings in Observation 5.3.2.

Observation 5.3.2: State-of-the-Art Comparison. On all instances, we are able to outperform GEN2PACK, the state-of-the-art method for arbitrary graphs in both solution quality and running time by multiple orders of magnitude. Moreover, we can solve a wide range of instances to optimality that previously have been unsolvable. When comparing with an algorithm specialized on planar graphs, we presented two options: one that is by more than a factor of 24 times faster with lower solution quality and one that is on par in terms of solution quality, but slower, compared to both configurations of the state-of-the-art *specialized* solver for planar graphs.

5.4 Maximum Weight 2-Packing Set

This section focuses on the weighted generalization of the 2-packing set. Instead of the maximum cardinality, we are now looking for a 2-packing set of maximum weight. For this generalization, we introduce new data reduction rules in Section 5.4.1. These rules also utilize the concept of links introduced in Section 5.1 and can be combined with the REDUCE&TRANSFORM routine described in Section 5.2. In Section 5.4.2, we present our new reduce and peel approach based on the metaheuristic CONCURRENT DIFFERENCE CORE HEURISTIC introduced in Section 4.4. Section 5.4.3 evaluates the preprocessing using several state-of-the-art independent set solvers equipped with REDUCE&TRANSFORM. Furthermore, we compare their performance with our new method.

5.4.1 Exact Data Reduction Rules

After introducing basic concepts, we present the 13 new data reduction rules for the MW2PS problem. Then, we present more details on the implementation and very efficient data reduction rules, which especially exploit the structure of a 2-packing set. The reduction rules presented here are derived from data reduction rules for the Maximum Weight Independent Set problem [107, 149], discussed in Section 4.1.

The presented reduction rules are applied exhaustively in a predefined order in our preprocessing. Whenever a reduction rule is applied successfully, we check the previous rules again on the parts of the graph that changed. After applying a rule, we refer to the resulting instance as the *reduced* instance. After this step, no more reduction rules are applicable to the instance and we pass it to the next step of the algorithm. We refer to this instance as $\mathcal{K}$.

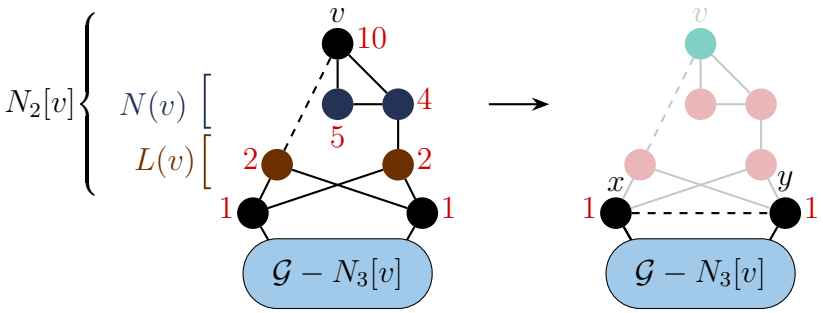
All rules are introduced following the scheme as used in [105]. We first define the pattern in the graph that the corresponding rule can reduce. Then, we present the details of the construction of the reduced instance $\mathcal{G}'$. The reduced link-graph is constructed in two steps. First, we extend the *link set* $\mathcal{L}$ of the link-graph $\mathcal{G}$. In this step, for a given set of vertices $X \subset V$ that is removed from the link-graph in the corresponding reduction, we add links between two vertices $u, v \in V \setminus X$, if the shortest path between u and v is of length two in $\mathcal{G}$ however, it is longer in $\mathcal{G} - X$, i. e., all shortest paths between u and v included vertices in X. We have to add these links to maintain this necessary 2-neighborhood information linking the vertices u and v in the reduced link-graph.

Then, $\mathcal{G}'$ is defined by removing vertices from $\mathcal{G}$. The *offset* describes the difference between the weight of an MW2PS on the reduced instance $\alpha_w^2(\mathcal{G}')$ and the weight of an MW2PS on the original graph $\alpha_w^2(\mathcal{G})$. In the *reconstruction*, we present how the solution on the reduced instance $\mathcal{S}'$ is used to construct a solution $\mathcal{S}$ for the original graph.

Basic Concepts

We first introduce a meta reduction Heavy Vertex, illustrated in Figure 5.6. This reduction is very similar to Reduction 4.15 for the MWIS problem. It helps to understand the intuition behind the different reduction rules introduced later. Moreover, this example also shows the importance of our additional link set $\mathcal{L}$. Without the link added between the vertices x and y in the reduced instance, both of these vertices could be part of an MW2PS in the reduced instance. This would create a conflict, as x and y have a common neighbor in the original graph. The link set $\mathcal{L}$ ensures that

Figure 5.6: Illustration of Heavy Vertex. The left graph is the original link-graph $\mathcal{G}$ and dashed lines are links in $\mathcal{L}$. Red numbers indicate the vertex weights. On the reduced link-graph $\mathcal{G}'$ (right), light colored edges and vertices are reduced (green included and red excluded). Note that after the reduction, $\deg(y) < \deg_{\mathrm{L}}(y)$.



the vertices x and y are not both part of an MW2PS in the reduced instance, which is crucial for the correctness of our reduction rules. In the reduction rule formulations we use the short notation $\omega_{max}(U) = \max\{\omega(u) \mid u \in U\}$ for a set $U \subseteq V$.

Reduction 5.11 (Heavy Vertex)

Let $v \in V$ such that the weight of the MW2PS in $N_2(v)$ is smaller than the weight of the vertex, i. e., $\alpha_w^2(\mathcal{G}[N_2(v)]) \leq \omega(v)$, then include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(v)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let $v \in V$ such that $\alpha_w^2(\mathcal{G}[N_2(v)]) \leq \omega(v)$ and $\mathcal{S}$ be the MW2PS. If $v \notin \mathcal{S}$, then we construct $\mathcal{S}' = \mathcal{S}' \setminus \{\mathcal{S} \cap N_2(v)\} \cup \{v\}$. The set $\mathcal{S}'$ is also a valid MW2PS and since $\alpha_w^2(\mathcal{G}[N_2(v)]) \leq \omega(v)$, it holds $\omega(\mathcal{S}) \leq \omega(\mathcal{S}')$. Therefore, we can always construct a solution including v. □

Considering the computational expense of calculating $\alpha_w^2(\mathcal{G}[N_2(v)])$, especially for large sets, we impose a bound on it instead. The most intuitive, also used in the independent set approaches, is summing up the weights in the set. It clearly holds that $\alpha_w^2(\mathcal{G}[N_2(v)]) \leq \omega(N_2(v))$. Given the nature of a 2-packing set, we can tighten this bound by the following observation. Since each neighbor has a common neighbor in $N(v)$, namely v, the direct neighborhood $N(v)$ forms a distance-2-clique, which is a set of vertices in $\mathcal{G}$ whose vertices are pairwise connected by a path of length at most 2. Therefore, there can only be one of the direct neighbors contributing to $\alpha_w^2(\mathcal{G}[N_2(v)])$. We bound its weight by $\omega_{max}(N(v))$, yielding the following lemma.

Lemma 5.1. *Let $v \in V$ and $\deg(v) \geq 1$, then $N(v)$ forms a distance-2-clique and it holds*

$$\alpha_w^2(\mathcal{G}[N_2(v)]) \leq \omega(L(v)) + \omega_{max}(N(v)).$$

Proof. Let $v \in V$. The statement is clear if $\deg(v) = 1$. Let therefore $\deg(v) > 1$. Each neighbor in $N(v)$ is adjacent to v, and therefore, each link-neighbor shares a common neighbor, which yields $N(v)$ as a distance-2-clique. It is clear that $\omega(N_2(v)) = \omega(N(v)) + \omega(L(v)) \geq \omega_{max}(N(v)) + \omega(L(v))$. We now show that this is still a bound for $\alpha_w^2(\mathcal{G}[N_2(v)])$. The set $N(v)$ forms a distance-2-clique. Therefore, only one of its vertices can be part of an MW2PS. We can bound this by the maximum weight in the direct neighborhood. Therefore, it holds $\omega_{max}(N(v)) + \omega(L(v)) \geq \alpha_w^2(\mathcal{G}[N_2(v)])$. □

In the example illustrated in Figure 5.6, we have $\alpha_w^2(\mathcal{G}[N_2(v)]) = 7$ and Heavy Vertex is applicable. However, using the naive bound by summing up the weights $\omega(N_2(v)) = 13 \not\leq 10 = \omega(v)$, we can not reduce it. Using Lemma 5.1 on the other hand, we get $\omega_{max}(N(v)) + \omega(L(v)) = 9 \leq 10 = \omega(v)$ and v is reducible.

Figure 5.6 also highlights the importance of our link set. The orange vertices with common neighbors in the original graph are *linked* in the reduced link-graph. This example shows that in a reduced instance, there can be vertices $x, y \in V$ such that $\deg(y) < \deg_L(y)$. Without the additional link set, it is not possible to maintain the correct 2-neighborhood information of the vertices during the reduction process.

Main Data Reduction Rules

Now, we introduce additional data reduction rules. The first reduction searches for neighbors $u \in N_2(v)$ of a vertex v, which can be removed since they can always be swapped for the vertex v in a solution containing u. We can use different bounds depending on whether u is a direct or a link neighbor.

Reduction 5.12 (Neighbor Removal)

Let $u, v \in V$ with

1. *$u \in N(v)$ and $\alpha_w^2(\mathcal{G}[L(v) \setminus N_2[u]]) + \omega(u) \leq \omega(v)$, or*
2. *$u \in L(v)$ and $\alpha_w^2(\mathcal{G}[N_2[v] \setminus N_2[u]]) + \omega(u) \leq \omega(v)$,*

then, exclude u.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(u)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - u$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

Proof. Let $v \in V$ and $u \in N_2[v]$ with $\alpha_w^2(\mathcal{G}[N_2[v] \setminus N_2[u]]) + \omega(u) \leq \omega(v)$. In the case of $u \in N(v)$, it holds $N[v] \subseteq N_2[u]$ which results in $L(v) \setminus N_2[u] = N_2[v] \setminus N_2[u]$. Further, let $\mathcal{S}$ be an MW2PS of $\mathcal{G}$. If u is not part of $\mathcal{S}$, then it is safe to remove u. Otherwise, it holds $u \in \mathcal{S}$. Then $v \in N_2[u]$ is not in $\mathcal{S}$ and $\omega(\mathcal{S} \cap N_2[v]) = \omega(\mathcal{S} \cap (N_2[v] \setminus N_2[u]) \cup \{u\}) = \alpha_w^2(\mathcal{G}[N_2[v] \setminus N_2[u]]) + \omega(u) = \omega(v)$; otherwise u and $\mathcal{S} \cap N_2[v] \setminus N_2[u]$ can be swapped with v in $\mathcal{S}$ for obtaining a 2-packing set of larger weight. Thus, $\mathcal{S}' = \mathcal{S} \setminus N_2[v] \cup \{v\}$ is an MW2PS of $\mathcal{G}$ excluding u and $\alpha_w^2(G) = \alpha_w^2(G')$. □

The next reduction is illustrated in Figure 5.6 and combines the idea of Heavy Vertex and uses the inequality from Lemma 5.1 as a bound.

Reduction 5.13 (Neighborhood Removal)

Let $v \in V$. If $\omega(v) \geq \omega(N_2[v]) + \omega_{max}(N(v))$, then, include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(v)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let $v \in V$ and $\omega(v) \geq \omega(L(v)) + \omega_{max}(N(v))$. We want to apply Neighbor Removal. Therefore, we distinguish two cases. First, let $u \in N(v)$ and assume it is in some MW2PS. Then, $\alpha_w^2(\mathcal{G}[N_2[v] \setminus N_2[u]]) + \omega(u) = \alpha_w^2(\mathcal{G}[L(v) \setminus N_2[u]]) + \omega(u)$ and we can bound this by $\alpha_w^2(\mathcal{G}[L(v) \setminus N_2[u]]) + \omega(u) \leq \alpha_w^2(\mathcal{G}[L(v)]) + \omega_{max}(N(v)) \leq \omega(L(v)) + \omega_{max}(N(v)) \leq \omega(v)$, using Lemma 5.1. This way, we can always swap v for u and remove all vertices in the direct neighborhood of v. Second, let $u \in L(v)$ be in some MW2PS. Since we already removed the complete direct neighborhood, it holds $\alpha_w^2(\mathcal{G}[N_2[v] \setminus N_2[u]]) + \omega(u) = \alpha_w^2(\mathcal{G}[L(v) \setminus N_2[u]]) + \omega(u) \leq \omega(L(v) \setminus N_2[u]) + \omega(u)$. Similar to the first case, we show that u can always be replaced by v with the following estimation $w(L(v) \setminus N_2[u]) + \omega(u) \leq \omega(L(v)) \leq \omega(v)$. This argument allows us to reduce the complete 2-neighborhood, and v is the only vertex left in its component. Hence, v must be in an MW2PS and can be removed from G. □

In Split Neighbor Removal, we tighten the bound used in Neighbor Removal even further, depending on different special cases.

Reduction 5.14 (Split Neighbor Removal)

Let $u, v \in V$ such that $u \in N_2(v)$. If one of the following cases applies, we have an upper-bound U for $\alpha_w^2(\mathcal{G}[N_2[v] \setminus N_2[u]])$ and obtain a special case of the Neighbor Removal Reduction: $U + \omega(u) \leq \omega(v)$.

1. If $u \in N(v)$, then, $U = \omega(L(v) \setminus N_2[u])$.

2. If $u \in L(v)$, then, $U = \min\{\omega(N_2[v] \setminus N_2[u]),\ \omega_{max}(N(v)) + \omega(L(v) \setminus N_2[u])\}$.

In these cases, we can exclude u.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(u)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - u$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

Proof. Let $u, v \in V$ such that $u \in N_2[v]$. We generally start with the upper-bound $U = \omega(\mathcal{G}[N_2[v] \setminus N_2[u]])$ from the Neighbor Removal Reduction. We can further tighten the bound and simplify its computation in the following cases.

1. Follows from Neighbor Removal (case 1).
2. Let $u \in L(v)$. Using Lemma 5.1, we estimate $\alpha_w^2(N_2[v] \setminus (N_2[u] \cup \{v\})) \leq \omega_{max}(N(v)) + \omega(L(v) \setminus N_2[u])$. Note that we can get a tighter bound when we stay with the original $\omega(\mathcal{G}[N_2[v] \setminus N_2[u]])$. We take the minimum of these two bounds.

□

Remark 5.2 (Efficient Split Neighbor Removal)**.** *Before we compute $\omega(L(v) \setminus N_2[u])$ in Split Neighbor Removal, we check whether $\omega(N(u)) \geq \omega(L(v)) + \omega(N(v))$ and if true, exclude u. We can do this, since $\omega(N(u)) \geq \omega(L(v)) + \omega(N(v)) = \omega(N_2[v]) - \omega(v)$ is equivalent to $\omega(N_2[v]) - \omega(N(u)) \leq \omega(v)$. We get $\omega(L(v) \setminus N_2[u]) + \omega(u) \leq \omega(N_2[v] \setminus N(u)) = \omega(N_2[v]) - \omega(N(u)) \leq \omega(v)$ and can apply case 1 of Split Neighbor Removal. This yields a tighter bound $U = \omega(L(v) \setminus N_2[u])$ if $\omega(N(v) \setminus N[u]) < \omega(L(v) \cap N(u)) = \omega(N(u) \setminus N[v])$.*

The idea behind the next reduction, called Intersection Removal, is similar to Neighbor Removal; however, we now consider two adjacent vertices: u and v. If the reduction is applicable, one of these will be in the MW2PS; therefore, we can exclude their common neighbors.

Reduction 5.15 (Intersection Removal)

Let $u, v \in V$ such that $u \in N_2(v)$. If $\omega(v) \geq \omega(N_2(v) \setminus \{u\})$, then, exclude $K = (N_2[u] \cap N_2[v]) \setminus \{u, v\}$.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(K)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - K$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

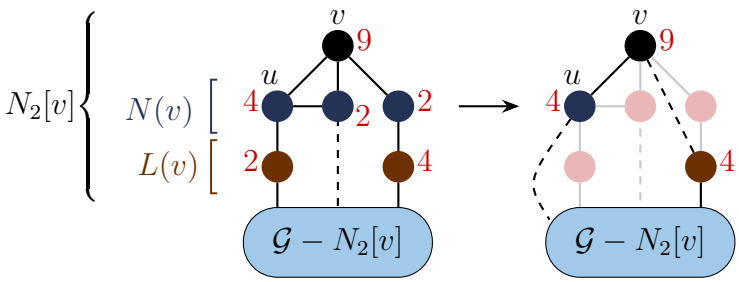


Figure 5.7: Application of Split Intersection Removal. The original link-graph on the left is reduced to the link-graph on the right. Red numbers indicate vertex weights, dashed lines links in $\mathcal{L}$. Light-colored vertices and edges are reduced. Note that Intersection Removal is not applicable in this case.

Proof. Let $u, v \in V$ such that $u \in N_2(v)$ and $\omega(v) \geq \omega(N_2(v) \setminus \{u\})$. We assume a set of vertices $\mathcal{S} \subset N_2(v) \setminus \{u\}$ is part of an MW2PS. We know that $\omega(v) \geq \omega(N_2(v) \setminus \{u\}) \geq \omega(\mathcal{S})$. Therefore, we can always create a new solution $\mathcal{S}' = \mathcal{S} \setminus \mathcal{S} \cup \{v\}$ of larger or equal size. This shows that either v or u are in an MW2PS, so the intersection of their neighborhoods can be removed. □

Split Intersection Removal, illustrated in Figure 5.7, tightens the bounds used by the Intersection Removal by further using Lemma 5.1.

Reduction 5.16 (Split Intersection Removal)

Let $u, v \in V$ *and* $K = (N_2(u) \cap N_2(v)) \setminus \{u, v\}$. *If*

1. $u \in N(v)$, $\omega(v) \geq \omega(L(v)) + \omega_{max}(N(v) \setminus \{u\})$
2. $u \in L(v)$ *and* $\omega(v) \geq \omega(L(v) \setminus \{u\}) + \omega_{max}(N(v))$,

then, exclude K.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(K)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - K$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

Proof. This reduction follows using Lemma 5.1 and Intersection Removal. □

The following reduction uses a special relation between two adjacent vertices u and v. It is applicable if $N[u] = N_2[v]$. In this case, $N_2[v]$ forms a distance-2-clique, and v can be included if it has the highest weight in $N_2[v]$, see Weighted Clique. If not, we exclude as many neighbors as possible.

Reduction 5.17 (Domination)

Let $v \in V$ and $u \in N(v)$ with $N[u] = N_2[v]$. If

1. $\omega(v) \geq \omega_{max}(N[u])$, then, include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(L(v))$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N[v]$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

2. $\omega(v) \geq \omega(N(u) \setminus \{v\})$, then, exclude $K = N(u) \setminus \{v\}$.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(K)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - K$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

3. $\omega(v) \geq \omega(u)$, then, exclude u.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(u)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - u$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}')$
Reconstruction	$\mathcal{S} = \mathcal{S}'$

Proof. The first case holds since every neighbor of v is a direct neighbor of u. Since u spans a distance-2-clique over its direct neighborhood, v is distance-2-simplicial. Therefore, if $\omega(v) \geq \omega_{max}(N[u])$, we can include it in the solution. In the second case, if $\omega(v) \geq \omega(N(u) \setminus \{v\})$ we get $\omega(N(u) \setminus \{v\}) = \omega(N[u] \setminus \{u, v\}) \geq \omega(N_2[v] \setminus \{v, u\})$ and we can apply Intersection Removal to get the stated result. In the third case, if $\omega(v) \geq \omega(u)$, we can apply Split Neighbor Removal Case 1. □

Remark 5.3 ((Memory) Efficient Domination)**.** *Instead of testing $N[u] = N_2[v]$, we test if $\deg_L(v) + \deg(v) = \deg(u)$, which is equivalent, since for $u \in N(v)$, each neighbor of u is a link-neighbor of v, i. e., $N[u] \subseteq N_2[v]$. The degree equality yields that $|N[u]| = 1 + \deg(u) = 1 + \deg_L(v) + \deg(v) = |N_2[v]|$. Therefore, the sets have to be equal, and we get $N[u] = N_2[v]$. To check this degree-equality, we need to initialize the link-neighborhood of each vertex. To avoid this for some cases when Domination is not applicable and the link-degree not computed yet, we first check whether $N[v] \subseteq N[u]$ and only compute the link-neighborhoods for these vertices.*

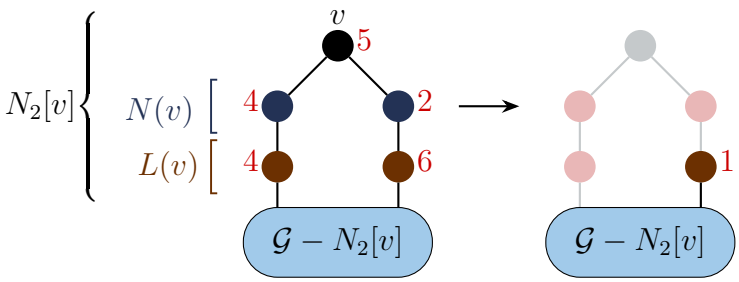


Figure 5.8: Application of D2-Simplicial Weight Transfer. The original link-graph on the left is reduced to the link-graph on the right. Red numbers indicate vertex weights. Light red vertices are excluded, gray vertices are folded, and light gray edges are removed.

The following reduction rule, Weighted Clique includes vertices v if these are distance-2-simplicial,i. e., their neighborhood $N_2[v]$ forms a distance-2-clique. If the weight of v is the highest in $N_2[v]$, we can include v in the solution.

Reduction 5.18 (Weighted Clique)

Let $v \in V$ be distance-2-simplicial in $\mathcal{G}$ such that $\omega(v) = \omega_{max}(N_2[v])$, then include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let $v \in V$ be distance-2-simplicial in $\mathcal{G}$ such that $\omega(v) = \omega_{max}(N_2[v])$. Since the vertex v is distance-2-simplicial, i. e., the neighborhood $N_2[v]$ forms a distance-2-clique, there can only be one of these vertices in the MW2PS. Since for all vertices $u \in N_2[v]$ it holds that $\alpha_w^2(\mathcal{G}[N_2[v] \cup N_2[u]]) \leq \alpha_w^2(\mathcal{G}[N_2[v]]) = \omega_{max}(N_2[v]) = \omega(v)$, we can remove u by applying the Neighborhood Removal Reduction resulting in v being left to include in the solution. □

If Weighted Clique is not applicable, we can still reduce some part of the neighborhood of distance-2-simplicial vertices. The cases where this is possible are described in D2-Simplicial Weight Transfer, illustrated in Figure 5.8.

Reduction 5.19 (D2-Simplicial Weight Transfer)

Let $v \in V$ be distance-2-simplicial, and suppose that the set of distance-2-simplicial vertices $S^2(v) \subset N_2(v)$ is such that for all $u \in S^2(v)$ holds $\omega(v) \geq \omega(u)$, then fold v into its neighborhood.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(K)$
Reduced Graph	*Construct the link-graph* $\mathcal{G}'$ *as follows* • *remove all* $u \in K \coloneqq \{u \in N_2(v) \mid \omega(u) \leq \omega(v)\}$*, and let the remaining neighbors be denoted by* $N'(v)$ • *remove* v *and for each* $u \in N'(v)$ *set* $\omega(u) = \omega(u) - \omega(v)$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$
Reconstruction	*If* $\mathcal{S}' \cup N'(v) = \emptyset$*, then* $\mathcal{S} = \mathcal{S}' \cup \{v\}$*, else* $\mathcal{S} = \mathcal{S}'$*.*

Proof. The first step of D2-Simplicial Weight Transfer is applying the Neighbor Removal Reduction to all vertices $u \in N_2(v)$ with $\omega(u) \leq \omega(v)$. Let $x \in N'(v)$ and $\mathcal{S}'$ be the solution on $\mathcal{G}'$. In the second step, we need to distinguish two cases.

Case 1: $\mathcal{S}' \cap N'(v) = \emptyset$. We show that $\omega(v) + \alpha_w^2(\mathcal{G}[V \setminus N_2[v]]) \geq \alpha_w^2(\mathcal{G}[V \setminus \{v\}])$. Since $x \notin \mathcal{S}$, we have $\alpha_w^2(\mathcal{G}') \geq \omega'(x) + \alpha_w^2(\mathcal{G}'[V' \setminus N_2[x]]) = \omega(x) - \omega(v) + \alpha_w^2(\mathcal{G}'[V' \setminus N_2[x]])$. Therefore, the heaviest 2PS containing v has at least the weight of the heaviest 2PS containing any link-neighbor of v. Since additionally, $\omega(v) + \alpha_w^2(\mathcal{G}[V \setminus N_2[v]]) \geq \alpha_w^2(\mathcal{G}[V \setminus \{v\}])$, we know that $\mathcal{S} = \mathcal{S}' \cup \{v\}$ is an MW2PS of $\mathcal{G}$.

Case 2: $\mathcal{S}' \cap N'(v) \neq \emptyset$. We show that $\mathcal{S}' = \mathcal{S}$ is an MW2PS of $\mathcal{G}$. Since v is distance-2-simplicial, it holds $|\mathcal{S}' \cap N'(v)| = 1$. Define $\mathcal{G}''$ as the link-graph resulting from increasing the weights of $N'(v)$ again by $\omega(v)$, i. e., the original weights. The set $\mathcal{S}'$ is also an MW2PS for the link-graph $\mathcal{G}''$, since otherwise, it would not have been optimal on $\mathcal{G}'$. Therefore, we have $\omega'(\mathcal{S}') + \omega(v) = \omega(\mathcal{S}')$, since exactly one vertex from $N'(v)$ is part of $\mathcal{S}'$. In the next step, we add the vertex v to the link-graph $\mathcal{G}''$, resulting in the link-graph $\mathcal{G}'''$, i. e., this is the link-graph after step 1 of the reduction. We now assume, that there is an MW2PS $\mathcal{S}^*$ for $\mathcal{G}'''$ with $\omega(\mathcal{S}^*) > \omega(\mathcal{S}')$. Then, $v \in \mathcal{S}^*$, since this is the only vertex added to $\mathcal{G}'''$. This implies that no neighbor of v is in $\mathcal{S}^*$. We now have $\omega(\mathcal{S}^* \setminus \{v\}) = \omega(\mathcal{S}^*) - \omega(v) > \omega(\mathcal{S}') - \omega(v) = \omega'(\mathcal{S}') + \omega(v) - \omega(v) = \omega'(\mathcal{S}')$. Since $\mathcal{S}^* \setminus \{v\}$ does not contain any vertex from $N_2[v]$, it is an MW2PS of $\mathcal{G}'$ that is of larger weight than $\mathcal{S}'$. This contradicts the assumption, and we have $\mathcal{S}' = \mathcal{S}$. □

The following reduction describes a folding. Here, we postpone the decision about the solution status of a vertex to a later point. Depending on the solution status of the new vertex, we decide the solution status of the folded vertices.

Our Neighborhood Folding is applicable to a vertex v with a 2-packing neighborhood, i. e., $N_2(v)$ is a 2-packing set in $\mathcal{G}$. In this situation, we cannot include the vertex v directly, but we know that in the end, either v or its complete neighborhood

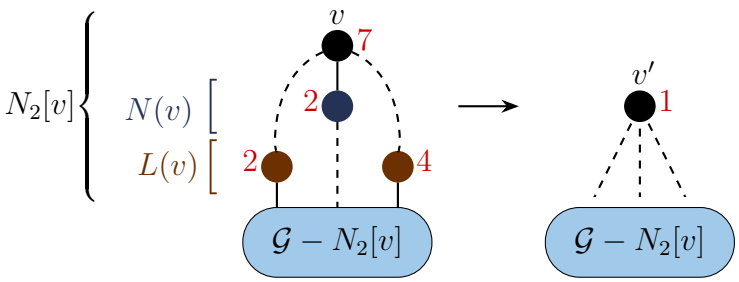


Figure 5.9: Application of Neighborhood Folding. The original link-graph on the left is reduced to the instance on the right. Red numbers indicate vertex weights and dashed lines links in $\mathcal{L}$.

is in the solution. Therefore, we can fold these vertices. Neighborhood Folding is illustrated in Figure 5.9.

Reduction 5.20 (Neighborhood Folding)

Let $v \in V$ and $N_2(v)$ be a 2-packing set in $\mathcal{G}$. If $\omega(N_2(v)) > \omega(v)$, but $\omega(N_2(v)) - \omega_{min}(N_2(v)) \leq \omega(v)$, then, fold $N_2[v]$ into a new vertex v'.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(N_2[v])$
Reduced Graph	$\mathcal{G}' = \mathcal{G}[(V \setminus N_2[v]) \cup \{v'\}]$, *set weight* $\omega(v') = \omega(N_2(v)) - \omega(v)$ *and* $22316v' = \emptyset$.
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$
Reconstruction	*If* $v' \in \mathcal{S}'$ *then* $\mathcal{S} = (\mathcal{S}' \setminus \{v'\}) \cup (N_2(v))$, *else,* $\mathcal{S} = \mathcal{S}' \cup \{v\}$.

Proof. Let $v \in V$ and $N_2(v)$ be a 2-packing set of $\mathcal{G}$. First, note that after folding, the following graphs are identical: $\mathcal{G}'[V' \setminus N_{2\mathcal{G}'}[v']] = \mathcal{G}[V \setminus N_4[v]]$ and $\mathcal{G}'[V' \setminus \{v'\}] = \mathcal{G}[V \setminus N_2[v]]$. Let $\mathcal{S}'$ be an MW2PS of $\mathcal{G}'$. Now, we distinguish two cases.

Case 1: $v' \in \mathcal{S}'$. We show that $\omega(N_2(v)) + \alpha_w^2(\mathcal{G}[V \setminus N_4[v]]) \geq \omega(v) + \alpha_w^2(\mathcal{G}[V \setminus N_2[v]])$ resulting in all neighbors of v are together in an MW2PS of $\mathcal{G}$. Since $v' \in \mathcal{S}'$,

$$\begin{aligned}\alpha_w^2(\mathcal{G}') &= \omega(v') + \alpha_w^2(\mathcal{G}'[V' \setminus N_{2\mathcal{G}'}[v']]) \\ &= \omega(N_2(v)) - \omega(v) + \alpha_w^2(\mathcal{G}'[V' \setminus N_{2\mathcal{G}'}[v']]) \\ &= \omega(N_2(v)) - \omega(v) + \alpha_w^2(\mathcal{G}[V \setminus N_4[v]])\end{aligned}$$

Since $v' \in \mathcal{S}'$, we have $\alpha_w^2(\mathcal{G}') \geq \alpha_w^2(\mathcal{G}'[V' \setminus \{v'\}]) = \alpha_w^2(\mathcal{G}[V \setminus N_2[v]])$. Thus, $\omega(N_2(v)) + \alpha_w^2(\mathcal{G}[V \setminus N_4[v]]) \geq \omega(v) + \alpha_w^2(\mathcal{G}[V \setminus N_2[v]])$ and the vertices of the link-neighborhood are together in some MW2PS of $\mathcal{G}$ and $\alpha_w^2(\mathcal{G}) = \omega(N_2(v)) + \alpha_w^2(\mathcal{G}[V \setminus N_4[v]]) = \alpha_w^2(\mathcal{G}') + \omega(v)$.

Case 2: $v' \notin \mathcal{S}'$. We show that v is in some MW2PS, by proving that $\omega(v) + \alpha_w^2(\mathcal{G}[V \setminus L(v)]) \geq \omega(N_2(v)) + \alpha_w^2(\mathcal{G}[V \setminus N_4[v]])$. Since $v' \notin \mathcal{S}'$, we have $\alpha_w^2(\mathcal{G}') =$

$\alpha_w^2(\mathcal{G}'[V' \setminus \{v'\}]) = \alpha_w^2(\mathcal{G}[V \setminus N_2[v]])$. Since $\mathcal{S}'$ is an MW2PS of $\mathcal{G}'$, we obtain $\alpha_w^2(\mathcal{G}[V \setminus N_2[v]]) = \alpha_w^2(\mathcal{G}') \geq \omega(N_2(v)) - \omega(v) + \alpha_w^2(\mathcal{G}[V \setminus N_4[v]])$. Therefore, we have $\omega(v) + \alpha_w^2(\mathcal{G}[V \setminus L(v)]) \geq \omega(N_2(v)) + \alpha_w^2(\mathcal{G}[V \setminus N_4[v]])$ and v is in some MW2PS of $\mathcal{G}$. Lastly, $\alpha_w^2(\mathcal{G}) = \omega(v) + \alpha_w^2(\mathcal{G}[V \setminus N_2[v]]) = \omega(v) + \alpha_w^2(\mathcal{G}')$. □

Remark 5.4. *In Neighborhood Folding, we can efficiently prune the search space by checking if* $\deg(v) \leq 1$. *Otherwise,* $N_2(v)$ *would contain a distance-2-clique, see Lemma 5.1 and therefore not be a 2-packing set.*

Data Structure

Here, we describe the data structure and some implementation details for the data reduction rules. These are improtant to understand the more efficient data reduction rules presented in the second part of this section.

Maintain Neighborhood Information. We maintain important neighborhood information of a vertex to speed up the reduction process. For this, we maintain the size of the link-neighborhood $\deg_{\mathrm{L}}(v)$ and the weight of the maximum weight vertex. Additionally, we maintain the sum of the weights in the direct neighborhood for a vertex.

On-Demand-Neighborhood. Our reductions operate on a dynamic link-graph data structure based on adjacency arrays representing undirected edges and links by two directed ones. Internally, we separate edges and links with two adjacency arrays. In addition to the necessary links for the reductions, we compute and store the full link-neighborhood of a vertex whenever we compute it. We do this even if the reduction was not successful to avoid recomputation. Generally, we only compute the link-neighborhood of a vertex on demand. Here, we make special use of reductions that do not need to know the link-neighborhood to exclude a vertex. These reductions are applied initially, reducing the link-graph without building the full square graph. This approach additionally reduces the amount of work needed to reduce the instance. When a vertex u is removed from the link-graph, we delete every edge and all links pointing to it. Removing all incoming edges can take time $\mathcal{O}(\Delta^4)$ where Δ is the highest degree in the graph. With the on-demand technique, the link-neighborhood of a vertex v is not computed in advance, and therefore, potentially fewer edges must be deleted.

Bulk Hide Operation. Hiding the incoming edges and links can be computationally expensive for high-degree vertices. For example, for large distance-2-cliques, it

can be expensive to exclude multiple clique members, i. e., applying Weighted Clique or D2-Simplicial Weight Transfer. Hiding one member of this distance-2-clique takes $\mathcal{O}(\Delta^4)$ time because we have to search through the edge and link adjacency array for each clique member to remove the connections. Then, hiding $k \in \mathcal{O}(\Delta^2)$ distance-2-clique members this takes $\mathcal{O}(k\Delta^4)$ time. To avoid this, we put all the hide operations of the respective distance-2-clique members together by using a *bulk hide operation*. This operation hides all edges and links to clique members in a single pass. This reduces the running time from $\mathcal{O}(k\Delta^4)$ to $\mathcal{O}(\Delta^4)$ and possibly decreases cache faults.

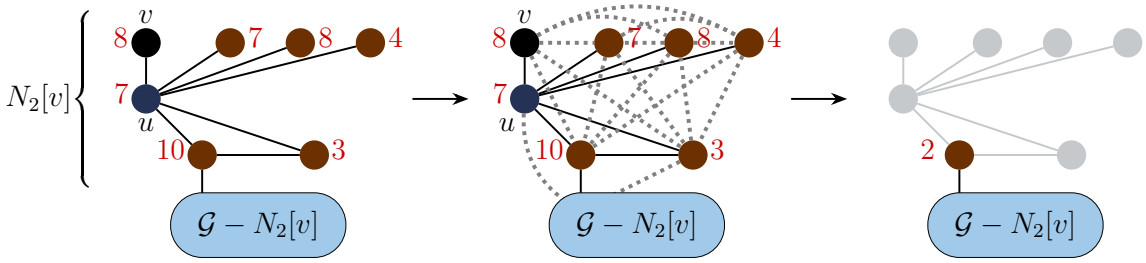
Efficient Data Reduction Rules

So far, we presented data reduction rules based on the link-neighborhood information. However, using the link-neighborhood, e. g., determining or iterating over the 2-neighborhood, can be expensive regarding running time and memory consumption. In our implementation of the reduction rules, we postpone determining a link neighborhood as long as possible. Especially if a reduction is not applicable, determining the link-neighborhood early results in more work throughout the whole reduction process as this link-neighborhood has to be maintained.

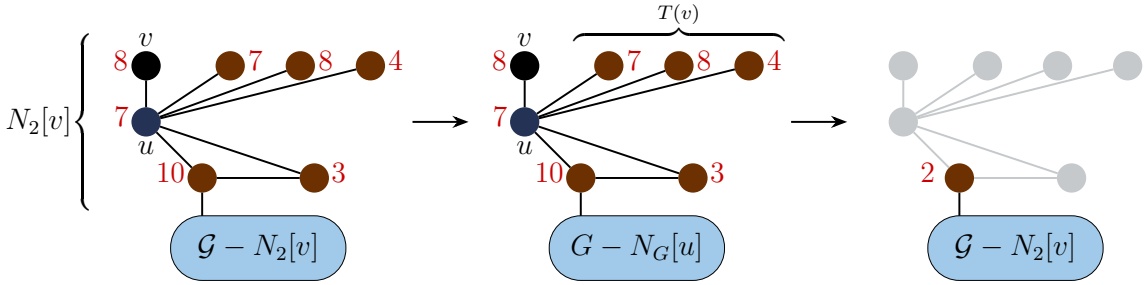
In this section, we present efficient data reduction rules which circumvent these issues. Whereas arbitrary link neighborhoods are of size $\mathcal{O}(\Delta^2)$, the following data reduction rules fully circumvent considering link neighborhoods of size $\omega(\Delta)$ or ensure that applying a data reduction rule for all vertices does overall take at most $\mathcal{O}(n+m)$ running time. Moreover, we implement them such that no link neighborhoods are initialized and maintained by the link-graph data structure during the applicability tests.

Instead, the main idea behind these rules is to exploit the properties of the vertices V_G in the original input graph G to reduce $\mathcal{G}$ further. Intuitively, we reduce vertices in the link-graph $\mathcal{G}$ by looking up neighborhoods of (already reduced) vertices in G. Note that G refers to the original, unreduced input graph (without the link set $\mathcal{L}$). With V_G, we refer to vertices in the original graph G, while $N_G(v)$ denotes the original neighborhood of the vertex v in G. We refer to neighborhoods in the currently reduced link-graph $\mathcal{G}$ if a refernce is omitted in the notation.

The first data reduction rule, Fast Degree-1, fully reduces vertices of degree one in the input graph G. When applying this reduction rule and building the link neighborhood in $\mathcal{G}$, a degree one vertex always forms a distance-2-clique. Figure 5.10 gives an example and points out the difference to the more general D2-Simplicial Weight Transfer Reduction, which also reduces this pattern. Fast Degree-1 can be applied to degree one and zero vertices that arise during the reduction progress in $\mathcal{G}$. Figure 5.10b illustrates Fast Degree-1.



(a) Illustration of D2-Simplicial Weight Transfer. In an intermediate step, links are initialized (OD2N) to check if $N_2[v]$ forms a distance-2-clique. These are illustrated as dotted gray lines. When applying D2-Simplicial Weight Transfer, we have to hide all of these links, which is computationally expensive even when using the bulk hide operation.



(b) Illustration of Fast Degree-1. By first checking the original input graph G, we know v forms a distance-2-clique and we can fold v with its twins $T(v)$ without initializing links.

Figure 5.10: We illustrate the benefits of using the fast reductions. The vertices in $L(v)$ are brown. Fast Degree-1 and the D2-Simplicial Weight Transfer are equally effective on this example but in contrast to D2-Simplicial Weight Transfer (Figure 5.10a), Fast Degree-1 fully circumvents to maintain links (Figure 5.10b).

Reduction 5.21 (Fast Degree-1)

Let $u \in V_G$ with $v \in V \cap N_G(u)$ so that $\deg(v) \leq 1$ and $L(v) \subset N_G(u)$. Further, let $T(v) = \{x \in V \cap N_G(u) : \deg(x) \leq 1 \wedge L(x) \subset N_G(u)\} \setminus \{v\}$ be denoted as the twins of v in $\mathcal{G}$ with $\omega(v) \geq \max_{x \in T(v)} w(x)$ and at most degree 1. Then, fold v and its twins.

Link Set	$\mathcal{L} = \mathcal{L}$
Reduced Graph	*Construct the link-graph $\mathcal{G}'$ as follows* • *remove all $x \in (V \cap N_G[u]) \setminus \{v\}$ with $\omega(x) \leq \omega(v)$, and let the remaining link-neighborhood of v be denoted by $L'(v)$* • *remove v and for each $x \in L'(v)$ set $\omega(x) = \omega(x) - \omega(v)$*
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$
Reconstruction	*If $\mathcal{S}' \cup L'(v) = \emptyset$, then $\mathcal{S} = \mathcal{S}' \cup \{v\}$, else $\mathcal{S} = \mathcal{S}'$.*

Proof. $T(v)$ is a subset of distance-2-simplicial vertices $S^2(v)$ of the considered distance-2-clique. We can now apply D2-Simplicial Weight Transfer by observing that we can relax its maximality constraint for $w(v)$. Although v might not have maximal weight among all distance-2-simplicial vertices in this distance-2-clique, we can still apply a weight shift by $w(v)$. To see this, we assume that simplicial vertices of larger weight than $w(v)$ remain in $\mathcal{G}'$. One can apply D2-Simplicial Weight Transfer to the remaining distance-2-clique $L'(v)$ with a simplicial vertex $x \in L'(v)$ that satisfies the maximality constraint in $(\mathcal{G}', w')$, and check that this gives the same reduced instance and weight function as applying the D2-Simplicial Weight Transfer directly to $N_2[x]$ in $(\mathcal{G}, w)$. Note that x also satisfies the maximality constraint in $\mathcal{G}$ since $w(x) = w'(x)+w(v) \geq w'(y)+w(v) = w(y)$ for all distance-2-simplicial $y \in S(v)\backslash T(v)$, and $w(x) \geq w(v) \geq w(y)$ for $y \in T(v)$. □

Remark 5.5 (Implementing the Fast Degree-1)**.** *The implementation works in rounds where each round considers only degree one vertices. Initially, we consider all degree one vertices of G, as they remain distance-2-simplicial as long as they are not yet reduced. In the upcoming rounds, we reduce new degree one and zero vertices. Note that the subset condition for the link set is not trivially fulfilled for these vertices. Therefore, we maintain a list of vertices that do not fulfill that condition.*

The following data reduction reduces degree-two vertices in V_G almost analogously to Fast Degree-1. The key difference is that they do not necessarily point to distance-2-cliques. However, we can still reduce twins and possibly include a degree-2 vertex using Lemma 5.1.

Reduction 5.22 (Fast Degree-2)

Let $u, y \in V_G$ with $v \in V \cap N_G(u) \cap N_G(y)$ so that $\deg(v) \leq 2$ and $L(v) \subset N_G(u) \cup N_G(y)$. Further, let $T(v) = \{x \in V \cap N_G(u) \cap N_G(y) \mid \deg(x) \leq 2 \wedge L(x) \subset N_G(u) \cup N_G(y)\} \setminus \{v\}$ be denoted as the twins of v in $\mathcal{G}$ of at most degree two. If $\omega(v) \geq \max_{x \in T(v)} w(x)$, then fold v with its twins.

Link Set	*If v is included, $\mathcal{L} = \mathcal{L} \cup link(N_2[v])$, else $\mathcal{L} = \mathcal{L}$*
Reduced Graph	*Construct the link-graph $\mathcal{G}'$ as follows* • *remove all $x \in T(v)$* • *include v if $u \in N_G(y)$ and $\omega(v) \geq \max\{w(u), w(y), c_u + c_y\}$, or $u \notin N_G(y)$ and $\omega(v) \geq \max\{w(u) + c_y, w(y) + c_u, c_u + c_y\}$ where $c_y = \max_{x \in N(y)\setminus\{u,v\}} \omega(x)$ and $c_u = \max_{x \in N(u)\setminus\{y,v\}} \omega(x)$.*

Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$ *if* v *was included; otherwise* $\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}')$
Reconstruction	*If* $\mathcal{S} \cup L'(v) = \emptyset$, *then* $\mathcal{S} = \mathcal{S}' \cup \{v\}$, *else* $\mathcal{S} = \mathcal{S}'$.

Proof. Consider $u, y \in V_G$ with $v \in V \cap N_G(u) \cap N_G(y)$ as given above. Then, it holds $N(v) = \{u, y\} \cap V = N(x)$ for every $x \in T(v)$. Since all links incident to v or x are given via their (former) direct neigbors u and y, it holds $L(x) \setminus \{v\} = L(v) \setminus \{x\}$. Therefore, we know $N_2[v] = N_2[u]$. If now $\omega(v) \geq \max_{x \in T(v)} w(x)$, then it holds $\omega(v) \geq \omega(x) = \omega(x) + \omega(N_2[v] \setminus N_2[x])$ for every twin $x \in T(v)$. Thus, we can apply the second case of Split Neighbor Removal to remove all twins safely.

If v has sufficiently large weight, we can include v by utalizing an upper bound U for $\alpha_w^2(\mathcal{G}[N_2(v)])$. It allows us to apply Heavy Vertex and further to include v. The key observation is that the remaining vertices of $N_G(u) \cap V \setminus \{y\}$ form a distance-2-clique in $\mathcal{G}$. Analogously, $N_G[y] \cap V \setminus \{u\}$ forms a distance-2-clique in $\mathcal{G}$. Note that, at most one vertex of a distance-2-clique can participate in an MW2PS.

Now, consider that u and y are neighbors in G and $\omega(v) \geq \max\{w(u), w(y), c_u + c_y\}$. Now, if u and y are direct neighbors, either u, y, or neighbors of u or y can be part of an MW2PS. Thus, $\max\{w(u), w(y), c_u + c_y\}$ is an upper bound for the 2-neighborhood of v.

If u and y are not adjacent, they are linked, and we can include v if $w(v) \geq \max\{\omega(u) + c_y, \omega(y) + c_u, c_u + c_y\}$ with a similar argument. If u is part of an MW2PS, only vertices of $N_G(y) \cap V$ regarding the 2-neighborhood of v can be part of an MW2PS. Analogously, if y is part of an MW2PS, only vertices of $N_G(u) \cap L(v)$ but none of $L(v) \setminus N_G(u)$ can be in an MW2PS. This gives us the upper bound for $\alpha_w^2(\mathcal{G}[N_2(v)])$, which allows us to include v with Heavy Vertex. □

Remark 5.6 (Implementing Fast Degree-2)**.** *Fast Degree-2 can be implemented similarly to Fast Degree-1. However, we do not trace new vertices of degree two or one and test this reduction only once for each vertex of* V_G.

The following data reduction is based on Neighborhood Removal and uses an upper bound on the summed weight in the link neighborhood. The upper bound is computed for G. Before any reductions are applied, we compute the upper bound for all vertices in time $\mathcal{O}(|E_G|)$ with two scans over all neighborhoods in G.

Reduction 5.23 (Fast Neighborhood Removal)

Let $v \in V$ with $w(v) \geq \omega_{max}(v) + \sum_{u \in N_G(u)} w(N_G(u)) - w(v)$, then include v.

Link Set	$\mathcal{L} = \mathcal{L} \cup link(v)$
Reduced Graph	$\mathcal{G}' = \mathcal{G} - N_2[v]$
Offset	$\alpha_w^2(\mathcal{G}) = \alpha_w^2(\mathcal{G}') + \omega(v)$
Reconstruction	$\mathcal{S} = \mathcal{S}' \cup \{v\}$

Proof. Let $v \in V$ with $w(v) \geq \omega_{max}(v) + \sum_{u \in N_G(u)} w(N_G(u)) - w(v)$. It holds $L(v) \subseteq N_G(N_G(v)) = \bigcup_{u \in N_G(v)} N_G(u) \setminus \{v\}$. Thus, we obtain $\omega(L(v)) \leq \sum_{u \in N_G(v)} \omega(N_G(u) - \omega(v))$ and can apply Neighborhood Removal. □

5.4.2 Algorithm Description

In this section, we introduce methods to solve the MW2PS problem using the data reduction rules presented in Section 5.4.1. Next to the REDUCE&TRANSFORM approach combined with MWIS solvers, as presented in Section 5.2, we contribute our new heuristic REDW2PACK which applies exact and heuristic data reductions to solve the problem without transforming the graph. Finally, we present the reduce and peel approach DRP, which combines REDW2PACK with the metaheuristic CONCURRENT DIFFERENCE CORE HEURISTIC introduced in Section 4.4.

Baseline Reduce-and-Peel Solver redW2pack

The pseudocode for our baseline approach REDW2PACK is shown in Algorithm 18. This approach alternates between exhaustively applying exact data reductions and a heuristic PEELING step.

The algorithm REDW2PACK *peels* a vertex which is selected by a (randomized) heuristic strategy, i. e., it either *includes* or *excludes* a vertex with respect to a heuristic rating, whenever our exact reduction style `core` was exhaustively applied. This iterative process continues until the graph is empty, and consequently, a heuristic solution for the reduced instance $\mathcal{K}$ is obtained. The peeling step possibly opens up the reduction space so that the exact `core` reductions can *reduce* further vertices.

With reduce-and-peel solvers, a heuristically excluded vertex can lead to a non-maximal solution when the reductions are undone. If possible, we fix this by simply including them when unrolling the stack of applied reductions. Other reductions face the same issue as well. To ensure that the solution for $\mathcal{K}$ is maximal, we maximize the solution greedily by adding free vertices of the largest weight.

Algorithm 18: Pseudocode for redW2pack.

```
Data: Graph G = (V, E, ω), heuristic config C
Result: 2-Packing set S
Procedure redW2pack(G, C):
    𝒢 ← (V, E, ω, ∅)
    while 𝒢 not empty do
        𝒢 ←exact_reduce(𝒢)
        𝒢 ←heuristic_reduce(𝒢,C)
    return restore(𝒢)
```

Other reduce-and-peel solvers for the MWIS, such as HtWIS, restrict reductions to reduce only vertices of a small degree. They keep track of the vertex degrees to efficiently apply reductions when new small-degree vertices arise from the peeling phase. On the contrary, our approach uses the full set of `core` reductions and employs the built-in dependency-checking to test reductions only for vertices in regions where the neighborhood has changed.

Heuristics. We utilize three heuristic ratings, well-known for the MWIS [103, 107]. We call these ratings weight_diff ($w(v) - w(L(v)) - w(N(v))$), weight ($w(v)$), and degree ($-\deg v - \deg_{\mathrm{L}} v$). The ratings are updated throughout the modifications to the instance using priority queues. We maintain the best k candidates in an array and choose uniformly at random one of them in the peeling step. Since randomness is very crucial for different D-Core instances, we also use a *non-adaptive* approach that pre-computes the rating and does a perturbation with a probability of $p \in [0.5, 1]$ of the remaining ranking before each peeling step. The intuition behind *excluding* a vertex with a small weight_diff rating is that a subset of its neighbors is likely to be part of an optimal solution; a small weight rating suggests that the vertex is unlikely to be part of it; and removing a vertex with a small degree rating possibly entails many new reduction applications. Further, one can *include* a vertex with the highest rating for weight_diff and weight. Our preliminary experiments indicate that if solution quality is highly important, it is wiser to *exclude* a vertex heuristically rather than *including* it. A possible reason is that including a vertex has a wider impact since its neighbors are consequently excluded, while the proposed heuristics only capture very local information.

Difference-Core Reduce and Peel

We now give the high-level idea of our heuristic Difference-Core Reduce and Peel (DRP), followed by an overview of the main components. A major drawback of basic reduce-and-peel approaches, as redW2pack, is that wrong decisions during the heuristic steps cannot be undone. With the new algorithm DRP, we aim to overcome this issue using a the new meta-heuristic. We present pseudocode for DRP in Algorithm 19.

High-level Description. We propose a new solver called DRP, which uses the baseline reducing-peeling approach redW2pack combined with the meta-heuristic Concurrent Difference Core Heuristic introduced by Großmann et al. [104].

The Difference Core (D-Core) is a subgraph of the input instance. It is constructed by first computing multiple solutions for an instance and then removing all vertices that are always included or excluded in all solutions generated. This way, the D-Core contains only the vertices where the heuristic is unsure about the decision, making it more likely to find improvements. We run the redW2pack algorithm multiple times with different heuristic strategies to generate different solutions.

By running redW2pack multiple times with different heuristic strategies, we can increase the randomization of our approach. Throughout this process, the best solution found is maintained. Furthermore, this approach enables us to return to the original instance without being stuck with a potentially wrong decision while still utilizing the information about the solutions computed to improve the overall performance.

The Exact Reduced Instance. We use our new data reduction rules with exact_reduce to obtain the reduced instance $\mathcal{K}$.

Building the Difference Core. The D-Core, presented in Section 4.4 is constructed by using a set of solutions $S = \{S_1, \ldots, S_k\}$ to a given problem. With this set of solutions, the D-Core is defined as the induced link-subgraph of a set of vertices $D \subseteq V$ where the solution status of the vertices in D is different in at least one of the solutions in S. Formally, the set D is defined by $D = \{v \in V \mid \exists S_i, S_j \in S : v \in S_i \wedge v \notin S_j\}$, additionally we defined the set of *similar* vertices as $U = V \setminus D$. Intuitively, the D-Core $G[D]$ contains more difficult parts of the instance. In our approach, the D-Core is an MWIS subproblem yielded by applying our reduce&transform routine to the link-graph $\mathcal{G}[D]$. When we find a better solution on the D-Core than the best-found solution so far, we embed the solution into the current best solution

for $\mathcal{K}$. In the following, we explain how we construct this set of solutions S. This approach differs from the original approach by Großmann et al. [104].

Generally, when a new best solution is found, we restart with an empty set $S = \emptyset$. Then, we compute different solutions using variations of the redW2pack heuristic until certain conditions for the D-Core are met. Note that the larger the set S, the more likely it is that the D-Core contains more vertices. The D-Core instances are expected to be small for a small number of solutions. Therefore, we employ a threshold ϕ to restrict the size of the D-Core. We solve the D-Core if the number of similar vertices relative to the vertices in $\mathcal{K}$ falls below ϕ. If the solution for the D-Core is already optimal, we slightly decrease ϕ to ϕ_- times the current measured similarity to observe more redW2pack solutions. On the other hand, if an improvement was made or the solver could not find an improving solution due to a time limit exceeded, we increase ϕ by a factor of ϕ_+ as smaller D-Core instances might be easier to solve. In the latter case, we restart computing the set of solutions S using our redW2pack approach.

Diversification. To diversify the solutions computed by redW2pack, we iterate through the heuristic ratings in the order they are introduced above and pick one configuration given the i-the step in next_config. In order to solve diverse D-Core instances, we also alternate between the adaptive and the non-adaptive rating (in this order) and even refine these two strategies by alternating between *exclude* and *include* in the case of weight_diff and weight. Whenever an adaptive rating strategy is used, we slightly modify it by incrementing k by 1; if a non-adaptive strategy is re-used, we choose a new $p \in [0.5, 1]$ uniformly at random. The exact reduction phase is diversified by shuffling the order of candidates before a reduction is tested.

Solving the Difference Core. To solve the problem on the D-Core, we use our reduce&transform routine, which can be combined with any MWIS solver. In this work, we propose two configurations. The first, DRP-BChils uses the baseline local search used in the CHILS heuristic [104], and the other, DRP-KaMIS uses the exact branch and reduce solver KaMIS BnR [149]. We also add a configuration not using the D-Core strategy for comparison.

5.4.3 Experimental Evaluation

This section presents the experimental evaluation of our work introduced for the MW2PS problem. First, we examine the impact of our reduction approaches on reducing the input instances and solving these using our routine reduce&transform

Algorithm 19: Pseudocode for DRP.

```
Data: Graph G = (V, E, ω), similarity threshold φ, scaling factors φ+ and φ−
Result: 2-Packing set S
Procedure DRP(G, φ, φ+, φ−):
    S ← ∅
    K, α ← ExactReduce(G)
    if K is empty then
        S ← restore(K, α, S)
        return S
    S ← redW2pack(K, C)                  // C initial heuristic configuration
    U ← V_K                              // Similar vertices
    while not time limit exceeded do
        while |U|/n(K) > φ do
            C ← next_config(C)
            S' ← redW2pack(K, C)
            if ω(S') > ω(S) then
                S ← S'
                U ← V_K                  // Reset similar vertices
            else
                U ← U ∩ (V_K \ (SΔS'))   // Update similar vertices
        S_H ← reduce&transform(K − U)
        if w(S_H) > w(S \ U) then
            S ← S \ U ∪ S_H              // Extend solution from D-Core
            if φ+ · φ < 1 then
                φ ← φ+ · φ               // Consider smaller D-Core
        else if S_H is optimal then
            φ ← φ− · |U|/n(K)            // Consider larger D-Core
        else
            U ← V_K                      // Exact solver timed out
            if φ+ · φ < 1 then
                φ ← φ+ · φ               // Consider larger D-Core
    return S
```

combined with several state-of-the-art independent set solvers. Afterward, we analyze different versions of our DRP algorithm and, finally, compare the best variants of

our DRP approach with the best configurations of the state-of-the-art MWIS solvers combined with REDUCE&TRANSFORM.

Methodology. We implemented our algorithm using C++17. The code is compiled using g++ version 12.2 and full optimizations turned on (-O3). To investigate the allocated memory, we use the MALLOC_COUNT library[1]. For the experiments in this chapter, we used Machien 1. We ran all our experiments with four different random seeds and report geometric mean values unless mentioned otherwise. We set the time limit for all algorithms to 4h and a memory limit of 200 GB. In case of a timeout or memory limit is reached, we report the best solution found until this point.

Overview/Competing Algorithms. In our experiments, we first investigate the influence of the data reduction rules in Section 5.4.3. Therefore, we compare several MWIS solvers combined with different configurations of our REDUCE&TRANSFORM routine. In particular, we examine the performance of HILS by Nogueira et al. [172], HtWIS by Gu et al. [107] as well as KaMIS BnR by Lamm et al. [149] and the two new heuristics m^{2}wis+s, introduced in Section 4.3 and CHILS, presented in Section 4.4. In the second part of our experiments, we examine the performance of the heuristic DRP using the two MWIS algorithms KaMIS BnR and CHILS for solving the D-Core instances. Finally, these two configurations are compared to the best configurations for each MWIS solver combined with REDUCE&TRANSFORM in Section 5.4.3. Our experiments in this section were conducted using Instance Set 7.

Instance Set 7 (Weighted 2-Packing)**.** *Our experiments were conducted on* 205 *graphs, with at least* 1,000 *vertices. It contains 40* `snap` *[155], six* `ssmc` *[53], five* `fe` *[210], 14* `mesh` *[185], and 34 Open Street Map (*`osm`*) graphs [20, 1]. These instances are a subset of Instance Set 1. Further, we added 18 large graphs from Instance Set 6. We assign weights for these 18 graphs using five different weight distributions (uniform, geometric, hybrid, degree, and unit weights), which results in an additional 90 weighted instances. We extended the* `osm` *instances from Instance Set 1 with eight large-scale* `osm` *graphs from the 10th DiMACS challenge [190], assigning weights sampling from a uniform distribution and using hybrid weights. For a detailed overview of all graphs and weight distribution, see Table A.7 in the Appendix.*

Impact of Data Reductions on Instances

In this section, we analyze the efficiency of our reductions for preprocessing. First, we compare different configurations of our REDUCE&TRANSFORM routine. Then, we show

[1]The MALLOC_COUNT library can be found at https://github.com/bingmann/malloc_count.

Configuration	Reduction Order									Transformation
	fast			main						
FULL	[5.21	5.22	5.23	5.12	5.17	5.19	5.16	5.14	5.20]	reduced link-graph
FAST	[5.21	5.22	5.23						]	reduced link-graph
STRONG	[5.21	5.22	5.23	5.12		5.19	5.16	5.14	5.20]	reduced link-graph
MAIN	[			5.12	5.17	5.19	5.16	5.14	5.20]	reduced link-graph
TRANSFORM	[								]	full graph

Table 5.3: Overview of all REDUCE&TRANSFORM configurations evaluated in our experiments. Column Reduction Order presents, which reduction rules are used and their order.

how different solvers can benefit from our preprocessing routine. Table 5.4 presents the performance of REDUCE&TRANSFORM with different configurations. Each reduction configuration is defined via the reductions used and their order given in the corresponding reduction lists. In the FULL configuration, we first apply our fast reductions once, followed by our main reductions as presented in Table 5.3.

The intuition behind the chosen order for the main part is that we want to apply reductions that *include* vertices first since they reduce bigger parts of the graph. The Domination is placed before the D2-Simplicial Weight Transfer and Split Intersection Removal because it is a special case of the latter. If none of the latter reductions are applicable anymore, we try to exclude vertices and fold neighborhoods. To evaluate the impact of our fast data reductions, we split FULL into a FAST and a MAIN configuration, where FAST only uses the fast data reductions, and MAIN uses only those

	median [%]		geo. mean			
Configuration	$n(\mathcal{K})/n(G^2)$	$m(\mathcal{K})/m(G^2)$	t [s]	mem. [MB]	$o(\mathcal{K})$	#opt [%]
FULL	**0.1**	**0.000 2**	0.37	65.3	**312 963**	**44**
FAST	50.0	26.372 5	**0.18**	61.9	126 430	-
STRONG	**0.1**	**0.000 2**	0.34	**61.8**	312 962	**44**
MAIN	**0.1**	**0.000 2**	1.38	89.8	312 958	**44**
TRANSFORM	100.0	100.000 0	0.30	101.7	-	-

Table 5.4: Performance of REDUCE&TRANSFORM with different configurations. The set is restricted so that all configurations can create the transformed graph. The geometric mean offset $o(\mathcal{K})$ was computed only for instances where all variants found an offset larger than zero.

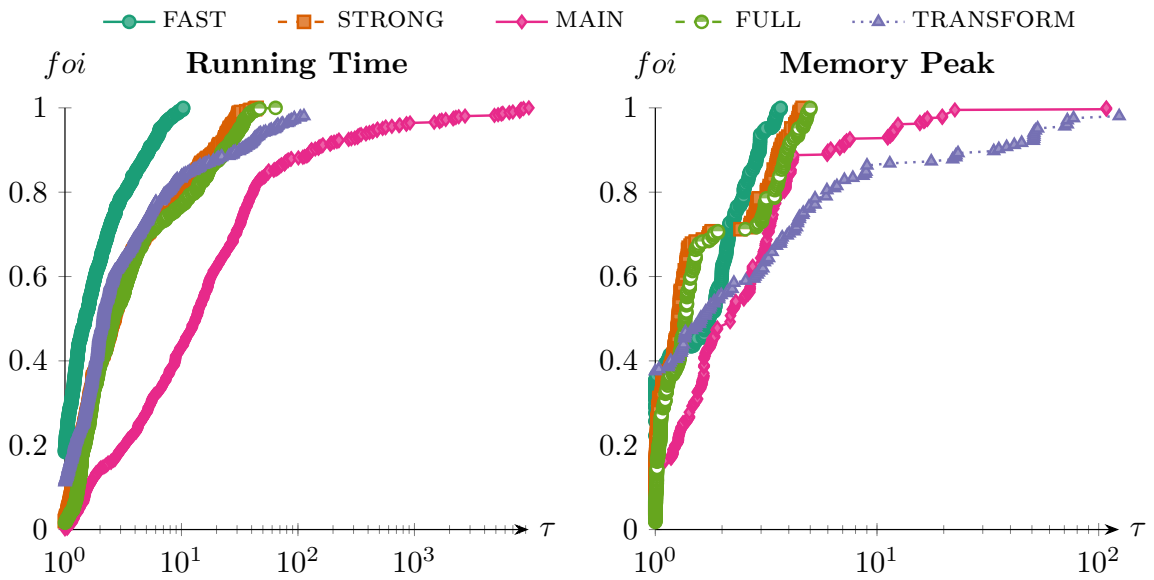


Figure 5.11: Performance profiles comparing different configurations of REDUCE&TRANSFORM, defined in Table 5.3. The left figure shows the running time to compute the (reduced) transformed graph, and the right figure presents the memory peak. The y-axis shows the corresponding fraction of instances (*foi*) solved faster or with less memory than τ times the best-performing algorithm on the respective instance.

from the main part except for the special case of neighborhood folding. Finally, we propose an equally effective but lighter configuration than FULL, called STRONG, that does not use Domination. Table 5.3 gives an overview of all configurations compared.

In Table 5.4, we report the sizes of the transformed graphs and the time and memory needed to compute them. Furthermore, we give the number of fully reduced graphs, i. e., the number of instances solved to optimality by our reductions, as well as the geometric mean reduction weight offset. To emphasize the efficiency of our proposed REDUCE&TRANSFORM configurations, we compare them with the configuration TRANSFORM that determines the square graph by building every 2-neighborhood. For an overview of all configurations compared, see Table 5.4.

Figure 5.11 shows that using FAST is always better than directly transforming the graph in all aspects regarding running time, graph sizes, and memory consumption. With this FAST configuration, we reduce the instances on average to less than 50 % of the original number of vertices while saving up to 39 % of memory and using less time. The results for the MAIN configuration clearly show the importance of our fast reductions. Compared to the FULL configuration, we can improve the running time by approximately a factor of 4 while reducing the memory consumption by 27 %. The STRONG configuration is the best performing regarding the size of the reduced

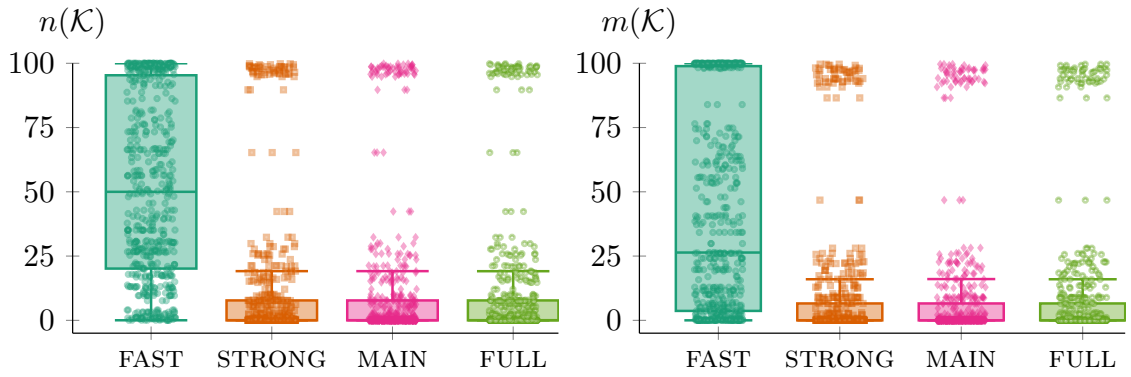


Figure 5.12: Comparison of reduced instances $\mathcal{K}$ computed by different reduction configurations defined in Table 5.3. On the right, we present the remaining number of vertices, and on the left, the remaining number of edges in % relative to the square graph.

transformed graph as well as memory consumption. Furthermore, on average, it uses only a factor of 1.88 more time than our fastest variant.

Figure 5.12 gives a more detailed insight into the reduction impact for our configurations regarding the reduced instance size: with our main reductions, we find for almost all instances a reduced instance with at most 25 % vertices and edges relative to the square graph. However, there are outliers left where our reductions are hardly applicable. Regarding the configuration FAST, we observe that its effectiveness varies highly. The first quantile is approximately at 24 % (5 %) remaining vertices (edges), the median at 50 % (27 %), and the third quantile at 90 % (95 %).

The configurations utilizing our main reductions can solve at least 43 % of the instances to optimality. Although FAST is not capable of reducing any instance fully, we still want to point out that when using fast and main reductions in conjunction (STRONG or FULL), there are instances where we can reduce running times from almost 3 hours down to less than 10 seconds as observed for snap_wiki-Talk-uniform. This graph has almost 74 % degree-one vertices, which are efficiently reduced with our Fast Degree-One Reduction, whereas MAIN and TRANSFORM are unable to find a (reduced) and transformed instance.

The performance profiles regarding the running time and memory peak in Figure 5.11 provide revealing results. In terms of running time, STRONG, FULL, and TRANSFORM perform very similarly. Almost 50 % of the runs are at most a factor of 2 slower than the fastest running times among all configurations on the respective instances. For 20 % of the runs, on the other hand, Figure 5.11 indicates that these are at least a factor of 10 up to 100 slower than the respective fastest running times. In the case of FAST, we notice that it is at most a factor of 10 slower on all runs while having the best running times on 20 % of the instances. The configuration MAIN performs

worst regarding running time. It is up to a factor of 10^4 slower than the fastest configuration. However, MAIN fully reduces 43 % of the instances while TRANSFORM only determines the square graph. In this sense, it is much to our surprise that STRONG and FULL can keep up with the performance of TRANSFORM while they are at least as good in solution quality as MAIN. The performance profiles regarding the memory peak clearly indicate that using our reductions is better than directly transforming the graph. Without reductions, TRANSFORM needs up to a factor of 10^2 more memory than the smallest achieved peaks, while our REDUCE&TRANSFORM configurations with(out) fast reductions are at most a factor 5 (20) apart from the smallest peaks.

Our experimental results underline the efficiency in terms of running time and memory consumption for FAST, STRONG, and FULL gained by using our fast reductions. Combining this preprocessing phase with our main reductions results in an effective reducing scheme that yields for 44 % of the instances optimal solutions, with median reduced instance sizes of 0.1 % vertices and 0.0002 % edges relative to the square graph. Moreover, TRANSFORM requires multiple orders of magnitude more memory and takes a similar amount of running time, making it inapplicable on some instances under reasonable resource limitations. Regarding the FULL configuration, it appears that the rule Domination has only a small positive impact compared to STRONG.

Observation 5.4.1: Reduction Impact on Instances. Considering the impact of reductions on the (reduced) transformed instance, our FAST configuration is always preferable to TRANSFORM. It is faster while requiring less memory than TRANSFORM. The configuration STRONG needs, on average, only 0.002 seconds more than TRANSFORM, but uses the least amount of memory and already solves 44 % of our instances optimally.

Impact of Data Reductions for Solving

In the following, we investigate the practical effect of the configurations FAST, STRONG, and TRANSFORM when solving the transformed graph with a wide range of MWIS solvers. All running times and memory presented in this section always include the time and memory used for REDUCE&TRANSFORM and the solving process combined. For the solution qualities, we always give the reduction offset and the solution quality found on the reduced and transformed instances found by the different solvers. Besides the exact solver KaMIS BnR, we add four heuristics in this comparison. We include the evolution-based solver M^{2}WIS+S and a reduce and peel solver HtWIS. Additionally, we have the local search algorithm HILS and the concurrent local search

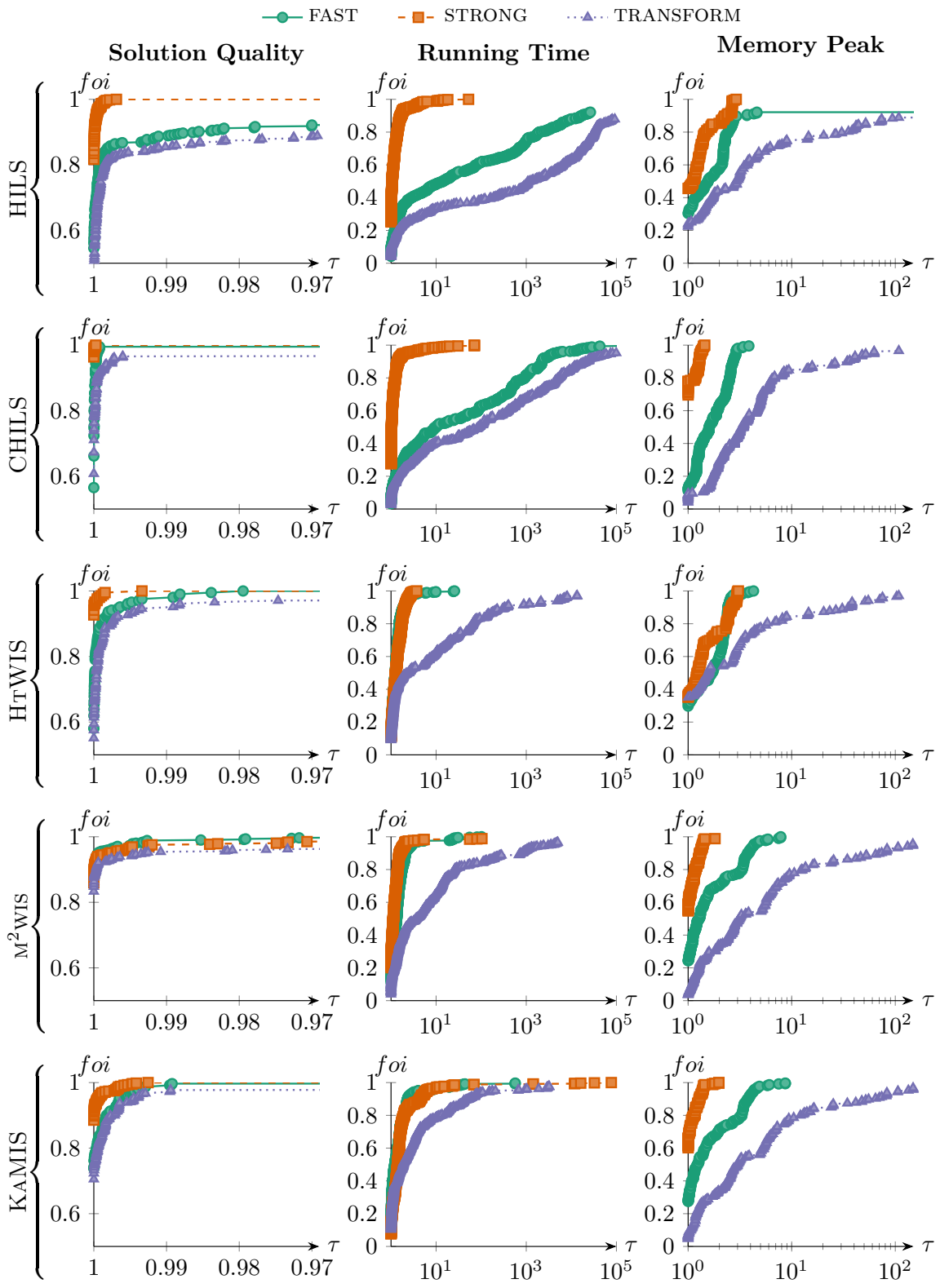


Figure 5.13: Performance profiles for reduction configuration comparison using different MWIS solvers (rows). Note that the y-axes—showing the fraction of instances (foi)—start at 0.5 for the solution quality. We present profiles for solution quality (left), running time to find the best solution (center), and the memory consumption (right).

MWIS solver	Configuration	$\omega(S)$	t	mem.	opt.	reached opt.	# add. sol.
HILS	STRONG	211 321	0.943	33.6	43.9	98.6	31
	FAST	211 222	20.291	39.2	0.0	74.6	15
	TRANSFORM	211 101	171.181	75.8	0.0	62.0	8
CHILS	STRONG	**211 445**	0.722	38.6	43.9	99.3	30
	FAST	211 438	23.638	61.0	0.0	67.6	30
	TRANSFORM	211 436	73.687	147.9	0.0	72.1	24
HtWIS	STRONG	210 582	**0.168**	**33.5**	43.9	76.8	**32**
	FAST	210 437	0.171	37.9	32.2	65.5	**32**
	TRANSFORM	210 418	1.107	70.0	29.3	61.3	26
m^2WIS	STRONG	211 296	0.889	45.3	59.5	**100.0**	26
	FAST	211 304	1.069	69.4	15.6	**100.0**	30
	TRANSFORM	211 299	5.310	205.3	15.1	98.6	22
KaMIS BnR	STRONG	209 822	0.369	50.4	**69.3**	**100.0**	30
	FAST	209 744	0.318	77.9	68.8	99.5	30
	TRANSFORM	209 742	0.767	225.4	67.8	97.9	26

Table 5.5: Performance of our reduction configurations in combination with various MWIS solvers. The columns $\omega(\mathcal{S})$, t and mem. show the geometric mean solution quality (including reduction offset), time found (in seconds, including REDUCE&TRANSFORM time), and memory peak (in MB, including REDUCE&TRANSFORM memory). These are evaluated on the set of instances where all solvers found a 2-packing set. The 'opt.' column shows the percentage of solutions proven optimal by the respective solver. The column 'reached opt.' shows the percentage of instances solved optimally among the instances where at least one solver proved optimality. The last column shows the number of additional feasible solutions computed by the respective solver compared to the common set of solutions. The **best** results are highlighted in bold.

CHILS. In Table 5.5, we compare the different reduction configurations and solvers. We present the geometric mean running time and solution quality. Moreover, we also report the number of additional instances solvable through our preprocessing, mainly due to memory issues with the original square graph. In Figure 5.13, we compare the different reduction configurations for each solver in more detail.

Table 5.5 shows that all algorithms benefit from using our preprocessing instead of the plain graph transformation. When comparing TRANSFORM and FAST, we see speedups of more than a factor of 8 (HILS) while still improving on solution quality. Additionally, the amount of memory needed is reduced by almost a factor of

two. These factors are further increased when comparing TRANSFORM to the STRONG variant. Here, we see speedups up to a factor of 180 (HILS) using even less memory than FAST and further improving on solution quality. The algorithm M^{2}WIS+S is the only one not benefiting from the STRONG variant improvements in solution quality. However, the FAST variant can improve running time, solution quality, and memory used. In this heuristic, additional independent set reductions are included, which is why it can increase the percentage of instances proven to be optimal from 43.9 %, which are the instances fully reduced by our STRONG reductions, to 59.5 %. The fastest and most memory-efficient configuration is STRONG combined with HTWIS. This configuration has the most additional feasible solutions. However, this comes with a loss in solution quality compared to the other heuristics, which we see by the small percentage of only 76.8 % of instances where it reached a (proven) optimal solution. In contrast to HTWIS, all other STRONG configurations reach 98.6 % or higher.

Regarding KAMIS BNR, we note that it often times out. In case of a timeout, the branch and reduce solver greedily builds a maximal solution for the remaining non-solved connected components.

In Figure 5.13, we see that for *all* solvers, the basic TRANSFORM is worse concerning solution quality, running time, and memory peak compared to both reduction configurations FAST and STRONG. With STRONG, the memory peak compared to TRANSFORM for all solvers tested on around 20 % of the instances is reduced by more than one order of magnitude. Especially for the local search algorithms, HILS and CHILS, we see an additional huge improvement in the running time when comparing STRONG with FAST. This speedup is up to multiple orders of magnitude, while the STRONG configurations also further improve solution quality and reduce the memory peak observed.

Observation 5.4.2: Reduction Impact on Solving. When considering the performance of different solvers on the (reduced) transformed instances while always adding the transformation time to the solve time, we see that the time needed for the STRONG configuration is well spent, and overall, these configurations find better solutions multiple magnitudes faster with less memory needed. Especially local search heuristics benefit from using the STRONG configuration.

The Algorithm redW2pack

In this set of experiments, we investigate the different configurations of our solver REDW2PACK. We present the parameters for the different configurations in Table 5.6. The parameters chosen worked best in preliminary experiments. Configurations with

	Configuration Details					Results		
Configuration	D-Core solver	t_H	ϕ	ϕ_+	ϕ_-	$\omega(S)$	t	mem.
DRP-BChils	CHILS Baseline	80	0.6	1.00	1.00	**446 750**	4.3	72.0
DRP-KaMIS	KaMIS BnR	80	0.8	1.05	0.95	446 687	**2.8**	78.9
DRP-no-core	-	-	-	-	-	445 710	3.8	**71.6**

Table 5.6: Geometric mean results of solution weight $\omega(S)$, time found t (in seconds), and memory peak mem. (in MB) for the different configurations of our algorithm DRP. The last configuration failed to finish solving for one instance within the set time limit. All configurations are evaluated on instances where all algorithms found a solution in time.

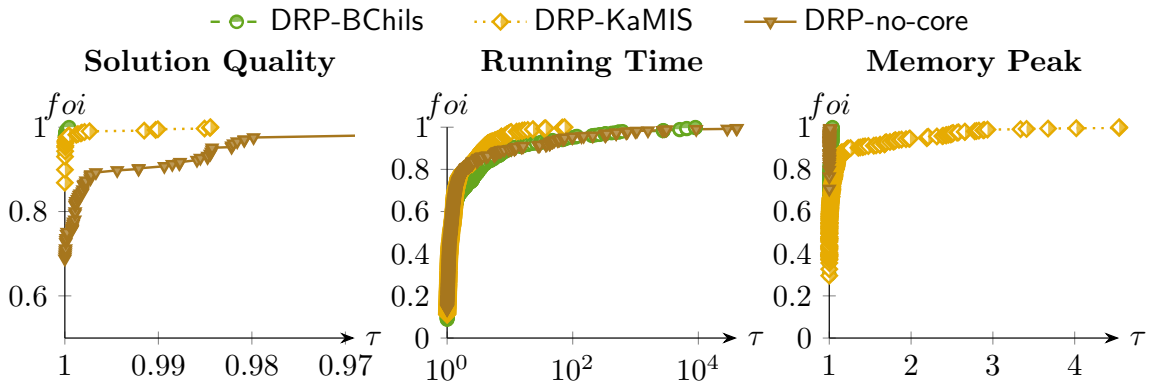


Figure 5.14: Performance profiles comparing the different configurations of DRP regarding solution quality (left), time for the best solution to be found (center), and the memory peak (right).

values for ϕ_- and ϕ_+ work well if they change ϕ only slightly. The intuition behind that is that if ϕ_- decreases ϕ too strong, it might need many new solutions to compute a large D-Core, which in the end might be hard to solve optimally. A good choice for the initial ϕ tends to yield D-Core instances of manageable size that the MWIS solver can solve in a reasonable time. Additionally to the parameters, we present the geometric mean results for different configurations of solution quality and time found in Table 5.6.

We compare solving the D-Core with the exact solver KaMIS BnR resulting in the configuration DRP-KaMIS and the heuristic baseline local search used in the CHILS algorithm, yielding DRP-BChils. Additionally, we present results without using the D-Core strategy with the variant DRP-no-core. Table 5.6 shows the configurations with the parameter values and the geometric mean solution quality, time found, and memory peak achieved by the configurations.

In Figure 5.14, we present performance profiles comparing the solution quality, time found, and memory peak for the different DRP configurations in more detail. Note that 44 % of the instances are fully reduced by the STRONG preprocessing and thereby solved optimally for all the configurations compared here. The two variants using the D-CORE perform better regarding solution quality. The configuration DRP-no-core computes on more than 30 % of the instances a worse solution than the other configurations. The configuration DRP-BChils performes best in terms of solution quality, while the configuration DRP-KaMIS has the fastest times to find the best solutions overall. It is counterintuitive that the configuration DRP-BChils is the best regarding solution quality, as the exact solver KAMIS BNR should be able to find better or equal solutions. However, the exact solver KAMIS BNR cannot solve all D-CORE instances to optimality within the time limit of 80 seconds. Since the baseline local search in CHILS is a very powerful algorithm, it computes near-optimal solutions fast. This explains why the configuration DRP-BChils is better regarding solution quality. Nevertheless, the local search approach always utilizes the full time limit to solve the D-CORE, resulting in worse performance regarding running time. On the other hand, the variant using KAMIS BNR can save time when computing the D-CORE solutions, as it continues as soon as the solution found is proven optimal. The configuration DRP-BChils is almost as memory efficient as the configuration DRP-no-core, while the approach using the exact solver DRP-KaMIS needs up to four times as much memory.

Observation 5.4.3: DRP Configurations. Our experiments for the different DRP configurations clearly show that using the D-CORE strategy can improve the solution quality and running time. When using DRP-KaMIS, the algorithm can find the best solutions on average a factor of 1.4 faster than DRP-no-core. On the other hand, the configuration DRP-BChils yields the best solution quality on average while being almost as memory efficient as DRP-no-core.

State-of-the-Art Comparison

Figure 5.15 shows performance profiles comparing the best-performing configurations for all solvers on all instances. We present comparisons of solution quality, running time, and memory peak. Table 5.7 summarizes these results for the different solvers.

We can observe that the configuration STRONG-CHILS performs overall the best regarding solution quality, while the DRP configurations are very close to this performance. The configuration STRONG-HTWIS is the fastest configuration with the overall lowest memory peak, but it performs worst regarding solution quality. The

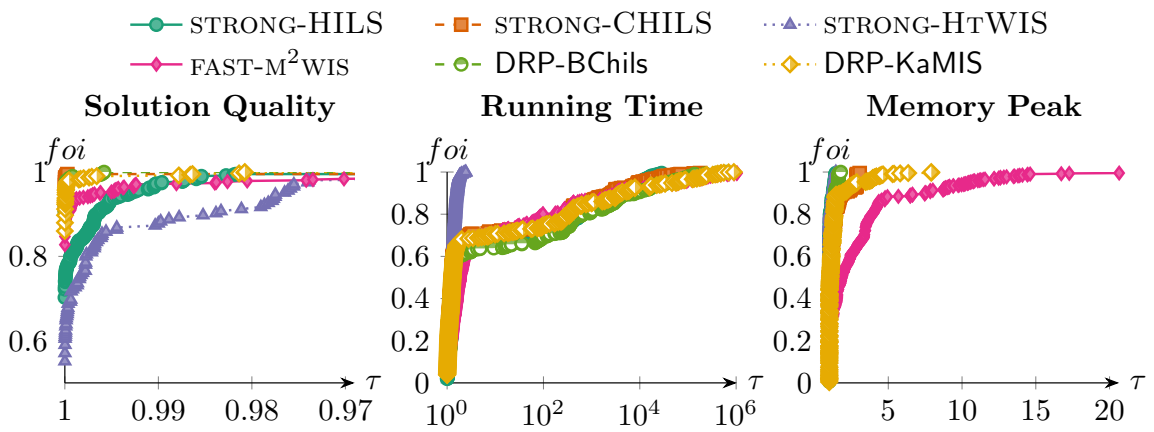


Figure 5.15: Performance profiles for the state-of-the-art comparison. Configurations of best performing REDUCE&TRANSFORM configurations and DRP configurations in terms of solution quality.

highest memory peak is observed for the configuration FAST-M²WIS. However, this is the only configuration that uses the FAST reduction routine, which means initially a bigger transformed reduced graph compared to the other approaches.

A detailed comparison of the different approaches on interesting instances is presented in Table 5.8. Here, we present per instance results for the 40 largest instances where the performance of the algorithms differed most regarding either solution quality or running time. Looking at different graph classes, we see that FAST-M²WIS performs particularly well for different `osm` instances. This observation can be explained by the fact that M²WIS has several reductions for the MWIS implemented, which work well on these `osm` instances. For instances where these reductions do not work well, e. g., `snap` or `mesh`, the STRONG-CHILS approach excels. Furthermore, the different weight distributions for the instances do not significantly affect the difference in performance between the algorithms. For example, for every weight distribution, the instance road_usa— the second largest instance in our data set—is always solved best by DRP-BChils.

Observation 5.4.4: State-of-the-Art Comparison. In the comparison between the different DRP configurations and REDUCE&TRANSFORM combined with different state-of-the-art MWIS solvers, we can see that both DRP configurations can keep up with the state-of-the-art MWIS solvers combined with REDUCE&TRANSFORM. Additionally, on some of the biggest instances (europe and road_usa), DRP-BChils and DRP-KaMIS can find the best solution quality, outperforming all MWIS approaches.

Solver	$\omega(S)$	t	mem.	opt.	reached opt.	# add. sol.
strong-HILS	441 458	2.88	68	**44**	**100**	3
strong-CHILS	**441 885**	2.09	78	**44**	99	2
strong-HtWIS	440 202	**0.34**	**67**	**44**	85	**4**
fast-m^{2}wis+s	441 454	2.89	140	16	**100**	2
DRP-BChils	441 864	4.36	72	**44**	**100**	**4**
DRP-KaMIS	441 801	2.82	79	**44**	**100**	**4**

Table 5.7: Performance of our reduction configurations in combination with various MWIS solvers. The columns $\omega(\mathcal{S})$, t and mem. show the geometric mean solution quality, time found (in seconds), and memory peak (in MB). These are evaluated on the set of instances where all solvers found a 2-packing set. Note that the solvers here are evaluated on a larger set than those considered in Table 5.5. The 'opt.' column shows the percentage of solutions proven optimal by the respective solver. The column 'reached opt.' shows the percentage of instances solved optimally among the instances where at least one solver proved optimality. The last column shows the number of additional feasible solutions found by the respective solver compared to the common set of solutions. The **best** results are highlighted in bold.

Table 5.8: We present detailed results for state-of-the-art comparison on interesting instances. For that, we picked the 40 largest graphs (in terms of vertices) where at least one solver has a worse solution quality than a factor 0.98 or needs at least a factor 100 more running time than the average for computing the best solution on that instance. We mark cells with "-" if a timeout occurs (column t) and if no feasible solution was computed until the timeout (column ω). The instances are grouped by their graph class and have different weight assignments uniform (uf), unit (u), hybrid (h), geometric (g), degree (d), or from the file (f).

			DRP-BChils		DRP-KaMIS		strong-CHILS		strong-HILS		strong-HtWIS		fast-m^{2}wis+s	
	Instance		ω	t	ω	t	ω	t	ω	t	ω	t	ω	t
mesh	buddha	uf	32 602 416	14 135.22	32 602 593	14 203.88	**32 603 831**	13 875.78	32 021 963	14 393.57	31 880 965	**4.57**	31 447 350	14 391.90
mesh	dragonsub	uf	18 464 865	14 199.63	18 464 623	14 355.58	**18 466 821**	13 023.43	18 324 835	14 395.13	17 987 255	**1.89**	17 499 416	-
mesh	ecat	uf	20 963 870	14 250.79	20 963 796	13 993.82	**20 965 446**	13 288.50	20 773 748	14 393.50	20 452 008	**5.61**	19 600 785	14 391.57
snap	as-skitter	uf	**22 747 281**	309.97	**22 747 281**	1 944.01	**22 747 281**	**24.06**	**22 747 281**	58.17	22 745 994	27.45	**22 747 281**	233.67
snap	com-amazon	f	6 784 132	1 345.69	**6 784 135**	673.02	**6 784 135**	161.71	6 783 500	528.77	6 779 497	**1.89**	**6 784 135**	5.65
snap	roadNet-CA	uf	69 533 118	13 506.27	69 533 078	13 547.56	**69 533 187**	7 500.36	69 441 335	14 392.67	69 298 011	**6.07**	69 531 967	7 944.87
snap	roadNet-PA	f	38 549 128	10 606.64	38 549 112	12 495.52	**38 549 235**	4 182.21	38 526 182	14 374.67	38 428 331	**3.17**	38 549 060	8 404.16
snap	roadNet-PA	uf	38 606 226	7 430.02	38 606 192	13 674.26	**38 606 300**	5 360.57	38 584 253	14 307.96	38 483 203	**2.70**	38 606 010	4 720.88
snap	roadNet-TX	uf	49 698 766	13 048.85	49 698 796	13 072.83	**49 698 835**	11 418.44	49 671 265	14 357.58	49 557 120	**3.87**	49 698 111	8 575.76
snap	soc-pokec-rel.	uf	24 557 118	7 946.46	**24 557 259**	394.37	24 557 109	6 862.02	24 552 172	12 212.59	24 541 657	**25.58**	24 557 199	4 671.27
snap	web-BerkStan	f	3 993 862	1 121.93	**3 993 879**	415.52	**3 993 879**	337.15	3 993 828	423.79	3 992 470	**158.42**	**3 993 879**	13 022.76
snap	web-BerkStan	uf	3 997 024	682.55	**3 997 026**	238.33	3 996 974	358.62	3 996 962	319.13	3 995 734	**210.57**	**3 997 026**	12 291.58
snap	web-Google	uf	**10 993 272**	39.57	**10 993 272**	2.68	**10 993 272**	**2.57**	**10 993 272**	2.82	10 993 265	3.10	**10 993 272**	5.59
ssmc	ca2010	f	9 098 481	8 025.21	9 098 171	14 083.31	**9 098 531**	2 094.41	9 091 051	13 683.52	9 010 605	**4.75**	9 097 549	13 037.78
ssmc	fl2010	f	4 655 766	8 275.04	4 655 665	12 814.44	**4 655 789**	1 009.34	4 654 381	12 920.20	4 635 302	**2.52**	4 655 757	10 420.59
ssmc	il2010	f	3 231 622	11 699.56	3 231 436	13 985.54	**3 231 632**	775.99	3 228 123	14 182.51	3 192 156	**3.46**	3 231 242	10 609.37

Continued on next page

Table 5.8 – *Continued from previous page*

	Instance		DRP-BChils ω	DRP-BChils t	DRP-KaMIS ω	DRP-KaMIS t	strong-CHILS ω	strong-CHILS t	strong-HILS ω	strong-HILS t	strong-HtWIS ω	strong-HtWIS t	fast-m^{2}wis+s ω	fast-m^{2}wis+s t
osm	asia	h	104 862 636	13 866.77	**104 862 684**	14 276.46	104 862 634	13 853.99	104 244 901	14 396.80	104 629 705	**23.49**	104 860 823	13 520.92
	asia	uf	124 441 135	8 053.17	124 441 141	8 195.60	124 441 137	4 428.06	124 425 250	14 384.99	124 403 501	**17.49**	**124 441 143**	222.29
	belgium	h	12 625 769	9 686.40	12 625 796	7 404.59	12 625 751	11 272.41	12 607 633	14 381.66	12 581 266	**3.52**	**12 625 797**	202.48
	belgium	uf	**14 853 069**	1 341.68	**14 853 069**	46.18	**14 853 069**	162.34	14 852 592	2 039.53	14 851 095	**2.38**	**14 853 069**	6.22
	europe	h	448 902 095	14 250.24	**448 903 472**	14 343.86	448 899 155	14 262.91	-	-	447 920 680	**154.00**	448 883 604	-
	europe	uf	529 717 578	10 019.66	529 717 576	10 906.18	529 717 579	10 408.97	529 286 469	14 397.53	529 637 587	**110.99**	**529 717 599**	562.93
	germany	uf	**119 677 619**	523.92	**119 677 619**	233.19	**119 677 619**	314.74	119 674 575	14 194.11	119 665 411	**25.68**	**119 677 619**	51.24
	germany	h	102 376 273	13 735.38	102 376 346	12 480.37	102 375 878	14 196.11	101 390 183	14 397.53	102 139 965	**30.19**	**102 376 351**	1 630.99
	great-britain	h	69 224 797	11 081.76	69 224 825	7 045.04	69 224 786	12 173.27	69 018 599	14 397.00	69 115 568	**15.57**	**69 224 827**	551.96
	great-britain	uf	**81 011 319**	423.07	**81 011 319**	62.40	**81 011 319**	36.63	81 010 241	10 312.23	81 005 926	**12.05**	**81 011 319**	25.25
	italy	h	58 703 499	12 351.94	58 703 552	12 163.02	58 703 514	12 189.68	58 547 438	14 397.23	58 592 323	**13.10**	**58 703 583**	6 990.41
	italy	uf	**69 970 692**	579.19	**69 970 692**	859.43	**69 970 692**	168.25	69 968 413	13 695.05	69 960 840	**9.72**	**69 970 692**	35.02
	netherlands	h	19 781 545	10 011.18	19 781 556	3 876.05	19 781 549	9 415.14	19 731 383	14 395.07	19 718 874	**4.66**	**19 781 557**	314.71
	netherlands	uf	**22 599 881**	566.92	**22 599 881**	79.47	**22 599 881**	8 567.77	22 598 638	11 823.36	22 593 996	**4.33**	**22 599 881**	16.92
red2pack	road_central	d	13 499 922	14 179.00	13 499 758	14 063.58	**13 499 960**	14 238.69	13 457 810	14 395.20	13 470 102	**35.75**	13 492 263	14 392.47
	road_central	g	**235 169 677**	14 395.97	235 161 329	14 342.85	235 165 168	14 308.29	232 195 767	14 397.57	234 241 460	**54.75**	234 484 588	14 393.77
	road_central	h	549 590 890	14 148.78	549 590 464	14 075.05	**549 591 087**	13 750.86	547 901 981	14 397.23	548 981 006	**52.93**	549 582 921	12 895.80
	road_central	u	8 999 961	13 959.77	8 999 827	14 161.04	**8 999 975**	14 013.27	8 971 625	14 396.07	8 980 068	**39.51**	8 994 753	14 392.30
	road_central	uf	**8 999 967**	13 923.06	8 999 843	13 553.04	**8 999 967**	14 262.95	8 971 109	14 397.33	8 980 068	**39.49**	8 995 065	14 392.30
	road_usa	d	**22 946 461**	14 270.94	22 946 188	14 249.10	22 946 259	14 276.51	22 835 809	14 396.93	22 894 944	**40.14**	22 924 695	14 392.93
	road_usa	g	**399 716 160**	14 396.80	399 702 024	14 379.61	399 705 895	14 332.03	394 096 678	14 395.93	398 129 609	**58.87**	397 709 419	14 397.27
	road_usa	h	**909 315 888**	14 298.16	909 311 627	14 386.40	909 311 190	14 289.75	902 975 074	14 396.27	907 736 821	**59.81**	909 070 721	14 394.97
	road_usa	u	**15 297 660**	14 263.56	15 297 438	14 274.22	15 297 510	14 295.70	15 225 993	14 397.23	15 263 296	**41.75**	15 283 063	14 393.43
	road_usa	uf	**15 297 632**	14 338.10	15 297 465	14 262.34	15 297 512	14 289.57	15 224 282	14 396.47	15 263 296	**41.55**	15 282 400	14 393.13

5.5 Conclusion

This chapter introduces novel data reduction rules for the Maximum 2-Packing Set and the Maximum Weight 2-Packing Set problem. For both problems, we can use the reduction to the Maximum (Weight) Independent Set problem. We compute high-quality and optimal solutions by using our preprocessing routine reduce&transform combined with the corresponding data reduction rules and maximum (weight) independent set solvers.

In detail, these approaches work in three phases. First, the new data reduction rules are applied to the input graph, resulting in a reduced link-graph. Following the reduction phase, this reduced link-graph is transformed, such that a solution on the transformed graph for the Maximum Independent Set problem corresponds to a solution of the Maximum 2-Packing Set problem for the original graph. The third phase of the method consists of solving the Maximum (Weight) Independent Set problem on the transformed graph.

In the cardinality case, we evaluate our new exact algorithm red2pack_b&r that uses these reductions to exactly solve the Maximum 2-Packing Set problem on large-scale arbitrary graphs and our new heuristic red2pack_heuristic. Our experiments show that these methods outperform the previous best algorithm for arbitrary graphs regarding solution quality and running time on all instances. For instance, we can compute optimal solutions for 63% of the data set in under a second. In contrast, the competing method for arbitrary graphs achieves this solution quality only for 5% of the instances, even with a ten-hour time frame. Furthermore, our method successfully solves many large instances that remained unsolved before. Lastly, our algorithm can compete with a specialized solver on planar instances regarding solution size and computes near-optimum solutions.

For the weighted problem, we evaluate reduce&transform using several state-of-the-art independent set solvers. Our experiments show that our reductions can fully reduce and thereby optimally solve 44 % of the instances in our data set. Furthermore, reduce&transform improves solution quality, running time, and memory consumption compared to a transformation for every independent set solver tested. With reduce&transform, we achieve speedups compared to the naive reduction to independent set up to multiple orders of magnitude. Moreover, we propose a new heuristic DRP, a reduce and peel approach based on the metaheuristic Concurrent Difference Core Heuristic introduced in Section 4.4. Especially for large graphs DRP excels. Our experiments indicate that this heuristic is able to keep up with the state-of-the-art independent set solvers equipped with our preprocessing routine re-

DUCE&TRANSFORM. Additionally, DRP can find the best solution on the biggest instances in our data set, outperforming all independent set approaches.

For future work, we want to extend the set of reductions for the M2PS and the MW2PS further. Additionally, since the iterated local search approaches for the MAXIMUM WEIGHT INDEPENDENT SET problem work very good, we are interested in engineering a new local search heuristic for the 2-packing set problems to fully circumvent any graph transformation, thereby saving even more running time and memory. We are also interested in the k-packing set problem for larger values of k and find independent motifs in graphs via hypergraph matching algorithms. Our code is publicly available under https://github.com/KarlsruheMIS/red2pack.

Chapter 6

Hypergraph-b-Matching

Recently, the matching problem has been extended to the more general problem of finding b-matchings in hypergraphs, broadening the scope of potential applications and challenges. The concept of b-matchings, where b is a function that assigns positive integers to the vertices of the graph, is a natural extension of matchings in graphs, where each vertex v is allowed to be matched to up to $b(v)$ edges. The weighted b-matching problem seeks to select a subset of hyperedges that fulfills the constraint and maximizes the weight. In this chapter, we engineer novel algorithms for this generalized problem. More precisely, in Section 6.1, we introduce new exact data reductions for the problem, followed by different approaches to compute high-quality hypergraph b-matchings in Section 6.2. The experimental evaluation in Section 6.3, on a wide range of real-world hypergraphs, shows that our new data reductions are highly practical, and our initial solutions are competitive on graphs and hypergraphs.

References. *This chapter is based on joined work with Felix Joos, Henrik Reinstädtler and Christian Schulz [100]. This paper is currently in submission. Large parts of this chapter are copied verbatim from this paper.*

One of the most well-known problems in graph theory is the matching problem. A matching in a graph is a set of pairwise vertex-disjoint edges. Computing (these) matchings in a graph is a ubiquitous combinatorial problem with many applications in various fields [110]. By now, maximum weight/cardinality matchings in graphs in the internal-memory model have been extensively studied, leading to various breakthroughs. However, finding maximum (weight) matchings in graphs in the internal-memory model is only the tip of the iceberg. Recently, researchers have extended the concept of matchings to the more general problem of finding b-matchings in hypergraphs, broadening the scope of potential applications and challenges. A hypergraph is a natural graph extension in which edges can have more than two endpoints and

thus model more complex relationships. For example, online hypergraph matching can be used to model auctions of advertisement campaigns [141]. Moreover, the concept of b-matchings (with $b : V \to \mathbb{N}$) is a natural extension of matchings, where each vertex v is allowed to be matched to up to $b(v)$ edges, rather than just one. The weighted b-matching problem then seeks to select a subset of the hyperedges that fulfill the constraint and maximize the weight.

Applications. Many applications require the computation of a matching $\mathcal{M}$ with certain properties, like being maximal (no edge can be added without violating the matching property), having maximum cardinality, or having maximum total weight $\sum_{e \in \mathcal{M}} \omega(e)$. For example, in multi-level (hyper)graph partitioning, the problem of coarsening a (hyper)graph without losing the characteristics of the original (hyper)graph in multi-level decomposition algorithms can be solved by computing a hypergraph matching problem [191]. Similarly, hypergraph b-matching plays a critical role in agglomerative hypergraph clustering [174], where hyperedges are evaluated based on the likelihood of merging adjacent clusters. In this context, the function b assigned to the vertices can serve as a mechanism to regulate the pace of agglomeration. Other important example applications include allocating resources to machines or auctioning goods [51], ride-sharing [176] and load balancing [124]. Currently, however, researchers have only developed approximation algorithms [171] for the weighted case, and practical implementations of heuristics to tackle the hypergraph matching problem are limited to special classes of hypergraphs without weight [65].

Our Results. In this chapter, we devise and engineer new exact data reduction rules as well as new greedy and local search algorithms for the weighted hypergraph b-matching problem in general hypergraphs. While our main focus is on the most general weighted hypergraph b-matching problem, we also compare our greedy algorithms on graphs against solvers that do not work on hypergraphs. Our experiments show that we are able to obtain better initial solutions by greedy heuristics of up to 10 %, a speedup of 6.85 for exactly solving the hypergraph b-matching problem, and quality improvements of up to 30 % by our local search algorithm for the 1-matching case.

6.1 Exact Data Reduction Rules

Only a few exact data reduction rules are known that can be used for the hypergraph matching problem [65]. These data reductions are based on Karp-Sipser rules and are 1) not applicable to the weighted problem and 2) do not apply to the more

general b-MATCHING problem in hypergraphs. However, applying exact data reductions is a very important technique to decrease the problem size, especially for large instances. In the following, we introduce a large set of new reductions for the weighted b-matching problem on hypergraphs. In contrast to the reductions presented for the MWIS and MW2PS problems, this chapter presents reductions focused on both vertices and weighted edges. The reduction rules determine whether a vertex or hyperedge can be safely removed or if a hyperedge is guaranteed to be in an optimum b-matching and thus can be added to our solution. As in the previous chapters, some reductions use the concept of folding. However, in the hypergraph context, we combine edges to reduce the hypergraph size. The decision for these edges only depends on whether the edge they are folded into is part of the solution or not. In the following, we denote a hypergraph by H and the reduced hypergraph by H'. The sets $\mathcal{M}$ and $\mathcal{M}'$ are the solutions on the original and reduced instance, respectively. The reduction rules are introduced following the scheme used for the MWIS reductions described in Section 4.1.

The first reduction is the removal of abundant vertices. A vertex $v \in V$ is considered *abundant* if its capacity $b(v)$ equals or exceeds its degree. Whenever the degree of a vertex is smaller or equal to its capacity, the vertex can be removed.

Reduction 6.1 (Abundant Vertices and Empty Edges)

Let $v \in V$ be an abundant vertex, then exclude v.

Reduced Graph	$H' = H - v$
Offset	$\alpha_{\mathcal{M}}(H) = \alpha_{\mathcal{M}}(H')$
Reconstruction	$\mathcal{M} = \mathcal{M}'$

Let $e \in E$ be an empty edge, then include e.

Reduced Graph	$H' = H - e$
Offset	$\alpha_{\mathcal{M}}(H) = \alpha_{\mathcal{M}}(H') + \omega(e)$
Reconstruction	$\mathcal{M} = \mathcal{M}' \cup \{e\}$

Proof. An abundant vertex v can be removed from the hypergraph as v does not restrict the selection problem since all incident edges of v could be contained in an optimum matching as the capacity is larger than the number of adjacent edges. Thus, we can remove v from the hypergraph. If there is an empty edge $e \in E$, it is part of an optimal solution since it cannot be blocked at any other vertex. □

The next data reduction rule is inspired by Reduction 4.10 for the independent set problem. It adds edges to the solution that satisfy a local upper bound for the solution in its neighborhood.

Algorithm 20: Neighborhood Removal

Data: hypergraph $H = (V, E, \omega)$, capacity function b

Result: reduced hypergraph and offset

Reduction `NeighborhoodRemoval(`H, b`):`

- **for** $v \in V$ **do**
  - $checked \leftarrow 0$
  - **for** $e \in E(v)$ ordered by weight desc **do**
    - $checked \leftarrow checked + 1$
    - **if** $checked > b(v)$ **then**
      - **break**
    - $\omega_d \leftarrow 0$
    - **for** $u \in e$ **do**
      - $\omega_d \leftarrow \omega_d + \operatorname{nmax}(\hat{\omega}\left(E(u) \setminus \{e\}\right), b(u))$
      - **if** $\omega_d > \omega(e)$ **then**
        - **break**
    - **if** $\omega_d \leq \omega(e)$ **then**
      - // Include e and reduce capacity of vertices in e
      - **for** $u \in e$ **do**
        - $b(u) \leftarrow b(u) - 1$
      - $H, \alpha_{\mathcal{M}} \leftarrow H - e \cup blockedEdges(e), \alpha_{\mathcal{M}} + \omega(e)$
- **return** H, b, 0

Reduction 6.2 (Neighborhood Removal. Figure 6.1)
Let $e \in E$ be an edge with $\omega(e) \geq \sum_{v \in e} \operatorname{nmax}(\hat{\omega}(E(v) \setminus \{e\}), b(v))$, i. e., the edge has a higher weight than the total sum of weights of the $b(v)$-th heaviest edge (excluding e) in each of its vertices v. Then, we include e.

Reduced Graph	$H' = H - e \cup blockedEdges(e)$ *and*
	set $b(v) = b(v) - 1 \ \forall v \in e$
Offset	$\alpha_{\mathcal{M}}(H) = \alpha_{\mathcal{M}}(H') + \omega(e)$
Reconstruction	$\mathcal{M} = \mathcal{M}' \cup \{e\}$

Proof. Let $\mathcal{M}$ be an optimal solution and $e \in E$ with the above property. Assume $e \notin \mathcal{M}$, then we show that we always find an equal or better solution that includes the edge e, which proves the correctness of the reduction. For each $v \in e$ with $b(v) = |\mathcal{M}(v)|$ (conflicting vertices), we remove the lightest incident edge that is in the solution. Let

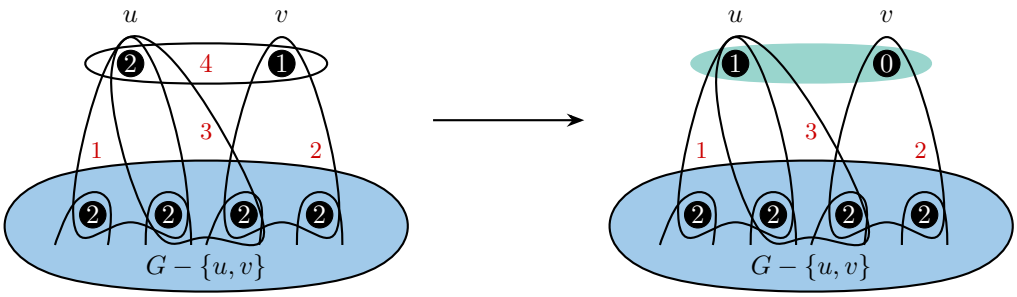


Figure 6.1: Illustration for Neighborhood Removal, see Reduction 6.2. We present the edge weights as red and vertex capacities as white numbers. The green edge is included and removed from the hypergraph. Note that now the capacity of v is zero, and therefore, the edge containing it can be removed as well.

this matching be $\mathcal{M}'$. It follows for all vertices $v \in e$: $|\mathcal{M}'(v)| < b(v)$. This implies that $\mathcal{M}'' \coloneqq \mathcal{M}' \cup \{e\}$ is also a valid matching. We now show $\omega(\mathcal{M}'') \geq \omega(\mathcal{M})$. Let e'_v be a lightest edge removed from $\mathcal{M}$ at v. Its weight $\omega(e'_v)$ contributing to $\mathcal{M}$ is smaller or equal to the $b(v)$ heaviest edge incident to v since $b(v)$ edges have been in $\mathcal{M}$. Thus, $\omega(e'_v) \leq \operatorname{nmax}(\hat{\omega}(E(v) \setminus \{e\}), b(v))$. The weight of all edges removed from $\mathcal{M}$ is smaller or equal to $\omega(e)$ if the equation holds. This yields $\omega(\mathcal{M}'') \geq \omega(\mathcal{M})$. □

In Algorithm 20 we present pseudocode for Reduction 6.2. We iterate over each vertex and check up to $b(v)$ incident edges. For each edge e, we calculate the sum of weights it needs to dominate and break early if the condition cannot be satisfied anymore. If we find a candidate, we can include it as part of an optimal matching and update the hypergraph and capacity accordingly. The time complexity for this algorithm is $\mathcal{O}(\min(n\beta, m)\Delta_E + n\Delta_V \log \Delta_V)$. At each vertex, we have to check up to $\beta = \max_{v \in V} b(v)$ edges, and using a map for skipping already checked edges, we have in total up to $\mathcal{O}(\min(n\beta, m))$ candidates. Since we keep the edge vector in each vertex sorted, the nmax operation runs in $\mathcal{O}(1)$. This initial sorting requires $\mathcal{O}(n\Delta_V \log \Delta_V)$ steps. Note that we can find multiple reductions in the same pass.

Inspired by the concept of isolated vertices and Reduction 4.17 for the MWIS problem, we introduce the following reduction rule based on the concept of isolated edges. An edge is isolated if it has the highest weight in its neighborhood, i. e., $\omega(e) \geq \max_{f \in \mathcal{N}(e)} \omega(f)$, and all neighbors have a common vertex with capacity 1. In this case, there is an optimal solution including this edge, as shown in the following reduction.

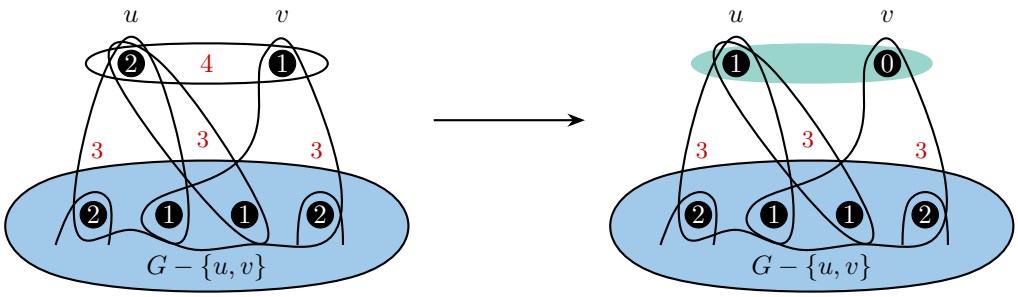


Figure 6.2: Illustration of Weighted Isolated Edge Removal, see Reduction 6.3. We present the edge weights as red and vertex capacities as white numbers. The green edge is included and removed from the hypergraph. Here, Reduction 6.2 is not applicable but with Reduction 6.3 the edge containing u and v can be removed and the vertex capacities decreased.

Reduction 6.3 (Weighted Isolated Edge Removal. Figure 6.2)

Let $e \in E$ be an isolated edge, then include e.

Reduced Graph	$H' = H - (\{e\} \cup blockedEdges(e))$ *and*
	$b(v) = b(v) - 1 \; \forall v \in e$
Offset	$\alpha_{\mathcal{M}}(H) = \alpha_{\mathcal{M}}(H') + \omega(e)$
Reconstruction	$\mathcal{M} = \mathcal{M}' \cup \{e\}$

Proof. Because e and all its adjacent edges contain at least one vertex with capacity 1, we can select at most one edge in $\mathcal{N}(e)$ while excluding all other edges in this neighborhood. An optimal solution $\mathcal{M}$ must contain at least one edge of $\mathcal{N}(e)$. Otherwise, the edge e is free and can directly be added, yielding a heavier matching, which is a contradiction to the optimality. Let therefore f be the neighbor contained in $\mathcal{M}$, then $\mathcal{M} \setminus \{f\} \cup \{e\}$ is also optimal since $\omega(e) \geq \max_{f \in \mathcal{N}(e)} \omega(f)$. This shows that there always is an optimal solution that includes e, and therefore, the reduction is correct. □

In Algorithm 21, we give a procedure to detect isolated edges and apply the reduction efficiently in detail. We scan the heaviest edge at each vertex if it has not been scanned before. We simultaneously check whether it has maximum weight at each other vertex and collect its neighbors. After checking if the weight condition is satisfied, we mark each edge incident to the currently scanned edge also as scanned. We can do this because all neighbors that weigh less than the currently scanned edge cannot be candidates themselves for this reduction at other vertices later. For neighbors with capacity 1 vertices, we save the position we scanned this edge in a binary encoding. We add neighboring edges with higher capacity vertices in a vector N_l. Afterward, we check if all neighbors are incident to a vertex with capacity 1 (property

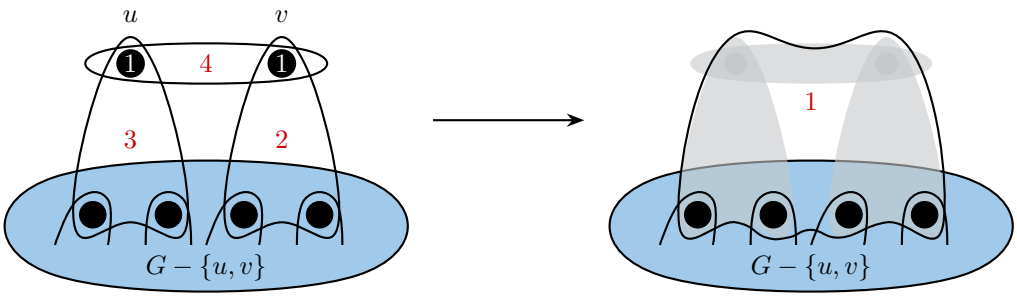


Figure 6.3: Illustration of Weighted Edge Folding, see Reduction 6.4. We present the edge weights as red and vertex capacities as white numbers. The edge containing u and v has exactly two non-adjacent neighbors that it dominates one by one but not in total.

S) and whether all pairs have a (blocking) common vertex of capacity 1. With the binary encoding, we can reduce the number of checks required. This is accomplished by a *bitwise and* of the encodings in N_b. If the result is non-zero, we have to do a detailed check. The overall complexity of this reduction is $\mathcal{O}(\min(m,n)\Delta_E{}^2)$ because in the worst case, we collect Δ_E distinct neighbors at a vertex with $b(v) = 1$ that we have to check for a common vertex.

The previous data reductions work by removing vertices (respectively, edges) from the graph. The following reduction modifies the structure of the hypergraph and postpones decisions to a later point.

Similar to Reduction 4.13 for the MWIS, where vertices get folded with their neighborhood, we introduce a reduction that folds edges with their adjacent edges.

Reduction 6.4 (Weighted Edge Folding. Figure 6.3)

Let $e \in E$ be an edge and $N = \mathcal{N}(e) \setminus \{e\}$ be the edges adjacent to e. Suppose the following holds:

1. *Each edge in N is linked to e via a vertex v with capacity $b(v) = 1$,*
2. *N is independent, i. e., the vertices in all distinct edges $f, g \in N$ are disjoint,*
3. *$\omega(N) > \omega(e)$, but it holds $\omega(e) > \omega(N) - \min_{f \in N} \omega(f)$.*

Then, fold the edge e and its adjacent edges N into a new edge e'.

Reduced Graph	$H' = H[(E \cup \{e'\}) \setminus (N \cup \{e\})]$, *set* $\omega(e') := \omega(N) - \omega(e)$ *and* $e' = \bigcup_{f \in N} f$.
Offset	$\alpha_{\mathcal{M}}(H) = \alpha_{\mathcal{M}}(H') + \omega(e)$
Reconstruction	*If* $e' \in \mathcal{M}'$, *then,* $\mathcal{M} = \mathcal{M}' \cup N$, *else* $\mathcal{M} = \mathcal{M}' \cup \{e\}$.

Algorithm 21: Weighted Isolated Edge Removal

Data: hypergraph $H = (V, E, \omega)$, capacity function b

Result: reduced hypergraph and offset

Reduction `WeightedIsolatedEdgeRemoval(`H, b`)`**:**

- **for** $v \in V$ with $b(v) = 1$ **and** not scanned before **do**
  - $e \leftarrow \text{argmax}_{e \in E(v)} \omega(e)$
  - N_b, N_l, *count*, *reducable* $\leftarrow \{\}$, $\emptyset$, 0, $True$
  - **for** $u \in e$ **do**
    - **if** $\omega(e) < \max f \in E(u) \omega(f)$ **then**
      - *reducable* $\leftarrow False$
      - **break**
    - mark edges $E(u)$ as scanned
    - **if** $b(u) = 1$ **then**
      - **for** $f \in E(u)$ **do**
        - $N_b[f] = N_b[f] + 2^{count}$
      - *count* $\leftarrow$ *count* $+ 1$
    - **else**
      - $N_l \leftarrow N_l \cup E(u)$
  - $S \leftarrow N_l \subseteq N_b$ // By checking $\forall f \in N_l \colon N_b[f] > 0$
  - **for** $f, g \in N_b$ **do**
    - **if** $N_b[g] \& N_b[f] = 0$ **then**
      - **if** $\nexists u \in e \cap f : b(u) = 1$ **then**
        - *reducable* $\leftarrow False$
        - **break**
  - **if** S **and** *reducable* **then**
    - **for** $u \in e$ **do**
      - $b(u) \leftarrow b(u) - 1$
    - // Include e and remove blockedEdges(e)
    - H, $\alpha_{\mathcal{M}} \leftarrow H - N_b$, $\alpha_{\mathcal{M}} + \omega(e)$
- **return** H, b, $\alpha_{\mathcal{M}}$

Proof. We first show that either the edge e or all edges in the set N are contained in a maximum b-matching $\mathcal{M}$. Assumption 2 guarantees that adding the whole set of edges N does not violate the matching constraints. Assumption 1 permits either e

or any edge from N to be included in a matching. Let $F = N \cap \mathcal{M}$ be a part of an optimal solution, we show that $F = N$ or $e \in \mathcal{M}$. Therefore, we first assume that F is nonempty and a strict subset of N, that is $F \subsetneq N$. Due to Assumption 1 this implies $e \notin \mathcal{M}$. Since F is a strict subset of N and we have $\omega(e) > \omega(N) - \min_{f \in N}\{\omega(f)\} \geq \omega(F)$ (Assumption 3), we can swap F for e in $\mathcal{M}$ and gain a better result. This contradicts $\mathcal{M}$ being optimal and $F \subsetneq N$ being a strict subset. If none of the edges in N are part of $\mathcal{M}$, then the edge e is free and can be included.

The vertices of the newly added edge e' in H' correspond to those of N in H. The vertices of e in H are only contained in e' in H' without further neighbors. Therefore, if e' is not in $\mathcal{M}'$, the edge e must be in an optimal solution for H, and otherwise, N is included in an optimal solution for H.

The formula for the weight is correct by the following case distinction. When e' is not contained in $\mathcal{M}'$, then e is free in the corresponding matching $\mathcal{M}$ and must be included in the optimal matching for H. Otherwise the weight of $\mathcal{M}'$ contains $\omega(e')$ and thus the optimal solution in H has weight $\omega(M') + \omega(e) = \omega(\mathcal{M}' \setminus \{e'\}) + \omega(e') + \omega(e) = \omega(\mathcal{M}' \setminus \{e'\}) + \omega(N) - \omega(e) + \omega(e) = \omega(\mathcal{M}' \setminus \{e'\}) + \omega(N)$. □

Algorithm 22 finds edges with two adjacent edges to fold to reduce the problem size. For complexity reasons, only edges of size two are considered. The complexity of this algorithm is $\mathcal{O}(\min(m, n)\Delta_E)$, because we have to check for $\mathcal{O}(\min(m, n))$ candidates if the two neighbors are independent which requires $\mathcal{O}(\Delta_E)$ checks. Without constraining the edge size, it would be $\mathcal{O}(\min(m, n)\Delta_V\Delta_E)$ since we would collect more neighbors. We collect the neighbors on the two vertices with capacity 1 and check if they are independent. If so, we merge the independent neighboring edges and replace the neighbors and the edge e by this merged e_n with a new weight. Figure 6.3 shows a (sub-)hypergraph where this reduction is applicable.

The following data reduction is inspired by the twin reduction for the maximum weight independent set, see Reduction 4.38. It groups non-adjacent edges with the same independent neighborhood together. Afterwards, we directly check if Reduction 6.2 or Reduction 6.4 is applicable.

Reduction 6.5 (Weighted Twin. Figure 6.4)

Let $e_1, e_2 \in E$ be non-adjacent edges and further $L_i = \mathcal{N}(e_i) \setminus \{e_i\}$ be the set of neighboring edges. These are linked with e_i via a vertex with a capacity of 1. Assume each set L_i is independent, $L_1 = L_2$ and $\omega(e_1) + \omega(e_2) > \omega(L_1) - \min_{f \in L_1} \omega(f)$ holds.

Algorithm 22: Weighted Edge Folding

Data: hypergraph $H = (V, E, \omega)$, capacity function b

Result: reduced hypergraph and offset

Reduction `WeightedEdgeFolding(`H, b`):`

- **for** $v \in V$ with $b(v) = 1$ ***and*** $|E(v)| = 2$ **do**
  - **for** $e \in E(v)$ with $|e| = 2$ **and** not scanned before **do**
    - $c, N \leftarrow True, \emptyset$
    - **for** $w \in e$ **do**
      - **if** $|E(w)| > 2 \vee b(w) > 1$ **then**
        - $c \leftarrow False$
      - **else**
        - $N \leftarrow N \cup E(w) \setminus \{e\}$
    - **if** $c \wedge N$ independent **then**
      - **if** $\omega(N) > \omega(e) \wedge \max_{e' \in N} \omega(e_n) \leq \omega(e)$ **then**
        - $e' \leftarrow \bigcup_{e \in N} e$
        - $\omega(e') \leftarrow \omega(N) - \omega(e)$
        - $\alpha_{\mathcal{M}} \leftarrow \alpha_{\mathcal{M}} + \omega(e)$
        - $E \leftarrow E \setminus (N \cup \{e\}) \cup \{e'\}$
- **return** H, b, 0

- *If $\omega(e_1) + \omega(e_2) > \omega(L_1)$, then include e_1 and e_2.*

Reduced Graph	$H' = H - (\{e_1, e_2\} \cup blockedEdges(e_1) \cup blockedEdges(e_2))$
Offset	$\alpha_{\mathcal{M}}(H) = \alpha_{\mathcal{M}}(H') + \omega(e_1) + \omega(e_2)$
Reconstruction	$\mathcal{M} = \mathcal{M}' \cup \{e_1, e_2\}$

- *Else, fold e_1, e_2 and L_1 into a new edge e'.*

Reduced Graph	$H' = H[E \setminus (\{e_1, e_2\} \cup L_1) \cup \{e'\}]$ *with* $\omega(e') = \omega(L_1) - \omega(e_1) - \omega(e_2)$ *and* $e' = \cup_{f \in L_1} f$.
Offset	$\alpha_{\mathcal{M}}(H) = \alpha_{\mathcal{M}}(H') + \omega(e_1) + \omega(e_2)$
Reconstruction	*If* $e' \in \mathcal{M}'$, *then,* $\mathcal{M} = \mathcal{M}' \cup L_1$, *else* $\mathcal{M} = \mathcal{M}' \cup \{e_1, e_2\}$.

Proof. First, we observe that either e_1 and e_2 are both in an optimal solution $\mathcal{M}$, or some subset of the edges of L_1 is. Which means, we can fold the edges e_1 and e_2 into a new edge e' with weight $\omega(e') = \omega(e_1) + \omega(e_2)$. Since $L_1 = L_2$ and all edges in L_2 are linked via a capacity 1 vertex, we do not need to include the vertices in e_2.

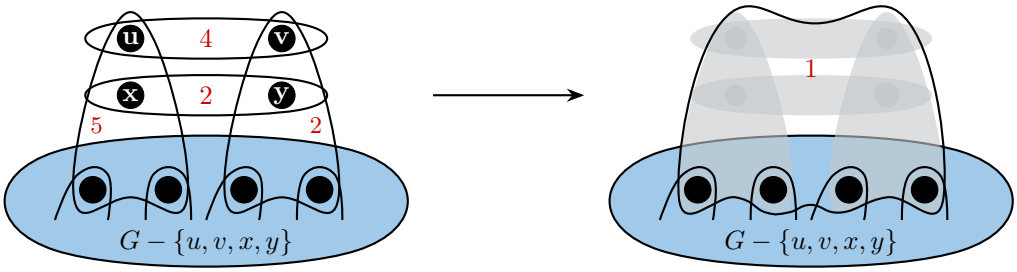


Figure 6.4: Illustration of Weighted Twin, see Reduction 6.5. Edge weights are red, and the capacities of the vertices u, v, x, and y are all equal to one. In this situation, the twin edges are folded with their neighboring edges.

Because any capacity constraint for an edge in L_2 at a vertex in e_2 is also present at a vertex in e_1. If $\omega(e') > \omega(L_1)$, the new edge e' is of higher weight than all of its neighbors combined, satisfying the condition of Reduction 6.2. Therefore, we can include the edge e', yielding e_1 and e_2 being in some optimal solution. Otherwise, $\omega(e') > \omega(L_1) - \min_{n \in L_1} \omega(n)$ still holds. Indeed, the neighbors L_1 and e_1 are linked, L_1 is independent and the weight inequality for Assumption 3 holds. With this, all the properties for Reduction 6.4 are satisfied, which gives the claimed result. □

Algorithm 23 lists a procedure for detecting twins. The algorithm first identifies all possible candidates that have only degree two vertices. Afterward, we identify twins and either apply Reduction 6.2 and add them directly to the matching or apply Reduction 6.4. In this case, they only dominate their neighborhood except for one edge, and we can merge the edges and assign a new weight to the new combined edge. In order to quickly find identical neighborhoods, we have to sort the candidates by neighborhood size. Each independence check requires $\mathcal{O}(\Delta_E \Delta_V)$ comparisons. The overall complexity of this algorithm is $\mathcal{O}(m\Delta_E\Delta_V + m \log m)$. The Reductions 6.4 and 6.5 share common steps and can be combined.

We extend the idea of the domination for maximum weight independent sets presented in Reduction 4.20 to edges in a weighted hypergraph.

Reduction 6.6 (Weighted Domination. Figure 6.5)

Let $e, f \in E$ be two edges with $\omega(e) \geq \omega(f)$ such that e is a subset of f. If, furthermore, there is a vertex $v \in e \subseteq f$ with capacity $b(v) = 1$ then, the edge f can be excluded.

Reduced Graph	$H' = H - f$
Offset	$\alpha_{\mathcal{M}}(H) = \alpha_{\mathcal{M}}(H')$
Reconstruction	$\mathcal{M} = \mathcal{M}'$

Algorithm 23: Weighted Twin

Data: hypergraph $H = (V, E, \omega)$, capacity function b
Result: reduced hypergraph and offset
Reduction `TwinReduction(`H, b`)`**:**

```
for v ∈ V with b(v) = 1 and |E(v)| = 2 do
    X ← ∅
    for e ∈ E(v) and not scanned before do
        candidate, neighbors ← True, ∅
        for u ∈ e do
            if |u| > 2 ∨ b(u) > 1 then
                candidate ← False
            neighbors ← neighbors ∪ E(u) \ {e}
        if candidate then
            X ← X ∪ {(e, neighbors)}
for ∃N, e1 ≠ e2 : (e1, N), (e2, N) ∈ X do
    if N independent then
        e' ← ⋃_{e∈N} e
        ω(e') ← ω(N) − ω({e1, e2})
        E ← (E \ {e1, e2}) ∪ {e'}
        if ω'(e') ≥ ω(N) then
            H, α_M ← NeighborhoodRemoval(H, b)      // H = (V, E, ω)
        else if ω(e') > ω(N) − min_{e∈N} ω(e) then
            H, α_M ← EdgeFolding(H, b)
return H, b, 0
```

Proof. Let e and f be two edges with $\omega(e) \geq \omega(f)$ and $e \subseteq f$. Furthermore, let $v \in e \subseteq f$ be a vertex with capacity $b(v) = 1$. Let $\mathcal{M}$ be an optimal solution containing f. First note that since $b(v) = 1$ and $v \in e \cap f$ the edges $e \notin \mathcal{M}$. However, since e is a subset of f and it holds $\omega(e) \geq \omega(f)$, we can construct a new optimal solution by replacing f with e, i. e., $\tilde{\mathcal{M}} \coloneqq \mathcal{M} \setminus \{f\} \cup \{e\}$. Since $\omega(\mathcal{M}) \leq \omega(\tilde{\mathcal{M}})$, there is an optimal solution that does not contain f, and it can be removed safely. □

In Algorithm 24, we show pseudocode for finding a weighted domination. We iterate over each vertex and its incident edges in descending weight. We check if the following edges satisfy the natural size constraint and check for the subset criterion

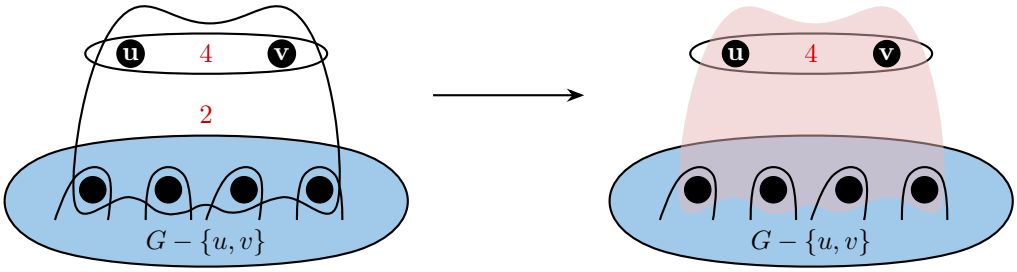


Figure 6.5: Illustration of Weighted Domination; see Reduction 6.6. Edge weights are red, and the capacities of u and v are equal to one. The edge containing only u and v is dominated.

Algorithm 24: Weighted Domination

Data: hypergraph $H = (V, E, \omega)$, capacity function b
Result: reduced hypergraph and offset
Reduction `WeightedDomination(`H, b`):`

```
for v ∈ V with b(v) = 1 do
    for e ∈ E(v) ordered by weight desc. and not scanned before do
        D ← ∅
        for f ∈ E(v) with ω(e) ≥ ω(f) do
            if |f| ≤ |e| ∧ hash(f, e) then
                D ← D ∪ {f}
                break
        for u ∈ e do
            D ← {c ∈ D | u ∈ c}
        E ← E \ D
return H, b, 0
```

using a simple hash function. After collecting all candidates, we check which of these candidates are super sets and remove them from the graph. The complexity with the shown subset check is $\mathcal{O}(m(\Delta_E \Delta_V \log \Delta_V))$ if we do not have a hashing function. For each edge, we can collect at most $\Delta_V - 1$ candidates. For each candidate, we have to check up to Δ_E vertices of e that they are indeed incident requiring $\log \Delta_V$ comparisons if the list is sorted. By multiplying the id of vertices contained in an edge, storing these results in a wide integer, and using the modulo operator to check for division without remainder, we can reduce the time complexity to $\mathcal{O}(m(\Delta_E + \Delta_V))$ in the ideal case. This requires a growable wide integer resulting in larger memory cost, so using a hash function, like multiplying only the k least significant bits of the

Algorithm 25: b-Matching Algorithm

Data: hypergraph $H = (V, E, \omega)$
Result: hypergraph-b-matching $\mathcal{M}$
Procedure BMatching(H):
  $\mathcal{K}, \alpha_{\mathcal{M}} \leftarrow$ ExactReduce(H, R) // R ordered list of reductions
  $\mathcal{M}' \leftarrow$ Greedy($\mathcal{K}$) or ILP($\mathcal{K}$)
  $\mathcal{M} \leftarrow$ restore(H, $\alpha_{\mathcal{M}}$, $\mathcal{M}'$)
  return $\mathcal{M}$

ids, seems reasonable. If B is the width of the wide integer and we want to use the k least significant bits, we can hash all edges if $\Delta_E < (\frac{B}{k}\log(2) - 1)$ holds.

6.2 Hypergraph b-Matching Algorithms

We now give an overview of our algorithms to solve the general weighted hypergraph b-matching problem.

Our approach shown in Algorithm 25 starts by using exact data reductions devised in Section 6.1 to reduce the instance size. The reduced instances can then be used as input to our heuristic or the exact solver (based on the ILP). Our heuristic, introduced in Section 6.2.1, computes a good initial b-matching using a greedy strategy. In Section 6.2.2, we present a local search that improves the solution quality further. Afterward, in Section 6.2.3, we give the exact integer linear program for this problem. Once a solution is computed on the reduced instance, we reconstruct it into a solution for the original instance.

6.2.1 Greedy Approaches

We now present our greedy approaches to compute initial solutions. Roughly speaking, we use greedy algorithms that sort the edges by a priority function and add free edges in this order. Note that the most intuitive order of adding heavy edges first may yield poor results. This is because any edge may block a wide range of other edges from being added, for example, if the current edge has many vertices. Thus, other priority functions are necessary. The core idea of our algorithm is to assign each edge a positive priority value and then add edges greedily in descending order of their priority (edges with the highest priority are added first). To overcome the problems of the weight priority function explained above, we scale the weight function with the number of their incident vertices and with the capacity at each vertex of the edge. This ensures

	Scale by	**Definition**
CAP	Capacity	$h_{cap}(e) \coloneqq \omega(e) \prod_{v \in e} b(v)$
PIN	Inverse edge size	$h_{pin}(e) \coloneqq \omega(e)\lvert e\rvert^{-1}$
PIN,CAP	Capacity and inverse edge size	$h_{pin,cap}(e) \coloneqq \omega(e)\lvert e\rvert^{-1} \prod_{v \in e} b(v)$
SCALED	Capacity and inverse vertex degree	$h_{scaled}(e) \coloneqq \omega(e) \prod_{v \in e} b(v)\lvert E(v)\rvert^{-1}$

Table 6.1: Overview of different objective functions used in the general framework.

that we select edges first that a) have a high weight, b) do not block a lot of other edges, and c) have vertices with a lot of remaining capacity. Table 6.1 defines the different objective functions that result in the four algorithms CAP, PIN, PIN, CAP and SCALED, defined by the general framework and the respective objective function.

6.2.2 Local Search

We now give a brief overview of our local search algorithm based on swapping. A pseudocode for the approach is given in Algorithm 26. A *swap* removes one edge e such that two edges e_1, e_2 become unblocked. A swap is feasible if the combined weight of e_1 and e_2 is greater than that of the removed edge e.

Swapping. In our swapping method, we are searching for a matched edge and two non-matched adjacent edges that are only blocked by the matched edge and can be admitted to the matching simultaneously without conflict.

$(1, 2)$**-Swaps.** For each edge in the matching e, we first collect all neighboring edges that satisfy the condition of being only blocked by e. In the second step, we identify a pair of edges that can be added without conflict when e is removed from the matching, resulting in an improvement. If we find such an edge pair, we include them in the matching and remove the edge e from it. After a successful swap, we add all free edges to the solution.

Perturbation. The swapping algorithm ends up in a local optimum if executed repeatedly. Therefore, we perturb the solution (forcing hyperedges into the solution) similarly to Andrade et al. [7], who do this for the independent set problem, see Section 3.2.3. The geometric distribution and the number of unmatched edges forced in the solution are directly adopted from Andrade et al. [7].

Algorithm 26: Local Search

Data: hypergraph $H = (V, E, \omega)$, solution $\mathcal{M}$ to be improved

Result: improved solution

Procedure OneTwoSwap(H, $\mathcal{M}$)**:**

- **for** $c \in \mathcal{M}$ **do**
  - $l \leftarrow \emptyset$
  - **for** $p \in c$ **do**
    - **for** $e \in E(p) \setminus \mathcal{M}$ **do**
      - **if** $blocked(e, \mathcal{M}) \subseteq blocked(c, \mathcal{M})$ **then**
        - $l \leftarrow l \cup \{e\}$
  - **if** $|l| > 1$ **then**
    - $\Phi(x) := blocked(x, \mathcal{M} \setminus \{c\} \cup \{x\})$
  - **if** $\exists x, y \in l : \Phi(y) \cap \Phi(x) = \emptyset$ and $\omega(x) + \omega(y) > \omega(c)$ **then**
    - $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\} \cup \{x, y\}$
    - $\mathcal{M} \leftarrow$ maximize($\mathcal{M}$)
- **return** $\mathcal{M}$

Procedure ExhaustiveOneTwoSwap(H, $\mathcal{M}$)**:**

- **while** improvement **do**
  - $\mathcal{M} \leftarrow$OneTwoSwap(H, $\mathcal{M}$)
- **return** $\mathcal{M}$

Procedure ILS(H, $\mathcal{M}$)**:**

- $\mathcal{M}_{\text{best}} \leftarrow \mathcal{M}$
- **while not** stopping criterion holds **do**
  - $\mathcal{M}_* \leftarrow$ Perturb(H, $\mathcal{M}$)
  - $\mathcal{M}_* \leftarrow$ExhaustiveOneTwoSwap(H, $\mathcal{M}_*$)
  - $\mathcal{P} \leftarrow ((\omega(\mathcal{M}_{\text{best}}) - \omega(\mathcal{M}_*))(\omega(\mathcal{M}) - \omega(\mathcal{M}_*)))^{-1}$
  - **if** $\omega(\mathcal{M}_*) > \omega(\mathcal{M})$ **then**
    - $\mathcal{M} \leftarrow \mathcal{M}_*$
  - **else if** $x \in \mathrm{U}(0, 1) < \mathcal{P}$ **then**
    - $\mathcal{M} \leftarrow \mathcal{M}_*$
  - **if** $\omega(\mathcal{M}) > \omega(\mathcal{M}_{best})$ **then**
    - $\mathcal{M}_{\text{best}} \leftarrow \mathcal{M}$
- **return** $\mathcal{M}_{best}$

Iterated Local Search. The iterated local search drives the whole process. While the stopping criterion, a fixed number of unsuccessful searches, is not met, the current solution $\mathcal{M}$ gets perturbed and then exhaustively improved by $(1,2)$-swaps. We call the obtained solution $\mathcal{M}_*$. A better solution automatically becomes the new starting point for the next iteration. Similarly to Andrade et al. [7], we allow a slightly worse solution to be accepted as the starting point of our next local search. However, accepting worse solutions is rather unlikely and only done with a low probability determined by $((\omega(\mathcal{M}_{\text{best}}) - \omega(\mathcal{M}_*))(\omega(\mathcal{M}) - \omega(\mathcal{M}_*)))^{-1}$.

6.2.3 Integer Linear Programming Formulation

To solve the b-matching problem to optimality, either on the original or exactly reduced hypergraph, we use the following integer linear program:

$$\begin{aligned} &\max \sum_{e \in E} x_e \omega(e) \\ &s.t. \quad \sum_{e \in E(v)} x_e \leq b(v) && \forall v \in V \\ &\qquad\quad x_e \in \{0,1\} && \forall e \in E \end{aligned}$$

For every edge $e \in E$, the integer linear program has a variable x_e, which is set to $x_e = 1$ if and only if the edge e is part of the matching and set to zero otherwise. The maximization term is the sum of the weights of the selected edges. The main constraint restricts the number of selected edges to obey the capacity at each vertex in the original hypergraph.

6.3 Experimental Evaluation

We conclude this chapter with the experimental evaluation of our reductions and proposed algorithm. We first analyze the priority functions and compare these with bSuitor by Khan et al. [136]. We then evaluate the effectiveness of our exact data reductions, and finally, we benchmark our local search and compare against the method by Dufosse et al. [65].

Methodology. We implemented our algorithms and data structures in C++17. We compiled our program and all compared implementations using g++-12.1 with full optimization turned on (-O3 flag). In our experiments, we used Machine 4. We run the experiments 10 times and take the arithmetic mean per instance. For solving the ILP, we use SCIP [26], one of the fastest open-source ILP solvers. Deterministic

experiments were only executed once if the results (size, weight) and not the time was measured. The experiments were scheduled in parallel up to the number of physical cores of the machine, and the number of cores used by SCIP concurrently was limited to one. Each experiment run has a memory budget of 60 GB per instance and 140 GB in total.

Data Sets. We use Instance Set 8, which contains a wide range of hypergraph instances collected from various sources to evaluate our algorithms and to compare against state-of-the-art approaches.

Instance Set 8 (Hypergraphs)**.** *This set contains 488 hypergraph instances, provided by Gottesbüren et al. [96]. These instances come from four different use cases of hypergraphs, including 18 from circuit design (`ISPD98`) [6], 10 routability-driven placement (`DAC2012`) [209], 184 instances derived from general matrices from the Suite Sparse Collection (`SPM`) [53] and 276 instances derived from SAT-solving problems (`SAT14`) [22].*

When comparing against state-of-the-art b-matching algorithms on graphs, we use the 10 graphs from the Florida Sparse Matrix Collection by Davis and Hu [53], as also done by Khan et al. [136] who also present the algorithm that we compare against. We perform experiments with uniform capacities of 1, 3, 5, and random capacity. If we use random capacity, we sample the capacity uniformly at random for each vertex between one and its vertex degree. For general experiments, we assigned weights uniformly at random to the edges between 1 and 100.

This section is organized as follows. First, we compare our greedy priority functions introduced in Section 6.2.1 on hypergraphs. Furthermore, we compare our results with those of the b-matching graph algorithm bSuitor by Khan et al. [136]. Then, we study the effectiveness of our exact data reductions using SCIP [26], which is a state-of-the-art open-source ILP solver. Finally, we benchmark our local search and compare against the method by Dufosse et al. [65] on 6-partite, 6-uniform hypergraphs.

Priority Functions and Initial Solutions

The priority functions for this experiment are defined in Section 6.2.1. We evaluate them on the general hypergraph data set. Figure 6.6 shows the results for instances with random capacity comparing the priority functions. For reference, we include the simple greedy WEIGHT function without any scaling as a baseline. For the uniform capacity of 1, the PIN,CAP and PIN compute the same best result since their objective function for this capacity is the same. Our other two proposals are nearly identical to the simple greedy WEIGHT approach. For a higher uniform capacity of 3, 5, and

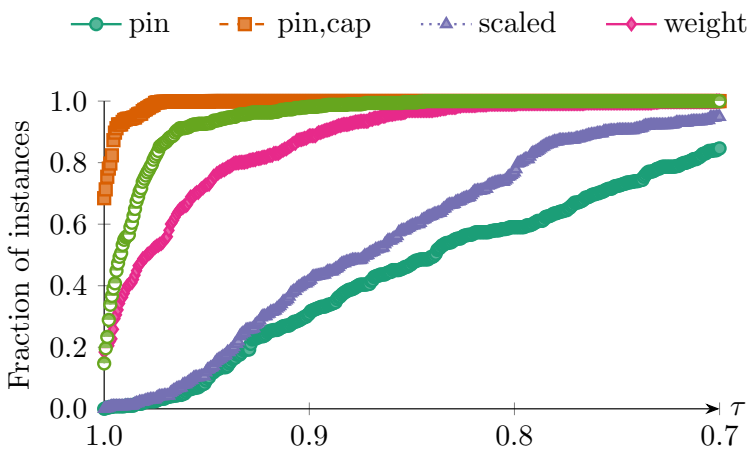


Figure 6.6: Performance profile for solution quality comparing the priority functions on the general hypergraph data set with $b(v) = \text{rnd}$ and random edge weights.

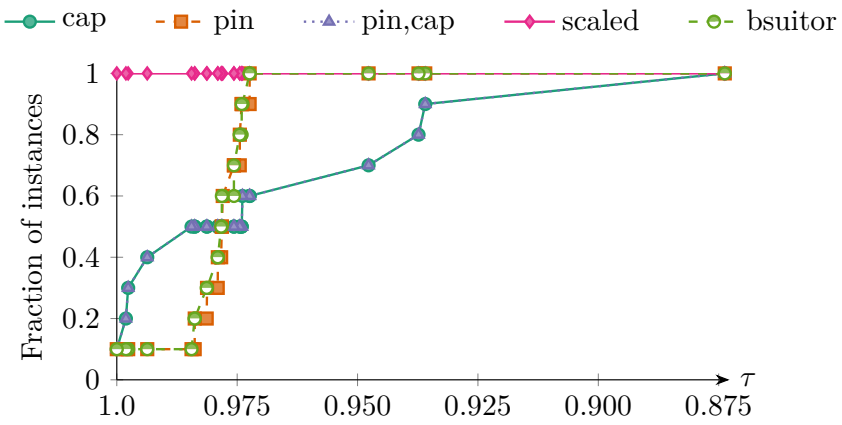


Figure 6.7: Performance profile comparing solution quality for different priority functions with BSUITOR. Here, we are using $b(v) = \text{rnd}$ and random edge weights on graph instances selected by Khan et al. [136].

random capacities, the quality of all of our proposals except for the PIN is worse than the WEIGHT function. The PIN function finds the best solution in around 70 to 80 % of the cases and requires a maximum $\tau = 0.9$ across all capacities. Since we only need to precompute the values of PIN once, the running time of PIN only slightly exceeds the running time of the simple WEIGHT algorithm, on average by 0.4 % of the WEIGHT function, which we consider neglectable. We *conclude* that PIN is a natural good choice to compute initial solutions on general hypergraphs. The other proposals are not viable since they yield worse results than the greedy WEIGHT algorithm.

Reduction	Constraint	Default
6.2	edge size	10
6.3	edge size	8
6.3	clique size	80
6.6	subedge size	6
6.6	candidate	6
6.5	edge size	4
All	iterations	10

Table 6.2: Parameter constraints of the reductions.

Special Case: Graph b-Matching

In order to show the versatility of our approaches, we also tested them on normal graphs and compared our results to those by Khan et al. [136]. The results of this first experiment are shown in Figure 6.7. The method BSUITOR by Khan et al. [136] is run in comparison to our approaches introduced in Section 6.2.1. We have a fixed edge size of two (graphs) for this experiment, and the capacity is static in all but one experiment. This results in, all but one function from Section 6.2.1 reporting the same results. Only the SCALED approach differs by considering the number of incident edges, which yields a better solution than the other approaches. We can report an improvement of up to 2 % over BSUITOR on random capacity and up to 7 % on capacity 1. The running time of BSUITOR on a single core in sequential can not be matched by our algorithms on graphs. On these instances, our approaches are, on average, 3.19 times slower. This is, however, expected since our data structure supports hypergraphs, which introduces an additional overhead when only considering graphs.

Reductions and Speedup

In this experiment, we investigate how well our reductions can speed up solving b-matching problems with SCIP [26]. As some data reductions can be expensive, we restrict different parameters in the search. The parameters we use are given in Table 6.2. We empirically chose the parameters to balance between the reduction time and success rate. The most sensitive tuning parameters are the ones used in the Algorithm 24, since allowing bigger edges causes more checks to be needed. We apply our search algorithms for the reductions up to ten times and pass the resulting core problem to SCIP. We then compare its total running time to the time it takes to solve the whole (hyper-)graphs without applying reductions. When applying reductions, the time to find and apply them is included in the overall running time. We set a time limit for the SCIP computations of 1 800 seconds and test it on uniform capacities of 1, 3, and 5, as well as on random capacity. We ran this experiment once to determine which

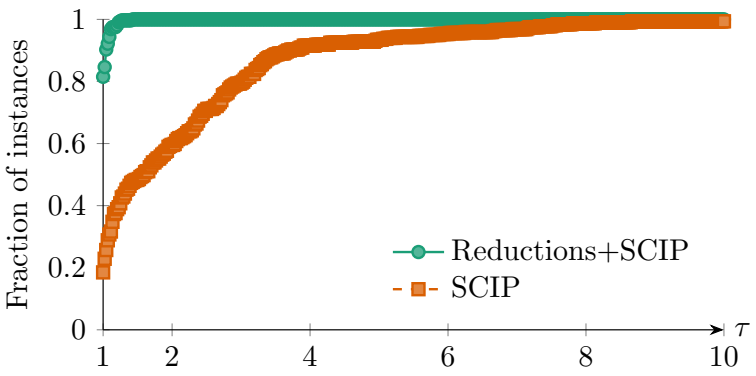


Figure 6.8: Performance profile for comparing the impact of using reductions with the optimum ILP solver SCIP. We run the solver with and without reductions on a subset of 395 hypergraphs, which are solvable within the given time limit using $b(v) = \text{rnd}$ and random edge weights.

instances are solvable in the given time with any of these capacities and repeated the experiment on the solvable subset of 395 hypergraphs ten times.

Only instances on which SCIP returns the optimum within the time limit are considered in the time performance profile for random capacity shown in Figure 6.8. Similar results were obtained for the static capacities. The performance profiles show that some instances are strongly affected by the reductions. Some instances are so small that the time to search for reductions is longer than when solving directly. Overall, 80 % of instances benefit from running the reductions search, and 40 % have a speedup of at least factor 2.

In Table 6.3, we report the average and geometric speedup sorted by hypergraph class. Our reductions achieve the best average speed up on the `DAC2012` instances of nearly 7 for random capacity. Our reductions halve the average edge count of these instances. The least speed up is observed on the `ISPD98` instances for uniform capacities.

A plot of the removal effectiveness is shown in Figure 6.9. On the y-axis, we display the relative edge count of the hypergraphs after applying our reductions. Similarly, on the x-axis, we plot the relative vertex count. Different shapes signify different capacities, and colors are used for the different classes of hypergraphs. Most instances are located above the diagonal, meaning the vertices are more reduced than the edges. There are two major clusters, one for not reducible instances and one for nearly completely reduced instances. Figure 6.10 shows the relative running time of the reductions and effect on all 488 instances. The relative effect is computed on how many edges are included, excluded, or deferred by the respective reductions. For all capacities, the Abundant Vertices, Neighborhood Removal, and Weighted Domination are the most significant reductions and take up the most time. The impact of the other

Instances			Reduction		Speedup	
Set	$b(v)$	#	$E\,\%$	$V\,\%$	Avg.	Geom.
ISPD98	1	15	96	84	1.48	1.43
	3	12	64	42	1.10	1.02
	5	16	33	20	1.71	1.65
	rnd	18	66	33	2.50	2.48
SPM	1	76	93	46	2.60	1.74
	3	85	84	45	1.64	1.21
	5	95	51	18	2.86	1.57
	rnd	149	86	58	1.52	1.29
DAC2012	3	9	39	30	3.14	2.96
	5	10	6	3	7.12	6.66
	rnd	10	50	15	6.98	6.85
SAT14	1	88	99	92	2.61	1.50
	3	154	55	8	3.71	2.18
	5	164	44	3	5.77	3.00
	rnd	155	75	25	2.38	2.05
overall	1	179	97	78	2.51	1.59
	3	260	57	11	2.89	1.75
	5	285	41	4	4.62	2.40
	rnd	332	74	31	2.14	1.74

Table 6.3: Reduction effectiveness and speed up per class of hypergraphs. We report the average percentage reduction of edges (E %) and vertices (V %), as well as the speedup for each class. Only exactly solvable instances are considered.

reductions (Weighted Edge Folding, Weighted Isolated Edge Removal, and Weighted Twin) is limited, but also the share of running time is small. The Neighborhood Removal reduction has the biggest impact for randomly assigned capacities, halving over 50 % of the affected edges while taking only a small fraction of running time. The Weighted Domination contributes the most affected edges for a uniform capacity of 1 but also takes the most of the running time. The Weighted Isolated Edge Removal finds many candidates at a capacity of 1 and uses around one-quarter of the running time, finding only a few affected edges. For higher or random capacities, the Abundant Vertices reduction requires most of the running time but also contributes more than half of the affected edges in the static capacity case.

Local Search

The iterated local search (ILS) is the main focus of the following experiment. The ILS has only one threshold parameter k, determining after how many fruitless perturbations and local searches the search is terminated. Then, the best result found so far is returned. In our experiment, we start with the best solution obtained using our priority function PIN and improve it using ILS. We chose $k = 15$, as it showed

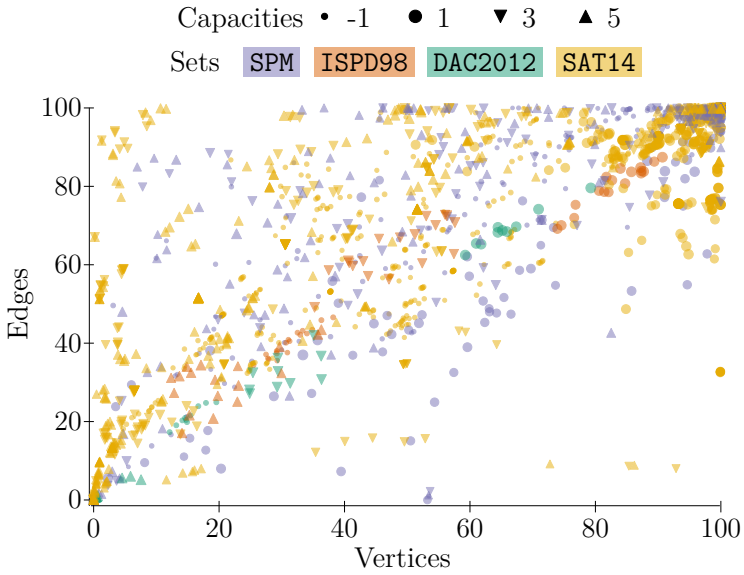


Figure 6.9: Percentage of size of reduced graphs for instances with random edge weights. Some instances are completely reduced, while others are not reduced at all.

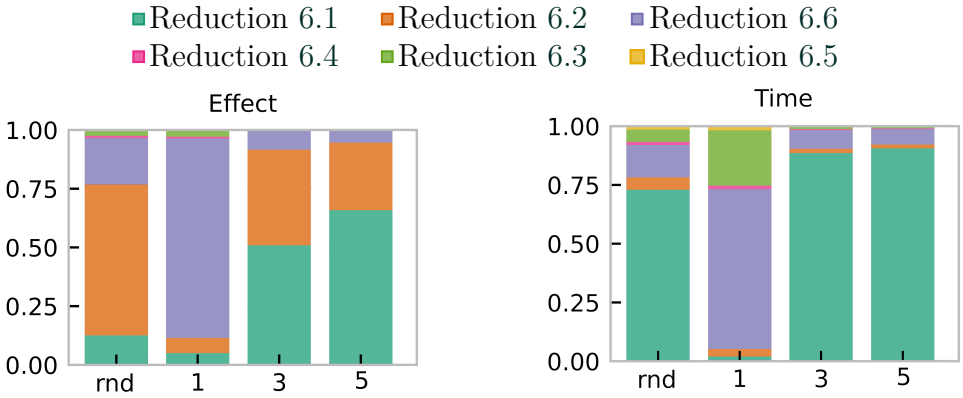


Figure 6.10: Relative running time and contribution to the reduced size of each reduction rule evaluated on instances with random and equal capacities of 1, 3, and 5.

a good balance between running time and improvement on this collection of hypergraphs. For reference, we include $k = 50$ in the performance profile, which has an average running time tenfold in comparison to $k = 15$. We report an improvement of up to 8 % at max overall capacities tested. Running time is, on average, ten times longer than computing the initial solution. However, this is unsurprising as the initial method only sorts by the priority function and adds edges greedily. The performance plot for this experiment is shown in Figure 6.11. Some instances are only improved by a negligible amount, while the best instances are improved by 8 %. The average improvement is around 3 %. Increasing $k = 50$ can improve the solution quality, but it comes with up to 100 times slower running times.

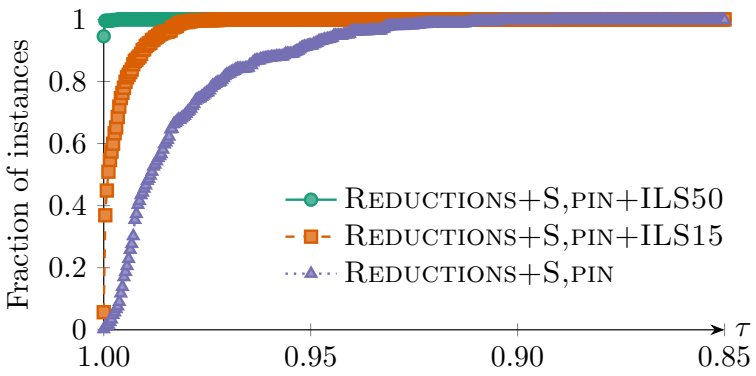


Figure 6.11: Performance profile comparing solution quality improving REDUCTIONS+S,PIN with different ILS configurations setting $k = 15$ and $k = 50$.

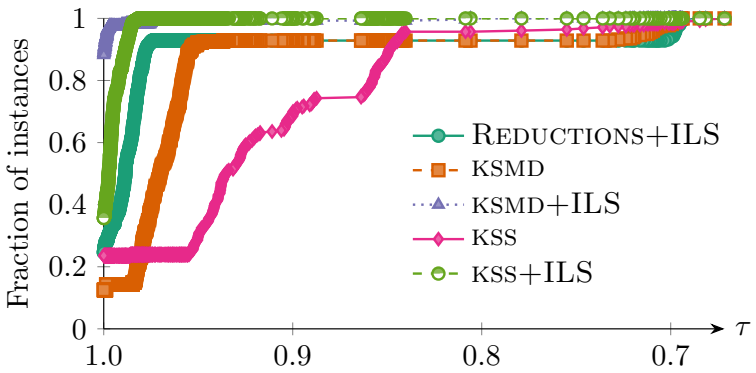


Figure 6.12: Performance profile for solution quality comparison of REDUCTIONS+ILS with KSMD, KSS by Dufosse et al. [65]. Additionally, we give the solution quality when improving the competitor solutions with ILS. Here, we present results on instances with uniform edge weights and capacities set to 1. The local search threshold is set to $k = 1\,500$.

1-Matching on d-partite, d-uniform Hypergraphs

This experiment compares and combines our algorithm with those developed by Dufosse et al. [65] on a special class of hypergraphs with uniform edge weight and structure. They implemented KSS and KSMD, which employ two configurations of the Karp-Sipser approach. We are using KSS with 20 scaling repetitions as proposed by the authors. We built and linked both algorithms into our benchmark program to ensure equal compile flags and settings.

Because of the uniform structure of these instances, we need more time in the perturbation phase to find an improvement. We can only report minimal improvements for a low threshold of $k = 50$ while running time stays close to the base approaches. This is why we set $k = 1\,500$, which is significantly higher than for the non-uniform

instances. Figure 6.12 shows compares the solution quality of this ILS configuration, when combining ILS with KSS and KSMD or our approaches as initial solution.

Our REDUCTIONS+ILS, which combines the reductions, a greedy initial matching by PIN, and iterated local search with $k = 1\,500$ approach, has a better solution quality than the KSS and KSMD approaches. Therefore, we want to combine both approaches.

Our data reductions can not be combined with KSS and KSMD since these algorithms require uniform edge size, and our data reductions do not ensure this. However, we considerably improve the quality of KSS and KSMD by using our ILS as a postprocessing step. In the case of KSMD, we report a quality improvement of more than 30 % on around 8 % of the instances, even surpassing the results of the REDUCTIONS+ILS approach. Setting $k = 1\,500$ unsuccessful tries can result in a hundredfold running time on a few instances while consistently yielding better quality. The improvement for KSS is more pronounced than for KSMD, which has a specific reduction rule for this kind of hypergraph. Improved quality comes at the expense of running time. The KSMD is the fastest approach and ten times faster than the KSS approach. On average, we need tenfold the time to compute the solution than KSS. Both KSMD and KSS are deterministic and thus can not be repeated multiple times for a more fair comparison. In conclusion, the choice of k is sensitive for determining quality and running time and is dependent on the structure of the problem. Computing solutions on uniform weight instances require more tries than their non-uniform counterparts.

6.4 Conclusion

Using novel data reductions, we developed a scalable algorithm for the general MAXIMUM WEIGHT b-MATCHING problem in hypergraphs. Our reductions can identify and incorporate optimum edges into a preliminary solution, reduce the instance size by combining the decision for multiple edges, and eliminate non-optimal edges from the input. We engineer new greedy heuristics and a local search approach to improve solutions. Experiments show that our data reductions scale well to large instances and can accelerate state-of-the-art black-box solvers. The new greedy solutions are up to 10 % better than the state-of-the-art on general hypergraphs. On graphs, these approaches also yield good results. The local search is able to improve solutions up to 30 % over recent results by Dufosse et al. [65] on some instances. Given the good results, we will release our software as an open-source project. Future work includes parallelization of our algorithms, research in nonlinear optimization objectives, improving local search techniques, and applying our methods to related **NP**-hard problems.

Chapter 7

Discussion

In this chapter, we summarize the contributions of this dissertation and discuss directions of future work. We give a more detailed conclusions for the Maximum Weight Independent Set problem in Section 4.6, the Maximum (Weight) 2-Packing Set problem in Section 5.5, and the Hypergraph b-Matching problem in Section 6.4.

7.1 Conclusion

This dissertation contributes and evaluates various data reduction rules for independence problems. Additionally, we present novel algorithms using these reductions, including exact and heuristic approaches. The methods are evaluated on large, real-world data sets. This evaluation shows the importance of data reduction rules for these problems and that our approaches outperform state-of-the-art solvers.

For the Maximum Weight Independent Set problem, we contribute several new reduction rules combined with a comprehensive overview of the existing ones. We analyze the impact of the reductions and their order. Addintionally, we show, what rules are used in different solvers. We present LearnAndReduce, a preprocessing algorithm for the MWIS problem that combines Graph Neural Networks (GNNs) with a large collection of reduction rules to reduce further and faster than previously possible. Then, we contribute two heuristics for the MWIS problem. The first heuristic, m^{2}wis, is an evolutionary approach that uses exact and heuristic reduction rules. The second heuristic, CHILS, combines a local search approach with our new Difference-Core strategy. Both heuristics are evaluated on large, real-world data sets and compute high-quality solutions, outperforming the state-of-the-art. Moreover, we present the new optimal neighborhood exploration technique for the

dynamic setting, which can compute high-quality solutions to the dynamic M(W)IS problem with a powerful update operation.

For the Maximum (Weight) 2-Packing Set problem, we present 23 new reduction rules. Using these rules, we present several new methods to compute high-quality solutions faster than state-of-the-art techniques by multiple orders of magnitude. The Maximum (Weight) 2-Packing Set problem can be reduced to the Maximum (Weight) Independent Set problem using the square graph. Utilizing our new data reduction rules combined with this reduction, we present a preprocessing technique that can speed up solving the problem for state-of-the-art independent set solvers. In our experiments, we see speed-ups up to multiple orders of magnitude faster. Furthermore, we present a new heuristic that uses the Concurrent Difference-Core Heuristic to compute high-quality solutions to the Maximum (Weight) 2-Packing Set problem, outperforming the independent set approaches on the transformed graph, especially on large instances.

We propose novel data reduction rules for the b-Matching problem in hypergraphs and a scalable algorithm utilizing these data reductions. We also engineer new greedy approaches and a local search framework to improve solutions further. Our experiments show that the new data reductions scale well to large instances and accelerate state-of-the-art black-box solvers.

Overall, we evaluate the effectiveness of our new data reductions and show that our algorithms compute high-quality solutions to independence problems in graphs, dynamic graphs, and hypergraphs. Our approaches improve solution quality, running time, and memory consumption compared to state-of-the-art methods. These results make the proposed techniques very promising for practical applications.

7.2 Future Work

There are several directions for future research, starting from the contributions of this dissertation. Here, we want to discuss high-level ideas to show how the presented work can be applied to other problems. The conclusion of each chapter presents a detailed discussion about directions for future work, starting from specific contributions to improve the current methods presented.

The technique of data reduction rules is very general and can be applied to many combinatorial optimization problems. The presented work introduces a comprehensive overview of several reduction rules for the independent set problem and presents examples of how to translate these rules to other independence problems. In future

work, we want to use the ideas behind these rules to find new reduction rules for other combinatorial optimization problems.

Furthermore, it is interesting to investigate the LearnAndReduce technique for other combinatorial optimization problems. Moreover, it could be interesting to use GNNs to reduce the graph directly, i. e., without checking if a reduction rule applies. Even though this is no longer exact, it could be a powerful heuristic for many combinatorial optimization problems.

Additionally, we are interested in investigating our new D-Core approach for other combinatorial optimization problems. Since the main ideas are very flexible, they can be used for a variety of problems. As a metaheuristic, the introduced Concurrent Difference Core Heuristic only requires that solutions can be compared to find the Difference-Core; otherwise, any heuristic method can be used internally. Examples of problems to try include Vertex Cover, Dominating set, Graph Coloring, and Connectivity Augmentation. As an added benefit, the Concurrent Difference-Core Heuristic is trivially parallelizable, which could enable improvements in the parallel setting too.

Bibliography

[1] OpenStreetMap. https://www.openstreetmap.org.

[2] Faisal N. Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent Advances in Practical Data Reduction. In *Algorithms for Big Data*, pages 97–133. Springer, 2022.

[3] Alexander A. Ageev. On finding critical independent and vertex sets. *SIAM Journal on Discrete Mathematics*, 7(2):293–295, 1994.

[4] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609(1):211–225, 2016.

[5] Gabriela Alexe, Peter L. Hammer, Vadim V. Lozin, and Dominique de Werra. Struction revisited. *Discrete applied mathematics*, 132(1-3):27–46, 2003.

[6] Charles J. Alpert. The ISPD98 circuit benchmark suite. In *Proceedings of the International Symposium on Physical Design*, pages 80–85, New York, NY, USA, 1998. Association for Computing Machinery.

[7] Diogo Vieira Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012.

[8] Georg Anegg, Haris Angelidakis, and Rico Zenklusen. Simpler and stronger approaches for non-uniform hypergraph matching and the füredi, kahn, and seymour conjecture∗. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 196–203. SIAM, 2021.

[9] Eugenio Angriman, Henning Meyerhenke, Christian Schulz, and Bora Uçar. Fully-dynamic weighted matching approximation in practice. In *Proceedings of the SIAM Conference on Applied and Computational Discrete Algorithms (ACDA)*, pages 32–44. SIAM, 2021.

[10] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Symposium on Theory of Computing (STOC)*, pages 815–826. ACM, 2018.

[11] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *Proceedings of the 13th Symposium on Discrete Algorithms (SODA)*, pages 1919–1936. SIAM, 2019.

[12] Yoshito Atsuta and Satoshi Takahashi. The maximum weighted k distance-d independent set problem on interval graph. In *Proceedings of the 9th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 840–841, 2020.

[13] Davide Bacciu, Alessio Conte, and Francesco Landolfi. Generalizing downsampling from regular data to graphs. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, pages 6718–6727, 2023.

[14] Gábor Bacsó, Dániel Marx, and Zsolt Tuza. H-Free Graphs, Independent Sets, and Subexponential-Time Algorithms. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 1–12, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[15] D. A. Bader, H. Meyerhenke, P. Sanders, C. Schulz, A. Kappes, and D. Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73–82. Springer, 2014.

[16] David A. Bader, Andrea Kappes, Henning Meyerhenke, Peter Sanders, Christian Schulz, and Dorothea Wagner. Benchmarking for Graph Clustering and Partitioning. In *Encyclopedia of Social Network Analysis and Mining, 2nd Edition*. Springer, 2018.

[17] Brenda S Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

[18] Michel L. Balinski. On a selection problem. *Management Science*, 17(3):230–231, 1970.

[19] Max Bannach and Sebastian Berndt. PACE solver description: The PACE 2023 Parameterized Algorithms and Computational Experiments Challenge:

Twinwidth. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[20] Lukas Barth, Benjamin Niedermann, Martin Nöllenburg, and Darren Strash. Temporal map labeling: A new unified framework with experiments. In *Proceedings of the 24th International Conference on Advances in Geographic Information Systems*, pages 1–10, 2016.

[21] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science*, pages 382–405. IEEE Computer Society, 2019.

[22] Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo. The SAT competition 2014., 2014.

[23] Una Benlic and Jin-Kao Hao. Breakout local search for maximum clique problems. *Computers & Operations Research*, 40(1):192–206, 2013.

[24] Elisabetta Bergamini and Henning Meyerhenke. Fully-dynamic approximation of betweenness centrality. In *Proceedings of the 23rd European Symposium on Algorithms (ESA)*, volume 9294, pages 155–166. Springer, 2015.

[25] Piotr Berman. A d/2 approximation for maximum weight independent set in d-Claw free graphs. In *Algorithm Theory - SWAT 2000*, pages 214–219, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[26] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP optimization suite 8.0. Technical Report, Optimization Online, 2021.

[27] Sujoy Bhore, Guangping Li, and Martin Nöllenburg. An algorithmic study of fully dynamic independent sets for map labeling. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA)*, volume 173, pages 1–24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[28] Marcel Birn, Vitaly Osipov, Peter Sanders, Christian Schulz, and Nodari Sitchinava. Efficient parallel and external matching. In *Proceedings of Euro-Par Parallel Processing*, volume 8097, pages 659–670. Springer, 2013.

[29] Jannick Borowitz, Ernestine Großmann, and Christian Schulz. Engineering fully dynamic Δ-orientation algorithms. In *Proceedings of the SIAM Conference on Applied and Computational Discrete Algorithms (ACDA)*, pages 25–37. SIAM, 2023.

[30] Jannick Borowitz, Ernestine Großmann, and Christian Schulz. Optimal Neighborhood Exploration for Dynamic Independent Sets. *arXiv: 2407.06912*, 2024.

[31] Jannick Borowitz, Ernestine Großmann, and Christian Schulz. Finding maximum weight 2-packing sets on arbitrary graphs. *arXiv:2502.12856*, 2025.

[32] Jannick Borowitz, Ernestine Großmann, and Christian Schulz. Optimal Neighborhood Exploration for Dynamic Independent Sets. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 1–14. Society for Industrial and Applied Mathematics, 2025.

[33] Jannick Borowitz, Ernestine Großmann, Christian Schulz, and Dominik Schweisgut. Finding Optimal 2-Packing Sets on Arbitrary Graphs at Scale. *arXiv: 2308.15515*, 2023.

[34] Salim Bouamama, Christian Blum, and Abdellah Boukerram. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. *Applied Soft Computing*, 12(6):1632–1639, 2012.

[35] Sergiy Butenko. *Maximum Independent Set and Related Problems, with Applications*. University of Florida, 2003.

[36] Sergiy Butenko and Svyatoslav Trukhanov. Using critical sets to solve the maximum independent set problem. *Operations Research Letters*, 35(4):519–524, 2007.

[37] Shaowei Cai, Wenying Hou, Jinkun Lin, and Yuanjie Li. Improving local search for minimum weight vertex cover by dynamic strategies. In *IJCAI*, pages 1412–1418, 2018.

[38] Shaowei Cai, Yuanjie Li, Wenying Hou, and Haoran Wang. Towards faster local search for minimum weight vertex cover on massive graphs. *Information Sciences*, 471:64–79, 2019.

[39] Shaowei Cai and Jinkun Lin. Fast solving maximum weight clique problem in massive graphs. In *IJCAI*, pages 568–574, 2016.

[40] Shaowei Cai, Jinkun Lin, Yiyuan Wang, and Darren Strash. A semi-exact algorithm for quickly computing a maximum weight clique in large sparse graphs. *Journal of Artificial Intelligence Research*, 72:39–67, 2021.

[41] Ümit V. Çatalyürek, Karen D. Devine, Marcelo Fonseca Faraj, Lars Gottesbüren, Tobias Heuer, Henning Meyerhenke, Peter Sanders, Sebastian Schlag, Christian Schulz, Daniel Seemaier, and Dorothea Wagner. More recent advances in (hyper)graph partitioning. *arXiv: 2205.13202*, 2022.

[42] James D. Chandler, Wyatt J. Desormeaux, Teresa W. Haynes, and Stephen T. Hedetniemi. Neighborhood-restricted [≤ 2] -achromatic colorings. *Discrete Applied Mathematics*, 207:39–44, 2016.

[43] Lijun Chang. Efficient maximum clique computation and enumeration over large sparse graphs. *Vldb Journal*, 29(5):999–1022, 2020.

[44] Lijun Chang, Wei Li, and Wenjie Zhang. Computing A Near-Maximum Independent Set in Linear Time by Reducing-Peeling. In *Proceedings of the International Conference on Management of Data*, pages 1181–1196. ACM, 2017.

[45] Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science*, pages 370–381. IEEE Computer Society, 2019.

[46] Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008.

[47] Ernie J Cockayne, Paul A Dreyer Jr, Sandra M Hedetniemi, and Stephen T Hedetniemi. Roman domination in graphs. *Discrete mathematics*, 278(1-3):11–22, 2004.

[48] Alessio Conte, Donatella Firmani, Maurizio Patrignani, and Riccardo Torlone. A meta-algorithm for finding large k-plexes. *Knowledge and Information Systems*, 63(7):1745–1769, 2021.

[49] Marek Cygan. Improved approximation for 3-dimensional matching via bounded pathwidth local search. In *FOCS*, pages 509–518. IEEE, 2013.

[50] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[51] Marek Cygan, Fabrizio Grandoni, and Monaldo Mastrolilli. How to sell hyperedges: The hypermatching assignment problem. In *Proceedings of the 24th Symposium on Discrete Algorithms*, pages 342–351. SIAM, 2013.

[52] Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Accelerating local search for the maximum independent set problem. In *Experimental Algorithms (SEA 2016)*, volume 9685, pages 118–133. Springer, 2016.

[53] T. A. Davis and Y. Hu. The university of Florida sparse matrix collection. *ACM Transacions on Mathematical Software*, 38(1):1:1–1:25, 2011.

[54] Richard Dawkins. *The Selfish Gene*. Oxford University Press, 1976.

[55] Camil Demetrescu and Giuseppe F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. *ACM Transactions on Algorithms*, 2(4):578–601, 2006.

[56] Yihua Ding, James Z Wang, and Pradip K. Srimani. Self-stabilizing algorithm for maximal 2-packing with safe convergence in an arbitrary graph. *IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 747–754, 2014.

[57] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[58] Christof Doll, Tanja Hartmann, and Dorothea Wagner. Fully-dynamic hierarchical graph clustering using cut trees. In *Proceedings of Workshop on Algorithms and Data Structures*, volume 6844, pages 338–349, 2011.

[59] Yuanyuan Dong, Andrew V. Goldberg, Alexander Noe, Nikos Parotsidis, Mauricio G. C. Resende, and Quico Spaen. A metaheuristic algorithm for large maximum weight independent set problems. *Networks. An International Journal*, 2022.

[60] Yuanyuan Dong, Andrew V Goldberg, Alexander Noe, Nikos Parotsidis, Mauricio GC Resende, and Quico Spaen. New instances for maximum weight independent set from a vehicle routing application. In *Operations Research Forum*, volume 2, pages 1–6. Springer, 2021.

[61] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W [1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.

[62] Doratha E. Drake and Stefan Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85(4):211–213, 2003.

[63] Andre Droschinsky, Petra Mutzel, and Erik Thordsen. Shrinking trees not blossoms: A recursive maximum matching approach. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 146–160. SIAM, 2020.

[64] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of The Acm*, 61(1), 2014.

[65] Fanny Dufossé, Kamer Kaya, Ioannis Panagiotas, and Bora Uçar. Effective heuristics for matchings in hypergraphs. In *Proceedings of the International Symposium on Experimental Algorithms (SEA)*, pages 248–264. Springer, 2019.

[66] Ch. Ebenegger, P.L. Hammer, and D. De Werra. Pseudo-Boolean functions and stability of graphs. In *North-Holland Mathematics Studies*, volume 95, pages 83–97. Elsevier, 1984.

[67] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[68] Mourad El Ouali, Antje Fretwurst, and Anand Srivastav. Inapproximability of b-Matching in k-Uniform hypergraphs. In *Proceedings of WALCOM: Algorithms and Computation*, pages 57–69, Berlin, Heidelberg, 2011. Springer.

[69] Mourad El Ouali and Gerold Jäger. The b-Matching problem in hypergraphs: Hardness and approximability. In *Combinatorial Optimization and Applications*, pages 200–211. Springer, 2012.

[70] Roman Erhardt, Kathrin Hanauer, Nils Kriege, Christian Schulz, and Darren Strash. Improved exact and heuristic algorithms for maximum weight clique. *arXiv:2302.00458*, 2023.

[71] Yi Fan, Nan Li, Chengqian Li, Zongjie Ma, Longin Jan Latecki, and Kaile Su. Restart and random walk in local search for maximum vertex weight cliques with evaluations in clustering aggregation. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 622–630, 2017.

[72] Zhiwen Fang, Chu-Min Li, and Ke Xu. An exact algorithm based on maxsat reasoning for the maximum weight clique problem. *Journal of Artificial Intelligence Research*, 55:799–833, 2016.

[73] Michael R. Fellows. Blow-ups, win/win's, and crown rules: Some new directions in FPT. In *Graph-Theoretic Concepts in Computer Science: 29th International Workshop, WG 2003. Elspeet, the Netherlands, June 19-21, 2003. Revised Papers 29*, pages 1–12. Springer, 2003.

[74] S M Ferdous, Alex Pothen, Arif Khan, Ajay Panyala, and Mahantesh Halappanavar. A parallel approximation algorithm for maximizing submodular b-matching. In *Proceedings of the SIAM Conference on Applied and Computational Discrete Algorithms (ACDA)*, pages 45–56. SIAM, 2021.

[75] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[76] Aleksander Figiel, Vincent Froese, André Nichterlein, and Rolf Niedermeier. There and Back Again: On Applying Data Reduction Rules by Undoing Others. In *Proceedings of the 30th European Symposium on Algorithms (ESA)*, volume 244, pages 53:1–53:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[77] Alejandro Flores-Lamas. Personal Communication, 2023.

[78] Alejandro Flores-Lamas, José Alberto Fernández-Zepeda, and Joel Antonio Trejo-Sánchez. Algorithm to find a maximum 2-packing set in a cactus. *Theoretical Computer Science*, 725:31–51, 2018.

[79] Alejandro Flores-Lamas, José Alberto Fernández-Zepeda, and Joel Antonio Trejo-Sánchez. A distributed algorithm for a maximal 2-packing set in Halin graphs. *Journal of Parallel and Distributed Computing*, 142:62–76, 2020.

[80] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer, 2006.

[81] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing.* Cambridge University Press, 2019.

[82] Daniele Frigioni, Mario Ioffreda, Umberto Nanni, and Giulio Pasqualone. Experimental analysis of dynamic algorithms for the single-source shortest-path problem. *ACM Journal of Experimental Algorithmics*, 3:5, 1998.

[83] Daniel Funke, Sebastian Lamm, Ulrich Meyer, Manuel Penschuck, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free massively distributed graph generation. *Journal of Parallel Distributed Computing*, 131:200–217, 2019.

[84] Martin Fürer and Huiwen Yu. Approximating the k-Set Packing problem by local improvements. In *ISCO*, pages 408–420, Germany, 2014. Springer.

[85] Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of STOC*, pages 448–456, 1983.

[86] Harold N Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 434–443, 1990.

[87] Martin Gairing, Robert M. Geist, Stephen T. Hedetniemi, and Petter Kristiansen. A self-stabilizing algorithm for maximal 2-packing. *Nordic Journal of Computing*, 11:1–11, 2004.

[88] Martin Gairing, Wayne Goddard, Stephen T. Hedetniemi, Petter Kristiansen, and Alice A. McRae. Distance-two information in self-stabilizing algorithms. *Parallel Processing Letters*, 14(03n04):387–398, 2004.

[89] Martin Gairing, Stephen T. Hedetniemi, Petter Kristiansen, and Alice A. McRae. Self-stabilizing algorithms for {k}-domination. In *Self-Stabilizing Systems*, pages 49–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[90] Xiangyu Gao, Jianzhong Li, and Dongjing Miao. Dynamic approximate maximum independent set on massive graphs. In *Proceedings of the IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1835–1847. IEEE, 2022.

[91] Alexander Gellner, Sebastian Lamm, Christian Schulz, Darren Strash, and Bogdán Zaválnij. Boosting Data Reduction for the Maximum Weight Independent Set Problem Using Increasing Transformations. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 128–142. SIAM, 2021.

[92] Giorgos Georgiadis and Marina Papatriantafilou. Overlays with preferences: Distributed, adaptive approximation algorithms for matching with preference lists. *Algorithms*, 6(4):824–856, 2013.

[93] Fred Glover. Tabu Search—Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[94] J.M. Gómez Soto, J. Leaños, L.M. Ríos-Castro, and L.M. Rivera. The packing number of the double vertex graph of the path graph. *Discrete Applied Mathematics*, 247:327–340, 2018.

[95] Kazushige Goto and Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):1–25, 2008.

[96] Lars Gottesbüren, Tobias Heuer, Nikolai Maas, Peter Sanders, and Sebastian Schlag. Scalable high-quality hypergraph partitioning. *arXiv: 2303.17679*, 2023.

[97] Nicholas Gould and Jennifer Scott. A Note on Performance Profiles for Benchmarking Software. *ACM Transactions of Mathematical Software*, 43(2):15:1–15:5, 2016.

[98] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *Journal of Experimental Algorithmics (JEA)*, 13:2–2, 2009.

[99] Ernestine Großmann, Tobias Heuer, Christian Schulz, and Darren Strash. The PACE 2022 parameterized algorithms and computational experiments challenge: Directed feedback vertex set. In *Proceedings of the 17th International Symposium on Parameterized and Exact Computation*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022.

[100] Ernestine Großmann, Felix Joos, Henrik Reinstädtler, and Christian Schulz. Engineering Hypergraph b-Matching Algorithms. *arXiv: 2408.06924*, 2024.

[101] Ernestine Großmann, Sebastian Lamm, Christian Schulz, and Darren Strash. Finding Near-Optimal Weight Independent Sets at Scale. *arXiv:2208.13645*, 2023.

[102] Ernestine Großmann, Sebastian Lamm, Christian Schulz, and Darren Strash. Finding Near-Optimal Weight Independent Sets at Scale. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 293–302. ACM, 2023.

[103] Ernestine Großmann, Sebastian Lamm, Christian Schulz, and Darren Strash. Finding Near-Optimal Weight Independent Sets at Scale. *Journal of Graph Algorithms and Applications*, 28(1):439–473, 2024.

[104] Ernestine Großmann, Kenneth Langedal, and Christian Schulz. Accelerating reductions using graph neural networks and a new concurrent local search for the maximum weight independent set problem. *arXiv:2412.14198*, 2024.

[105] Ernestine Großmann, Kenneth Langedal, and Christian Schulz. A Comprehensive Survey of Data Reduction Rules for the Maximum Weighted Independent Set Problem. *arXiv:2412.09303*, 2024.

[106] Martin Grötschel and Olaf Holland. Solving matching problems with linear programming. *Mathematical Programming*, 33:243–259, 1985.

[107] Jiewei Gu, Weiguo Zheng, Yuzheng Cai, and Peng Peng. Towards computing a near-maximum weighted independent set on massive graphs. In *Proceedings of the 27th Conference on Knowledge Discovery & Data Mining*, pages 467–477, 2021.

[108] Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for Maximal Independent Set and other problems. In *Proceedings of the 4th Symposium on Simplicity in Algorithms (SOSA)*, 2021.

[109] Gurobi Optimization, LLC. Gurobi optimizer, 2023.

[110] Mahantesh Halappanavar. *Algorithms for Vertex-Weighted Matching in Graphs*. PhD thesis, Old Dominion University, 2009.

[111] William K. Hale. Frequency assignment: Theory and applications. *IEEE*, 68(12):1497–1514, 1980.

[112] Magnús M. Halldórsson, Jan Kratochvíl, and Jan Arne Telle. Independent sets with domination constraints. *Discrete Applied Mathematics*, 99(1):39–54, 2000.

[113] Stefan Haller and Bogdan Savchynskyy. A Bregman-Sinkhorn Algorithm for the Maximum Weight Independent Set Problem. *arXiv:2408.02086*, 2024.

[114] Stefan Haller and Bogdan Savchynskyy. Personal Communication, 2024.

[115] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[116] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms. *arXiv:2102.11169*, 2021.

[117] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.

[118] Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *computational complexity*, 15(1):20–39, 2006.

[119] Monika Henzinger, Shahbaz Khan, Richard Paul, and Christian Schulz. Dynamic matching algorithms in practice. In *Proceedings of the 28th European Symposium on Algorithms (ESA)*, volume 173, pages 1–20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[120] Monika Henzinger, Alexander Noe, and Christian Schulz. Practical fully dynamic minimum cut algorithms. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 13–26. SIAM, 2022.

[121] Demian Hespe, Sebastian Lamm, and Christian Schorr. Targeted Branching for the Maximum Independent Set Problem. In *Proceedings of the 19th International Symposium on Experimental Algorithms (SEA)*, volume 190, pages 17:1–17:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[122] Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered: The winning solver from the PACE 2019 challenge, vertex cover track. In *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*, pages 1–11. SIAM, 2020.

[123] Manuel Holtgrewe, Peter Sanders, and Christian Schulz. Engineering a scalable high quality graph partitioner. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12, 2010.

[124] Wei Hou, Limin Meng, Xuqing Ke, and Lingjun Zhong. Dynamic load balancing algorithm based on optimal matching of weighted bipartite graph. *IEEE access : practical innovations, open solutions*, 10:127225–127236, 2022.

[125] Bert Huang and Tony Jebara. Fast b-matching via sufficient selection belief propagation. In *Proceedings of the International Conference Artificial Intelligence and Statistics*, pages 361–369, 2011.

[126] IBM. IBM ILOG CPLEX Optimizer. International Business Machines Corporation.

[127] Hua Jiang, Chu-Min Li, Yanli Liu, and Felip Manya. A two-stage maxsat reasoning approach for the maximum weight clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[128] Hua Jiang, Chu-Min Li, and Felip Manya. An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 830–838, 2017.

[129] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. A new upper bound based on vertex partitioning for the maximum k-plex problem. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1689–1696. International Joint Conferences on Artificial Intelligence Organization, 2021.

[130] Raka Jovanovic and Milan Tuba. An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. *Applied Soft Computing*, 11(8):5360–5366, 2011.

[131] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of Symposium on the Complexity of Computer Computation*, pages 85–103. Plenum Press, New York, 1972.

[132] Richard M. Karp and Michael Sipser. Maximum matching in sparse random graphs. In *Proceedings of the Foundations of Computer Science*, pages 364–375. IEEE, 1981.

[133] Ioannis Katsikarelis, Michael Lampis, and Vangelis Paschos. Improved (In-)Approximability Bounds for d-Scattered Set. *Journal of Graph Algorithms and Applications*, 27(3):219–238, 2023.

[134] Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. The PACE 2021 parameterized algorithms and computational experiments challenge: Cluster editing. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2021.

[135] Angus Kenny, Xiaodong Li, and Andreas T. Ernst. A merge search algorithm and its application to the constrained pit problem in mining. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 316–323, New York, NY, USA, 2018. Association for Computing Machinery.

[136] Arif Khan, Alex Pothen, Md. Mostofa Ali Patwary, Nadathur Rajagopalan Satish, Narayanan Sundaram, Fredrik Manne, Mahantesh Halappanavar, and Pradeep Dubey. Efficient approximation algorithms for weighted b-matching. *Scientific Computing*, 2016.

[137] Jin Kim, Inwook Hwang, Yong-Hyuk Kim, and Byung-Ro Moon. Genetic approaches for graph partitioning: A survey. In *Proceedings of the 13th Genetic and Evolutionary Computation Conference (GECCO)*, pages 473–480. ACM, 2011.

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

[139] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016.

[140] Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. In *ESA*, volume 112, pages 53:1–53:13, 2018.

[141] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, pages 508–520. Springer, 2009.

[142] Christos Koufogiannakis and Neal E. Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. In *Proceedings of the International Symposium on Distributed Computing*, pages 221–238. Springer, 2009.

[143] Ioannis Krommidas and Christos D. Zaroliagis. An experimental study of algorithms for fully dynamic transitive closure. *ACM Journal of Experimental Algorithmics*, 12:1.6:1–1.6:22, 2008.

[144] Piotr Krysta. Greedy approximation via duality for packing, combinatorial auctions and routing. In *MFCS*, pages 615–627. Springer, 2005.

[145] Jérôme Kunegis. KONECT: The Koblenz network collection. In *Proceedings of the 22nd World Wide Web Conference, WWW '13*, pages 1343–1350, 2013.

[146] P.J.M. Laarhoven van and E.H.L. Aarts. *Simulated Annealing : Theory and Applications*. Reidel, Dordrecht, 1987.

[147] Sebastian Lamm, Peter Sanders, and Christian Schulz. Graph partitioning for independent sets. In *Proceedings of the International Symposium on Experimental Algorithms (SEA)*, pages 68–81. Springer, 2015.

[148] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, 2017.

[149] Sebastian Lamm, Christian Schulz, Darren Strash, Robert Williger, and Huashuo Zhang. Exactly solving the maximum weight independent set problem on large real-world graphs. In *Proceedings of the 21st Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 144–158. SIAM, 2019.

[150] Sebastian Lamm, Christian Schulz, Darren Strash, Robert Williger, and Huashuo Zhang. KaMIS - Karlsruhe Maximum Independent Sets Homepage, 2024.

[151] Ailsa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica : journal of the Econometric Society*, 28(3):497–520, 1960.

[152] Kenneth Langedal, Demian Hespe, and Peter Sanders. Targeted Branching for the Maximum Independent Set Problem Using Graph Neural Networks. In *Proceedings of the 22nd International Symposium on Experimental Algorithms (SEA)*, volume 301, pages 20:1–20:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[153] Kenneth Langedal, Johannes Langguth, Fredrik Manne, and Daniel Thilo Schroeder. Efficient minimum weight vertex cover heuristics using graph neural networks. In *Proceedings of the 20th International Symposium on Experimental Algorithms (SEA)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[154] Juho Lauri, Sourav Dutta, Marco Grassia, and Deepak Ajwani. Learning fine-grained search space pruning and heuristics for combinatorial optimization. *Journal of Heuristics*, 29(2):313–347, 2023.

[155] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, 2014.

[156] Ruizhi Li, Shuli Hu, Shaowei Cai, Jian Gao, Yiyuan Wang, and Minghao Yin. NuMWVC: A novel local search for minimum weighted vertex cover problem. *Journal of the Operational Research Society*, 71(9):1498–1509, 2020.

[157] Ruizhi Li, Shuli Hu, Haochen Zhang, and Minghao Yin. An efficient local search framework for the minimum weighted vertex cover problem. *Information Sciences*, 372:428–445, 2016.

[158] Yalun Li, Zhengyi Chai, Hongling Ma, and Sifeng Zhu. An evolutionary game algorithm for minimum weighted vertex cover problem. *Soft Computing*, 27(21):16087–16100, 2023.

[159] Jinkun Lin, Shaowei Cai, Chuan Luo, and Kaile Su. A reduction based method for coloring very large graphs. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 517–523, 2017.

[160] Jianfeng Liu, Sihong Shao, and Chaorui Zhang. Application of causal inference techniques to the maximum weight independent set problem. *arXiv:2301.05510*, 2023.

[161] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer, 2003.

[162] Fredrik Manne and Mahantesh Halappanavar. New effective multithreaded matching algorithms. In *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium*, pages 519–528. IEEE, 2014.

[163] Fredrik Manne and Morten Mjelde. A memory efficient self-stabilizing algorithm for maximal k-Packing. In *Stabilization, Safety, and Security of Distributed Systems: 8th International Symposium*, pages 428–439. Springer, 2006.

[164] Julián Mestre. Greedy in approximation algorithms. In *Proceedings of the 14th Annual European Symposium*, pages 528–539. Springer, 2006.

[165] Silvio Micali and Vijay V. Vazirani. An O($\sqrt{}$|V||E|) algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Symposium on Foundations of Computer Science*, pages 17–27. IEEE Computer Society, 1980.

[166] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Evolutionary Computation*, 4(2):113–131, 1996.

[167] Morten Mjedle. K-packing and K-domination on tree graphs. Master's thesis, University of Bergen, 2004.

[168] Pablo Moscato, Carlos Cotta, and Alexandre Mendes. Memetic algorithms. *New optimization techniques in engineering*, 141:53–85, 2004.

[169] Matthias Müller-Hannemann and Alexander Schwartz. Implementing weighted b-matching algorithms: Insights from a computational study. *Journal of Experimental Algorithmics*, 5:8–es, 2000.

[170] George L Nemhauser and Leslie E Trotter Jr. Properties of vertex packing and independence system polyhedra. *Mathematical programming*, 6(1):48–61, 1974.

[171] Meike Neuwohner. Passing the limits of pure local search for weighted k-set packing. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1090–1137. SIAM, 2023.

[172] Bruno Nogueira, Rian G. S. Pinheiro, and Anand Subramanian. A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optimization Letters*, 12(3):567–583, 2018.

[173] Patric R.J. Östergård. A new algorithm for the maximum-weight clique problem. *Electronic Notes in Discrete Mathematics*, 3:153–156, 1999.

[174] David A. Papa and Igor L. Markov. Hypergraph partitioning and clustering. In *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.

[175] Ojas Parekh and David Pritchard. Generalized hypergraph matching via iterated packing and local ratio. In *WAOA*, pages 207–223. Springer, 2015.

[176] Marco Pavone, Amin Saberi, Maximilian Schiffer, and Matt Wu Tsao. Online hypergraph matching with delays. *Operations Research*, 70(4):2194–2212, 2022.

[177] Manuel Penschuck, Ulrik Brandes, Michael Hamann, Sebastian Lamm, Ulrich Meyer, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in scalable network generation. *arXiv:2003.00736*, 2020.

[178] Rick Plachetta and Alexander van der Grinten. Sat-and-reduce for vertex cover: Accelerating branch-and-reduce by sat solving. In *2021 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 169–180. SIAM, 2021.

[179] R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *STACS*, volume 1563, pages 259–269. Springer, 1999.

[180] Julia Preusse, Jérôme Kunegis, Matthias Thimm, Thomas Gottron, and Steffen Staab. Structural dynamics of knowledge networks. In *Proceedings of the International Conference on Weblogs and Social Media*, 2013.

[181] Wayne Pullan. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization*, 6(2):214–219, 2009.

[182] Huaxin Qiu, Changhao Sun, Xiaochu Wang, Wei Sun, and Qingrui Zhou. A population-based game-theoretic optimizer for the minimum weighted vertex cover. *Applied Soft Computing*, 116:108272, 2022.

[183] Steffen Rebennack, Marcus Oswald, Dirk Oliver Theis, Hanna Seitz, Gerhard Reinelt, and Panos M Pardalos. A branch and cut solver for the maximum stable set problem. *Journal of combinatorial optimization*, 21(4):434–457, 2011.

[184] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 4292–4293, Austin, Texas, 2015. AAAI Press.

[185] Pedro V. Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Transactions on Graphics (TOG)*, 27(5):1–9, 2008.

[186] Peter Sanders. Algorithm Engineering – An Attempt at a Definition. In *Efficient Algorithms*, volume 5760, pages 321–340, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[187] Peter Sanders and Christian Schulz. KaHIP – karlsruhe high qualtity partitioning homepage.

[188] Peter Sanders and Christian Schulz. Engineering multilevel graph partitioning algorithms. In *Proceedings of the 9th European Symposium on Algorithms (ESA)*, volume 6942, pages 469–480. Springer, 2011.

[189] Peter Sanders and Christian Schulz. Advanced multilevel node separator algorithms. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA)*, volume 9685, pages 294–309. Springer, 2016.

[190] Peter Sanders, Christian Schulz, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. *Encyclopedia of social network analysis and mining Springer*, 2014.

[191] Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. k-way hypergraph partitioning via n-level recursive bisection. In *Proceedings of the 18th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 53–67, 2016.

[192] Christian Schulz and Darren Strash. Graph partitioning: Formulations and applications to big data. In *Encyclopedia of Big Data Technologies*, pages 858–864. Springer International Publishing, 2019.

[193] Zhengnan Shi. A self-stabilizing algorithm to maximal 2-packing with improved complexity. *Information Processing Letters*, 112(13):525–531, 2012.

[194] Shyong Jian Shyu, Peng-Yeng Yin, and Bertrand MT Lin. An ant colony optimization algorithm for the minimum weight vertex cover problem. *Annals of Operations Research*, 131:283–304, 2004.

[195] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[196] Darren Strash. On the power of simple reductions for the maximum independent set problem. In *Proceedings of the International Computing and Combinatorics Conference*, pages 345–356. Springer, 2016.

[197] Darren Strash and Louise Thompson. Effective data reduction for the vertex clique cover problem. In *Proceedings of the Symposium on Algorithm Engineering and Experiments*, pages 41–53. SIAM, 2022.

[198] Thomas Stützle and Rubén Ruiz. Iterated Local Search. *Handbook of heuristics*, 1:2, 2018.

[199] Sándor Szabó and Bogdán Zavalnij. Combining algorithms for vertex cover and clique search. In *Proceedings of the 22nd International Multiconference*, pages 71–74, 2019.

[200] Yuting Tan, Junfeng Zhou, Xinqi Rong, Ming Du, and Caiyun Qi. Efficient computation of maximum weighted independent sets on weighted dynamic graph. *The Journal of Supercomputing*, 80(8):10418–10443, 2024.

[201] Theophile Thiery and Justin Ward. An improved approximation for maximum weighted ¡italic¿k¡/italic¿-set packing. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 1138–1162. SIAM, 2023.

[202] Joel Antonio Trejo-Sánchez, Daniel Fajardo-Delgado, and J Octavio Gutierrez-Garcia. A genetic algorithm for the maximum 2–packing set problem. *International Journal of Applied Mathematics and Computer Science*, 30(1):173–184, 2020.

[203] Joel Antonio Trejo-Sánchez and José Alberto Fernández-Zepeda. A self-stabilizing algorithm for the maximal 2-packing in a cactus graph. In *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 863–871. IEEE, 2012.

[204] Joel Antonio Trejo-Sánchez and José Alberto Fernández-Zepeda. Distributed algorithm for the maximal 2-packing in geometric outerplanar graphs. *Journal of Parallel and Distributed Computing*, 74(3):2193–2202, 2014.

[205] Joel Antonio Trejo-Sánchez, José Alberto Fernández-Zepeda, and Julio César Ramírez-Pacheco. A self-stabilizing algorithm for a maximal 2-packing in a cactus graph under any scheduler. *International Journal of Foundations of Computer Science*, 28(08):1021–1045, 2017.

[206] Joel Antonio Trejo-Sánchez, Francisco A. Madera-Ramírez, José Alberto Fernández-Zepeda, José L. uis López-Martínez, and Alejandro Flores-Lamas. A fast approximation algorithm for the maximum 2-packing set problem on planar graphs. *Optimization Letters*, 14:1435–1454, 2023.

[207] Volker Turau. Efficient transformation of distance-2 self-stabilizing algorithms. *Journal of Parallel and Distributed Computing*, 72(4):603–612, 2012.

[208] Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015.

[209] Natarajan Viswanathan, Charles Alpert, Cliff Sze, Zhuo Li, and Yaoguang Wei. The DAC 2012 routability-driven placement contest and benchmark suite. In *Proceedings of the 49th Annual Design Automation Conference*, pages 774–782, New York, NY, USA, 2012. Association for Computing Machinery.

[210] Chris Walshaw. Graph partitioning archive, 2000.

[211] Luzhi Wang, Chu-Min Li, Junping Zhou, Bo Jin, and Minghao Yin. An exact algorithm for minimum weight vertex cover problem in large graphs. *arXiv:1903.05948*, 2019.

[212] Yang Wang, Zhipeng Lü, and Abraham P. Punnen. A fast and robust heuristic algorithm for the minimum weight vertex cover problem. *IEEE access : practical innovations, open solutions*, 9:31932–31945, 2021.

[213] Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. SCCWalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artificial Intelligence*, 280:103230, 2020.

[214] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[215] Jeffrey S Warren and Illya V Hicks. Combinatorial branch-and-bound for the maximum weight independent set problem. *Relatório Técnico, Texas A&M University, Citeseer*, 9:17, 2006.

[216] Qinghua Wu, Jin-Kao Hao, and Fred Glover. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196:611–634, 2012.

[217] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[218] Mingyu Xiao, Sen Huang, and Xiaoyu Chen. Maximum Weighted Independent Set: Effective Reductions and Fast Algorithms on Sparse Graphs. *Algorithmica. An International Journal in Computer Science*, 86(5):1293–1334, 2024.

[219] Mingyu Xiao, Sen Huang, Yi Zhou, and Bolin Ding. Efficient Reductions and a Fast Algorithm of Maximum Weighted Independent Set. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3930–3940. ACM / IW3C2, 2021.

[220] Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theoretical Computer Science*, 469:92–104, 2013.

[221] Hong Xu, TK Satish Kumar, and Sven Koenig. A new solver for the minimum weighted vertex cover problem. In *Proceedings of the International Conference on AI and or Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 392–405. Springer, 2016.

[222] Katsuhisa Yamanaka, Shogo Kawaragi, and Takashi Hirayama. Exact Exponential Algorithm for Distance-3 Independent Set Problem. *IEICE Transactions on Information and Systems*, E102.D(3):499–501, 2019.

[223] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.

[224] Weiguo Zheng, Jiewei Gu, Peng Peng, and Jeffrey Xu Yu. Efficient weighted independent set computation over large graphs. In *IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 1970–1973, 2020.

[225] Weiguo Zheng, Chengzhi Piao, Hong Cheng, and Jeffrey Xu Yu. Computing a near-maximum independent set in dynamic graphs. In *Proceedings of the 35th IEEE International Conference on Data Engineering*, pages 76–87. IEEE, 2019.

[226] Weiguo Zheng, Qichen Wang, Jeffrey Xu Yu, Hong Cheng, and Lei Zou. Efficient computation of a near-maximum independent set over evolving graphs. In *Proceedings of the 34th IEEE International Conference on Data Engineering*, pages 869–880. IEEE Computer Society, 2018.

[227] Taoqing Zhou, Zhipeng Lü, Yang Wang, Junwen Ding, and Bo Peng. Multi-start iterated tabu search for the minimum weight vertex cover problem. *Journal of Combinatorial Optimization*, 32:368–384, 2016.

[228] Yi Zhou, Jin-Kao Hao, and Adrien Goëffon. PUSH: A generalized operator for the maximum vertex weight clique problem. *European Journal of Operational Research*, 257(1):41–54, 2017.

A Instance Details

Table A.1: Detailed properties for Instance Set 1. Instances marked with a ⋆ are used for parameter tuning experiments.

finEl	n	m
body⋆	45 087	327 468
ocean	143 437	819 186
pwt	36 519	289 588
rotor⋆	99 617	1 324 862
sphere⋆	16 386	98 304

mesh	n	m
blob	16 068	48 204
buddha⋆	1 087 716	3 263 148
bunny	68 790	206 034
cow	5 036	14 732
dragon⋆	150 000	450 000
dragonsub	600 000	1 800 000
ecat⋆	684 496	2 053 488
face	22 871	68 108
fandisk	8 634	25 636
feline	41 262	123 786
gameguy	42 623	127 700
gargoyle	20 000	60 000
turtle	267 534	802 356
venus	5 672	17 016

ssmc	n	m
ca2010⋆	710 145	3 489 366
fl2010⋆	484 481	2 346 294
ga2010⋆	291 086	1 418 056
il2010	451 554	2 164 464
nh2010	48 837	234 550
ri2010	25 181	125 750

osm	n	m
alabama-1	320	1 162
alabama-2	1 164	38 772
alabama-3	3 504	619 328
alaska-1	31	62
alaska-2	54	312
alaska-3	86	950
arkansas-1	26	38
arkansas-2	55	466
arkansas-3	103	2 752
california-1	77	260
california-2	231	6 148
california-3	587	55 072
canada-1	189	480
canada-2	449	5 894
canada-3	943	40 482
colorado-1	128	464

snap	n	m
as-skitter	1 696 415	22 190 596
ca-AstroPh	18 772	396 100
ca-CondMat	23 133	186 878
ca-GrQc	5 242	28 968
ca-HepPh	12 008	236 978
ca-HepTh	9 877	51 946
com-amazon	334 863	1 851 738
com-youtube	1 134 890	5 975 248
email-Enron	36 692	367 662
email-EuAll	265 214	728 962
loc-gowalla	196 591	1 900 654
p2p-G.04	10 876	79 988
p2p-G.05	8 846	63 678
p2p-G.06	8 717	63 050
p2p-G.08	6 301	41 554
p2p-G.09	8 114	52 026
p2p-G.24	26 518	130 738
p2p-G.25	22 687	109 410
p2p-G.30	36 682	176 656
p2p-G.31	62 586	295 784
roadNet-CA	1 965 206	5 533 214
RoadNet-PA⋆	1 088 092	3 083 796
roadNet-TX	1 379 917	3 843 320
soc-Ep.1	75 879	811 480
soc-LiveJ.1	4 847 571	85 702 474
soc-p.-rel.	1 632 803	44 603 928
soc-Sl.0811	77 360	938 360
soc-Sl.0902	82 168	1 008 460
Web-BS.⋆	685 230	13 298 940
web-Google	875 713	8 644 102
Web-ND.⋆	325 729	2 180 216
web-Stanford	281 903	3 985 272
wiki-Talk	2 394 385	9 319 130
wiki-Vote	7 115	201 524

osm	n	m
colorado-2	283	4 052
colorado-3	538	16 730
connec.-1	87	192
connec.-2	211	1 950
connec.-3	367	7 538
delaware-1	2	2
delaware-2	3	6
delaware-3	5	18
d.o.c.-1	2 500	49 302
d.o.c.-2	13 597	3 219 590
d.o.c.-3⋆	46 221	55 458 274
florida-1	475	2 554
florida-2	1 254	33 872
florida-3	2 985	308 086
georgia-1	294	868
georgia-2	746	15 506

Table A.1 – *Continued from previous page*

osm	n	m
georgia-3	1 680	148 252
greenland-1	77	682
greenland-2	686	100 436
greenland-3*	4 986	7 304 722
hawaii-1	411	2 846
hawaii-2	2 875	530 316
hawaii-3	28 006	98 889 842
idaho-1	136	416
idaho-2	552	70 442
idaho-3	4 064	7 848 160
illinois-1	113	404
illinois-2	261	4 276
indiana-1	2	2
indiana-2	2	2
indiana-3	4	12
iowa-1	90	328
iowa-2	155	1 908
kansas-1	190	800
kansas-2	602	32 948
kansas-3	2 732	1 613 824
kentucky-1	381	4 804
kentucky-2	2 453	1 286 856
kentucky-3	19 095	119 067 260
louisiana-1	157	362
louisiana-2	436	6 222
louisiana-3	1 162	74 154
maine-1	38	58
maine-2	81	486
maine-3	143	1 700
maryland-1	104	432
maryland-2	316	9 430
maryland-3	1 018	190 830
massach.-1	413	2 178
massach.-2	1 339	70 898
massach.-3	3 703	1 102 982
mexico-1	175	716
mexico-2	516	18 822
mexico-3	1 096	94 262
michigan-1	133	224
michigan-2	241	1 500
michigan-3	376	4 918
minnesota-1	86	272
minnesota-2	253	5 160
minnesota-3	683	68 376
mississippi-1	74	120
mississippi-2	151	732
mississippi-3	242	2 232
missouri-1	10	12
missouri-2	13	24
missouri-3	17	48
montana-1	109	388
montana-2	307	10 308
montana-3	837	138 586
nebraska-1	40	92
nebraska-2	93	1 468
nebraska-3	145	4 336
nevada-1	89	186
nevada-2	242	3 062
nevada-3	569	30 032
new-hamp.-1	195	604
new-hamp.-2	514	6 738
new-hamp.-3	1 107	36 042
new-jersey-1	4	12
new-jersey-2	4	12
new-jersey-3	4	12
new-mex.-1	3	6
new-mex.-2	3	6
new-mex.-3	3	6
new-york-1	42	236
new-york-2	224	12 798
new-york-3	837	177 456
north-car.-1	93	300
north-car.-2	398	20 232
north-car.-3	1 557	473 478
ohio-1	78	192
ohio-2	211	3 630
ohio-3	482	22 752
oregon-1	381	1 992
oregon-2	1 325	115 034
oregon-3	5 588	5 825 402
penns.-1	193	552
penns.-2	521	7 624
penns.-3	1 148	52 928
puerto-rico-1	60	126
puerto-rico-2	165	2 570
puerto-rico-3	494	53 852
rhode-is.-1	455	3 946
rhode-is.-2*	2 866	590 976
rhode-is.-3	15 124	25 244 438
south-car.-1	75	138
south-car.-2	165	1 426
south-car.-3	317	9 016
tennessee-1	49	78
tennessee-2	100	836
tennessee-3	212	6 430
utah-1	230	618
utah-2	589	9 384
utah-3	1 339	85 744
vermont-1	128	836
vermont-2	766	75 214
vermont-3	3 436	2 272 328
virginia-1	570	2 960
virginia-2	2 279	120 080
virginia-3	6 185	1 331 806
washington-1	713	4 632
washington-2	3 025	304 898
washington-3	10 022	4 692 426
w-virg.-1	65	300
w-virg.-2	317	16 656
w-virg.-3	1 185	251 240
wisconsin-1	54	102
wisconsin-2	89	438
wisconsin-3	136	1 176
wyoming-1	7	22
wyoming-2	8	32
wyoming-3	12	84

Table A.2: Detailed graph properties for the Instance Set 2. Graphs marked with a ⋆ are part of the parameter tuning set for experiments on these instances. We also present the number of vertices $n_{\mathcal{K}}$ and edges $m_{\mathcal{K}}$ of the instances reduced by LearnAndReduce with the configuration *Fast - initial tight* as well as the computed offset and running time t_{red} in seconds.

fe	n	m	$n_{\mathcal{K}}$	$m_{\mathcal{K}}$	offset	t_{red}
body	45 087	163 734	395	3 551	1 672 469	0.60
ocean	143 437	409 593	0	0	7 248 581	3.56
pwt⋆	36 519	144 794	15 239	106 943	813 076	2.17
rotor⋆	99 617	662 431	89 294	690 242	435 335	5.05
sphere	16 386	49 152	0	0	617 816	0.52

mesh	n	m	$n_{\mathcal{K}}$	$m_{\mathcal{K}}$	offset	t_{red}
blob	16 068	24 102	0	0	855 547	0.03
buddha	1 087 716	1 631 574	0	0	57 555 880	2.55
bunny	68 790	103 017	0	0	3 686 960	0.13
dragonsub	600 000	900 000	0	0	32 213 898	1.38
dragon	150 000	225 000	0	0	7 956 530	0.20
ecat	684 496	1 026 744	0	0	36 650 298	2.12
fandisk	8 634	12 818	0	0	463 288	0.02
feline	41 262	61 893	0	0	2 207 219	0.10
gameguy	42 623	63 850	0	0	2 325 878	0.07
gargoyle	20 000	30 000	0	0	1 059 559	0.04
turtle	267 534	401 178	0	0	14 263 005	0.42

osm	n	m	$n_{\mathcal{K}}$	$m_{\mathcal{K}}$	offset	t_{red}
alabama-2	1 164	19 386	0	0	174 309	0.01
alabama-3	3 504	309 664	815	61 637	173 907	1.45
california-2	231	3 074	0	0	47 153	0.02
california-3	587	27 536	359	19 705	35 724	0.21
canada-3	943	20 241	0	0	86 018	0.23
colorado-3	538	8 365	0	0	54 741	0.01
d.-of-c.-1	2 500	24 651	0	0	196 475	0.22
d.-of-c.-2	13 597	1 609 795	2 361	274 529	178 323	5.76
d.-of-c.-3	46 221	27 729 137	26 728	13 406 329	133 708	290.37
florida-3	2 985	154 043	658	41 043	226 379	0.52
georgia-3	1 680	74 126	481	31 388	207 992	0.38
greenland-2	686	50 218	0	0	10 718	0.08
greenland-3	4 986	3 652 361	3 402	2 141 096	7 572	27.39
hawaii-2	2 875	265 158	0	0	125 284	0.15
hawaii-3⋆	28 006	49 444 921	22 849	38 081 016	96 234	935.96
idaho-3	4 064	3 924 080	2 920	2 413 139	70 960	40.03
kansas-3	2 732	806 912	1 209	292 997	84 662	3.79
kentucky-2	2 453	643 428	0	0	97 397	0.27
kentucky-3	19 095	59 533 630	15 943	53 739 580	91 864	1 510.41
louisiana-3	1 162	37 077	0	0	60 024	0.08
maine-3	143	850	0	0	26 734	0.00
maryland-3	1 018	95 415	0	0	45 496	0.10
massachusetts-2	1 339	35 449	0	0	140 095	0.03
massachusetts-3	3 703	551 491	1 626	340 652	136 331	2.98
mexico-2	516	9 411	0	0	94 834	0.01
mexico-3	1 096	47 131	447	20 443	86 467	0.29
minnesota-3	683	34 188	0	0	32 787	0.07

Continued on next page

Table A.2 – *Continued from previous page*

osm	n	m	$n_{\mathcal{K}}$	$m_{\mathcal{K}}$	offset	t_{red}
montana-3	837	69 293	382	39 859	55 531	0.25
nevada-3	569	15 016	0	0	52 036	0.02
new-hampshire-3	1 107	18 021	0	0	116 060	0.12
new-york-2	224	6 399	0	0	14 330	0.01
new-york-3	837	88 728	526	50 572	11 447	0.42
north-carolina-3	1 557	236 739	997	133 106	39 783	0.87
ohio-3	482	11 376	0	0	52 634	0.05
oregon-3	5 588	2 912 701	3 159	1 817 045	162 009	23.75
pennsylvania-3	1 148	26 464	0	0	143 870	0.12
puerto-rico-3	494	26 926	0	0	33 590	0.10
rhode-i.-2	2 866	295 488	498	46 183	174 013	0.52
rhode-i.-3	15 124	12 622 219	12 189	11 225 470	144 106	153.06
tennessee-3	212	3 215	0	0	32 276	0.01
utah-2	589	4 692	0	0	95 087	0.00
utah-3	1 339	42 872	0	0	98 847	0.28
vermont-2	766	37 607	0	0	59 310	0.11
vermont-3*	3 436	1 136 164	1 946	526 463	52 788	5.50
virginia-2	2 279	60 040	0	0	295 867	0.06
virginia-3	6 185	665 903	2 385	293 068	272 730	2.51
washington-2	3 025	152 449	0	0	305 619	0.10
washington-3	10 022	2 346 213	6 671	1 955 568	264 406	14.55
west-virginia-3	1 185	125 620	864	98 399	37 678	0.52

snap	n	m	$n_{\mathcal{K}}$	$m_{\mathcal{K}}$	offset	t_{red}
as-skitter*	1 696 415	11 095 298	1 951	27 277	124 100 687	3.43
com-amazon	334 863	925 869	0	0	19 271 031	0.32
loc-gowalla-edges	196 591	950 327	304	2 938	12 268 682	0.33
roadNet-CA	1 965 206	2 766 607	0	0	111 360 828	2.33
roadNet-PA	1 088 092	1 541 898	0	0	61 731 589	1.19
roadNet-TX	1 379 917	1 921 660	0	0	78 599 946	1.46
soc-LiveJ.*	4 847 571	42 851 237	1 715	28 390	283 975 051	23.47
soc-p.-rel	1 632 803	22 301 964	782 676	14 234 808	53 362 233	644.00
web-BerkStan	685 230	6 649 470	0	0	43 907 482	3.87
web-Google	875 713	4 322 051	0	0	56 326 504	2.03
web-NotreDame	325 729	1 090 108	97	1 357	26 013 618	0.62
web-Stanford	281 903	1 992 636	0	0	17 792 930	1.39

ssmc	n	m	$n_{\mathcal{K}}$	$m_{\mathcal{K}}$	offset	t_{red}
ca2010	710 145	1 744 683	0	0	16 869 550	7.34
fl2010	484 481	1 173 147	2 045	8 459	8 707 619	3.62
ga2010	291 086	709 028	0	0	4 644 417	0.89
il2010	451 554	1 082 232	0	0	5 998 539	5.63
nh2010	48 837	117 275	0	0	588 996	0.16
ri2010	25 181	62 875	0	0	459 275	0.19

Table A.3: Graph properties for the vehicle routing instances in Instance Set 4. Additionally to the number of vertices n and edges m of the original graph, we also present the number of vertices $n_{\mathcal{K}}$ and edges $m_{\mathcal{K}}$ of these reduced instances computed by LearnAndReduce with the configuration *Fast - initial tight* as well as the computed offset and reduction time t_{red} in seconds. Instances marked with a $\star$ are used for parameter tuning.

large vr	n	m	$n_{\mathcal{K}}$	$m_{\mathcal{K}}$	offset	red.time
CR-S-L-1*	863 368	331 203 970	855 833	328 396 194	26 070	3 671.65
CR-S-L-2	880 974	342 158 741	873 941	339 503 913	20 282	5 234.35
CR-S-L-4	881 910	344 057 350	874 668	341 151 292	19 400	5 315.89
CR-S-L-6	578 244	219 717 582	572 888	217 572 336	25 208	3 056.68
CR-S-L-7	270 067	94 109 215	266 591	92 817 339	26 955	858.97
CR-T-C-1	602 472	194 753 152	594 883	192 451 235	59 577	2 668.29
CR-T-C-2	652 497	215 694 927	645 058	213 228 808	32 598	2 870.86
CR-T-D-4	651 861	220 480 534	644 845	218 142 885	22 898	2 342.34
CR-T-D-6	381 380	115 082 762	376 351	113 599 041	31 498	1 393.38
CR-T-D-7	163 809	43 028 583	160 630	42 184 292	29 046	341.15
CW-S-L-1	411 950	283 860 106	409 559	282 656 866	11 718	4 465.88
CW-S-L-2	443 404	315 569 883	441 093	314 281 707	8 310	5 816.47
CW-S-L-4*	430 379	303 042 962	427 984	301 676 399	6 009	5 262.94
CW-S-L-6	267 698	171 132 761	265 820	170 150 154	9 603	2 512.19
CW-S-L-7	127 871	78 459 291	126 866	77 941 263	4 283	859.79
CW-T-C-1	266 403	144 634 578	264 389	143 674 135	10 530	2 064.78
CW-T-C-2*	194 413	111 098 006	192 871	110 337 336	10 128	1 387.53
CW-T-D-4	83 091	37 881 529	82 110	37 453 654	3 108	356.97
CW-T-D-6	83 758	38 781 839	82 723	38 299 758	5 338	400.99

small vr	n	m	$n_{\mathcal{K}}$	$m_{\mathcal{K}}$	offset	t_{red}
MR-D-03	21 499	130 508	17 390	131 591	832 349 816	1.06
MR-D-05	27 621	236 044	24 250	236 758	676 962 870	1.86
MR-D-FN*	30 467	296 369	27 301	297 905	639 582 861	2.46
MR-W-FN	15 639	126 800	14 657	128 925	691 250 460	0.75
MT-D-01	979	3 125	0	0	238 166 485	0.02
MT-D-200	10 880	505 359	10 676	493 364	6 914 835	2.19
MT-D-FN	10 880	604 041	10 682	595 799	45 784 382	2.33
MT-W-01	1 006	2 411	0	0	312 121 568	< 0.01
MT-W-200	12 320	515 871	11 472	474 079	85 332 696	2.33
MT-W-FN	12 320	553 895	11 456	503 617	128 611 366	2.43
MW-D-01	3 988	13 556	2 539	13 853	307 196 278	0.18
MW-D-20	20 054	606 318	19 473	600 915	104 526 987	3.15
MW-D-40*	33 563	1 879 303	33 062	1 870 537	74 805 076	8.33
MW-D-FN	47 504	4 017 196	47 070	4 005 378	60 228 177	18.80
MW-W-01	3 079	22 664	2 844	22 765	169 735 181	0.16
MW-W-05*	10 790	485 261	10 667	481 661	25 259 548	1.68
MW-W-10	18 023	1 451 813	17 921	1 445 959	16 146 685	4.81
MW-W-FN	22 316	2 275 623	22 215	2 269 047	15 670 946	9.86

Table A.4: Instance properties for Instance Set 3. The class `redOsm` are the `osm` instances reduced using KaMIS BnR [149] for the METAMIS comparison.

redOsm	n	m
alabama-3	1 614	117 426
d.o.c.-2	6 360	592 457
d.o.c.-3	33 367	17 459 296
florida-3	1 069	62 088
greenland-3	3 942	2 348 539
hawaii-3	24 436	40 724 109
idaho-3	3 208	2 864 466
kansas-3	1 605	408 108
kentucky-3	16 871	54 160 431
massach.-3	2 008	373 537
north-car.-3	1 178	189 362
oregon-3	3 670	1 958 180
rhode-is.-2	1 103	81 688
rhode-is.-3	13 031	11 855 557
vermont-3	2 630	811 482
virginia-3	3 867	485 330
washington-3	8 030	2 120 696

Table A.5: Instance properties of benchmark set of static and real dynamic graphs from Instance Set 5. We report the original number of update operations $\mathcal{O}$, after removing obsolete updates (such as parallel edges, self-loops etc.). Note that most of these instances only feature insertions.

mesh	n	$\mathcal{O}$
3elt	4 720	13 722
4elt	15 606	45 878
add20	2 395	7 462
add32	4 960	9 462
citeulike-ui	731 770	842 421
crack	10 240	30 380
cs4	22 499	43 858
cti	16 840	48 232
data	2 851	15 093
fe-4elt2	11 143	32 818
fe-body	45 087	163 734
fe-ocean	143 437	409 593
fe-pwt	36 519	144 794
fe-sphere	16 386	49 152
t60k	60 005	89 440
whitaker3	9 800	28 989
wing	62 032	121 544

osm	n	$\mathcal{O}$
asia	11 950 757	12 711 603
belgium	1 441 295	1 549 970
germany	11 548 845	12 369 181
great-britain	7 733 822	8 156 517
italy	6 686 493	7 013 978
luxembourg	114 599	119 666
netherlands	2 216 688	2 441 238
uk	4 824	6 837

social	n	$\mathcal{O}$
amazon-ratings	2 146 058	477 676
dnc-temporalGraph	2 030	4 384
facebook-wosn-wall	46 953	183 412
haggle	275	2 124
lastfm-band	174 078	894 388
lkml-reply	63 400	159 996
sociopatterns-infections	411	2 765
topology	34 762	107 720

Table A.6: Detailed properties of Instance Set 6. We report the number of vertices n and edges m for each graph, grouped by graph class.

cactus	n	m
cac50	50	52
cac100	100	105
cac150	150	154
cac200	200	208
cac250	250	255
cac300	300	305
cac350	350	350
cac400	400	401
cac450	450	459
cac500	500	510
cac550	550	565
cac600	600	603
cac650	650	658
cac700	700	708
cac750	750	753
cac800	800	811
cac850	850	859
cac900	900	906
cac950	950	960
cac1000	1 000	1 008
adjnoun	112	425
as-22july06	22 963	48 436
celegans_metabolic	453	2 025
celegansneural	297	2 148

small social	n	m
chesapeake	39	170
cond-mat	16 726	47 594
dolphins	62	159
email	1 133	5 451
email-EuAll	16 805	60 260
football	115	613
hep-th	8 361	15 751
jazz	198	2 742
lesmis	77	254
netscience	1 589	2 742
p2p-Gnutella04	6 405	29 215
PGPgiantcompo	10 680	24 316
polbooks	105	441
power	4 941	6 594
soc-Slashdot0902	28 550	379 445
wordassociation-2011	10 617	63 788

planar	n	m
outP500_1	62 320	65 138
outP1000_1	148 564	154 609
outP1500_1	227 107	236 077
outP2000_1	301 431	313 433
outP2500_1	375 728	390 874
outP3000_1	448 689	466 782
outP3500_1	523 959	545 122
outP4000_1	600 173	624 188
outP4500_1	675 339	702 423
outP5000_1	748 383	778 411

erdos	n	m
Erdos37-2	37	73
Erdos37-16	37	66
Erdos37-23	37	68
Erdos37-44	37	76
Erdos37-45	37	58
Erdos38-2	38	68
Erdos38-14	38	81
Erdos38-18	38	83
Erdos38-46	38	67
Erdos38-48	38	68
Erdos39-14	39	72
Erdos39-22	39	61
Erdos39-25	39	80
Erdos39-29	39	75
Erdos39-44	39	80
Erdos40-0	40	71
Erdos40-4	40	74
Erdos40-8	40	87
Erdos40-10	40	72
Erdos40-43	40	75
amazon-2008	735 323	3 523 472
astro-ph	16 706	121 251
caidaRouterLevel	192 244	609 066
citationCiteseer	268 495	1 156 647

large social	n	m
cnr-2000	325 557	2 738 969
coAuthorsCiteseer	227 320	814 134
coAuthorsDBLP	299 067	977 676
cond-mat-2003	31 163	120 029
cond-mat-2005	40 421	175 691
coPapersCiteseer	434 102	16 036 720
coPapersDBLP	540 486	15 245 729
enron	69 244	254 449
G_n_pin_pout	100 000	501 198
loc-brightkite_edges	56 739	212 945
loc-gowalla_edges	196 591	950 327
preferentialAttachment	100 000	499 985
road_central	14 081 816	16 933 413
smallworld	100 000	499 998
web-Google	356 648	2 093 324
road_usa	23 947 347	28 854 312

planar	n	m
outP500_2	73 959	76 976
outP1000_2	151 091	157 100
outP1500_2	226 090	235 100
outP2000_2	301 692	313 729
outP2500_2	373 931	389 003
outP3000_2	451 224	469 413
outP3500_2	529 022	550 144
outP4000_2	600 288	624 264
outP4500_2	677 075	704 222
outP5000_2	750 308	780 191

Table A.7: Detailed graph properties of Instance Set 7. We present the number of vertices $|V|$ and edges $|E|$ of the instances as well as the average degree. The column weights shows the different weight assignements unit (u), uniform (uf), degree (d), geometric (g), hybrid (h) and from file (f).

red2pack	n	m	weights	avg. deg
as-22july06	22 963	48 436	u, uf, d, g, h	4.2
astro-ph	16 706	121 251	u, uf, d, g, h	14.5
cactus1000	1 000	1 008	u, uf, d, g, h	2.0
caidaRouterLevel	192 244	609 066	u, uf, d, g, h	6.3
citationCiteseer	268 495	1 156 647	u, uf, d, g, h	8.6
coAuthorsCiteseer	227 320	814 134	u, uf, d, g, h	7.2
coAuthorsDBLP	299 067	977 676	u, uf, d, g, h	6.5
cond-mat-2003	31 163	120 029	u, uf, d, g, h	7.7
cond-mat-2005	40 421	175 691	u, uf, d, g, h	8.7
coPapersCiteseer	434 102	16 036 720	u, uf, d, g, h	73.9
coPapersDBLP	540 486	15 245 729	u, uf, d, g, h	56.4
hep-th	8 361	15 751	u, uf, d, g, h	3.8
loc-brightkite_edges	56 739	212 945	u, uf, d, g, h	7.5
loc-gowalla_edges	196 591	950 327	u, uf, d, g, h	9.7
netscience	1 589	2 742	u, uf, d, g, h	3.5
power	4 941	6 594	u, uf, d, g, h	2.7
road_central	14 081 816	16 933 413	u, uf, d, g, h	2.4
road_usa	23 947 347	28 854 312	u, uf, d, g, h	2.4
asia	11 950 757	12 711 603	uf, h	2.1
belgium	1 441 295	1 549 970	uf, h	2.2
europe	50 912 018	54 054 660	uf, h	2.1
germany	11 548 845	12 369 181	uf, h	2.1
great-britain	7 733 822	8 156 517	uf, h	2.1
italy	6 686 493	7 013 978	uf, h	2.1
luxembourg	114 599	119 666	uf, h	2.1
netherlands	2 216 688	2 441 238	uf, h	2.2
fe	n	m	weights	avg. deg
body	45 087	163 734	uf	7.3
ocean	143 437	409 593	uf	5.7
pwt	36 519	144 794	uf	7.9
rotor	99 617	662 431	uf	13.3
sphere	16 386	49 152	uf	6.0
mesh	n	m	weights	avg. deg
blob	16 068	24 102	uf	3.0
buddha	1 087 716	1 631 574	uf	3.0
bunny	68 790	103 017	uf	3.0
cow	5 036	7 366	uf	2.9
dragonsub	600 000	900 000	uf	3.0
dragon	150 000	225 000	uf	3.0
ecat	684 496	1 026 744	uf	3.0
face	22 871	34 054	uf	3.0
fandisk	8 634	12 818	uf	3.0
feline	41 262	61 893	uf	3.0
gameguy	42 623	63 850	uf	3.0
gargoyle	20 000	30 000	uf	3.0
turtle	267 534	401 178	uf	3.0
venus	5 672	8 508	uf	3.0

Continued on next page

Table A.7 – *Continued from previous page*

osm	n	m	weights	avg. deg
alabama-AM2	1 164	19 386	f	33.3
alabama-AM3	3 504	309 664	f	176.7
district-of-columbia-AM1	2 500	24 651	f	19.7
district-of-columbia-AM2	13 597	1 609 795	f	236.8
district-of-columbia-AM3	46 221	27 729 137	f	1 199.9
florida-AM2	1 254	16 936	f	27.0
florida-AM3	2 985	154 043	f	103.2
georgia-AM3	1 680	74 126	f	88.2
greenland-AM3	4 986	3 652 361	f	1 465.0
hawaii-AM2	2 875	265 158	f	184.5
hawaii-AM3	28 006	49 444 921	f	3 531.0
idaho-AM3	4 064	3 924 080	f	1 931.1
kansas-AM3	2 732	806 912	f	590.7
kentucky-AM2	2 453	643 428	f	524.6
kentucky-AM3	19 095	59 533 630	f	6 235.5
louisiana-AM3	1 162	37 077	f	63.8
maryland-AM3	1 018	95 415	f	187.5
massachusetts-AM2	1 339	35 449	f	52.9
massachusetts-AM3	3 703	551 491	f	297.9
mexico-AM3	1 096	47 131	f	86.0
new-hampshire-AM3	1 107	18 021	f	32.6
north-carolina-AM3	1 557	236 739	f	304.1
oregon-AM2	1 325	57 517	f	86.8
oregon-AM3	5 588	2 912 701	f	1 042.5
pennsylvania-AM3	1 148	26 464	f	46.1
rhode-island-AM2	2 866	295 488	f	206.2
rhode-island-AM3	15 124	12 622 219	f	1 669.2
utah-AM3	1 339	42 872	f	64.0
vermont-AM3	3 436	1 136 164	f	661.3
virginia-AM2	2 279	60 040	f	52.7
virginia-AM3	6 185	665 903	f	215.3
washington-AM2	3 025	152 449	f	100.8
washington-AM3	10 022	2 346 213	f	468.2
west-virginia-AM3	1 185	125 620	f	212.0
snap	n	m	weights	avg. deg
as-skitter	1 696 415	11 095 298	uf	13.1
ca-AstroPh	18 772	198 050	uf	21.1
ca-CondMat	23 133	93 439	uf,f	8.1
ca-GrQc	5 241	14 484	uf,f	5.5
ca-HepPh	12 008	118 489	uf	19.7
ca-HepTh	9 877	25 973	uf	5.3
com-amazon	334 863	925 869	f	5.5
com-youtube	1 134 890	2 987 624	f	5.3
email-Enron	36 692	183 831	uf,f	10.0
email-EuAll	265 214	364 481	uf	2.7
loc-gowalla_edges	196 591	950 327	f	9.7
p2p-Gnutella04	10 876	39 994	uf	7.4
p2p-Gnutella05	8 846	31 839	uf	7.2
p2p-Gnutella06	8 717	31 525	uf	7.2
p2p-Gnutella08	6 301	20 777	uf	6.6
p2p-Gnutella09	8 114	26 013	uf	6.4
p2p-Gnutella24	26 518	65 369	uf	4.9
p2p-Gnutella25	22 687	54 705	uf	4.8
p2p-Gnutella30	36 682	88 328	uf	4.8

Continued on next page

Table A.7 – *Continued from previous page*

snap	n	m	weights	avg. deg
p2p-Gnutella31	62 586	147 892	uf	4.7
roadNet-CA	1 965 206	2 766 607	uf	2.8
roadNet-PA	1 088 092	1 541 898	uf,f	2.8
roadNet-TX	1 379 917	1 921 660	uf	2.8
soc-Epinions1	75 879	405 740	uf	10.7
soc-LiveJournal1	4 847 571	42 851 237	uf	17.7
soc-pokec-relationships	1 632 803	22 301 964	uf	27.3
soc-Slashdot0811	77 360	469 180	uf	12.1
soc-Slashdot0902	82 168	504 230	uf	12.3
web-BerkStan	685 230	6 649 470	uf,f	19.4
web-Google	875 713	4 322 051	uf	9.9
web-NotreDame	325 729	1 090 108	uf,f	6.7
web-Stanford	281 903	1 992 636	uf	14.1
wiki-Talk	2 394 385	4 659 565	uf	3.9
wiki-Vote	7 115	100 762	uf	28.3
ssmc	n	m	weights	avg. deg
ca2010	710 145	1 744 683	f	4.9
fl2010	484 481	1 173 147	f	4.8
ga2010	291 086	709 028	f	4.9
il2010	451 554	1 082 232	f	4.8
nh2010	48 837	117 275	f	4.8
ri2010	25 181	62 875	f	5.0

B Detailed Results for State-of-the-Art Comparisons

B.1 State-of-the-Art Results Comparing m²wis

Table B.1: Overview of the state-of-the-art algorithms for all graph classes. The table shows the number of best solutions, the geometric mean solution weight $\overline{\omega}$ over all instances in the corresponding class, and the geometric mean time $\overline{t}$ in seconds required to compute $\overline{\omega}$ for each algorithm. Detailed per-instance results are presented in Table B.2.

	finite elemente			mesh			osm			snap			ssmc		
Algorithm	best	$\overline{\omega}$	$\overline{t}$	best	$\overline{\omega}$	$\overline{t}$	best	$\overline{\omega}$	$\overline{t}$	best	$\overline{\omega}$	$\overline{t}$	best	$\overline{\omega}$	$\overline{t}$
KaMIS BnR	1/5	-	-	14/14	**3 054 726**	0.50	132/148	-	-	28/34	-	-	0/6	-	-
GNN-VC	0/5	1 873 906	1 201.98	0/14	3 052 303	827.18	101/148	39 597	4.92	23/34	6 678 015	8.06	0/6	3 208 739	2 845.45
HILS	0/5	1 868 676	1 780.48	0/14	3 050 072	1 418.48	142/148	**39 823**	12.49	12/34	6 671 408	3 338.88	0/6	3 213 935	11 515.77
HtWIS	0/5	1 827 149	0.05	0/14	3 051 422	0.03	86/148	39 520	<0.01	11/34	6 677 139	0.06	0/6	3 206 698	0.13
NuMWVC	0/5	1 841 892	446.40	0/14	3 020 035	682.18	0/148	-	-	0/34	-	-	0/6	3 093 453	4 546.46
Struction	1/5	-	-	14/14	**3 054 726**	0.07	136/148	-	-	31/34	-	-	6/6	**3 218 537**	0.75
m²wis+s	5/5	**1 882 030**	166.87	14/14	**3 054 726**	0.14	146/148	**39 823**	0.01	33/34	6 678 187	0.25	6/6	**3 218 537**	4.02
m²wis	2/5	1 881 878	263.30	13/14	**3 054 726**	0.19	146/148	**39 823**	0.01	34/34	**6 678 200**	0.24	3/6	3 218 499	88.11

Table B.2: Average solution weight f and time t in seconds required to compute f for our set of `finite element` instances. Bold numbers indicate the **best** solution among all algorithms. Rows have a gray background color, if KaMIS BnR or Struction computed an exact solution.

	Instance	KaMIS BnR		GNN-VC		HILS		HtWIS		NuMWVC		Struction		m^{2}wis+s		m^{2}wis	
		t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
fin. el.	body	-	-	3 112.38	1 678 235	1 259.67	1 678 510	0.04	1 645 650	289.44	1 654 412	-	-	112.85	**1 680 182**	901.26	1 680 179
	ocean	4.88	**7 248 581**	1 018.08	7 210 228	11 142.43	7 075 329	0.07	6 803 672	840.54	7 071 829	-	-	5.20	**7 248 581**	5.01	**7 248 581**
	pwt	-	-	2 754.08	1 174 472	761.52	1 175 437	0.03	1 153 600	396.97	1 149 919	-	-	3 846.33	**1 178 734**	5 131.59	1 178 583
	rotor	-	-	5 559.60	2 643 669	6 503.27	2 650 018	0.24	2 591 456	1 071.61	2 569 961	-	-	29 249.04	**2 662 247**	24 255.33	2 661 514
	sphere	-	-	51.71	615 017	257.42	615 958	0.02	608 401	171.28	613 128	0.57	**617 816**	1.96	**617 816**	2.25	**617 816**
mesh	blob	0.14	**855 547**	20 854.69	854 991	260.10	854 803	0.01	854 484	190.14	853 616	0.02	**855 547**	0.04	**855 547**	0.03	**855 547**
	buddha	51.87	**57 555 880**	1 439.13	57 497 819	36 000.07	57 258 790	0.47	57 508 556	29 022.48	55 973 791	1.57	**57 555 880**	4.83	**57 555 880**	2 417.30	57 555 864
	bunny	0.48	**3 686 960**	1 582.74	3 683 941	1 927.84	3 680 587	0.03	3 682 356	817.06	3 659 471	0.09	**3 686 960**	0.17	**3 686 960**	0.15	**3 686 960**
	cow	0.04	**269 543**	6 571.10	269 402	52.05	269 336	<0.01	269 304	15.62	269 220	0.01	**269 543**	0.01	**269 543**	0.01	**269 543**
	dragon	2.90	**7 956 530**	555.32	7 949 744	9 026.60	7 947 535	0.04	7 950 526	1 922.25	7 846 579	0.17	**7 956 530**	0.31	**7 956 530**	0.33	**7 956 530**
	dragonsub	4.76	**32 213 898**	619.21	32 182 406	36 000.07	32 148 544	0.24	32 163 872	16 990.27	31 294 908	0.93	**32 213 898**	2.10	**32 213 898**	1.93	**32 213 898**
	ecat	9.18	**36 650 298**	431.79	36 616 204	36 000.05	36 562 652	0.50	36 606 394	16 759.95	35 603 546	1.92	**36 650 298**	2.83	**36 650 298**	3.01	**36 650 298**
	face	0.16	**1 219 418**	480.83	1 218 552	403.52	1 218 433	0.01	1 218 515	199.42	1 213 445	0.02	**1 219 418**	0.04	**1 219 418**	0.03	**1 219 418**
	fandisk	0.04	**463 288**	146.07	462 910	114.01	462 794	<0.01	462 765	67.53	462 463	0.01	**463 288**	0.02	**463 288**	0.01	**463 288**
	feline	0.34	**2 207 219**	450.92	2 205 443	794.54	2 204 454	0.02	2 204 947	474.86	2 189 581	0.05	**2 207 219**	0.10	**2 207 219**	0.09	**2 207 219**
	gameguy	0.10	**2 325 878**	27.44	2 324 222	789.78	2 322 814	0.02	2 324 088	431.10	2 306 182	0.04	**2 325 878**	0.06	**2 325 878**	0.05	**2 325 878**
	gargoyle	0.22	**1 059 559**	1 748.69	1 058 724	346.19	1 058 536	0.01	1 058 656	283.58	1 055 563	0.02	**1 059 559**	0.04	**1 059 559**	0.03	**1 059 559**
	turtle	3.98	**14 263 005**	3 342.67	14 249 692	20 430.40	14 245 854	0.09	14 247 883	5 991.49	13 962 425	0.35	**14 263 005**	0.74	**14 263 005**	0.73	**14 263 005**
	venus	0.02	**305 749**	298.23	305 571	59.48	305 556	<0.01	305 182	26.18	305 457	0.01	**305 749**	0.01	**305 749**	0.01	**305 749**
osm	alabama-2	0.35	**174 309**	867.30	174 124	41.71	**174 309**	0.01	172 797	0.89	172 871	0.01	**174 309**	0.03	**174 309**	0.03	**174 309**
	alabama-3	36 000.00	185 707	32 238.96	181 548	321.23	**185 744**	0.40	182 667	39.57	180 461	1.53	**185 744**	32.67	**185 744**	28.53	**185 744**
	d.-of-c.-1	-	-	11 408.24	196 341	41.16	**196 475**	0.01	193 364	19.42	196 044	0.47	**196 475**	1.50	**196 475**	1.26	**196 475**
	d.-of-c.-2	-	-	11 021.45	204 843	985.93	209 131	2.25	198 327	178.71	202 111	-	-	184.04	**209 132**	107.25	**209 132**
	d.-of-c.-3	36 000.20	207 787	6 233.48	212 734	9 278.60	227 634	351.94	210 461	681.32	212 583	-	-	4 283.28	**227 682**	3 231.32	227 681
	florida-2	0.01	**230 595**	294.75	**230 595**	42.92	**230 595**	<0.01	230 008	4.65	229 496	<0.01	**230 595**	0.01	**230 595**	0.01	**230 595**
	florida-3	1 724.45	**237 333**	3 533.44	232 105	216.24	**237 333**	0.13	234 218	25.20	234 940	1.33	**237 333**	11.20	**237 333**	9.57	**237 333**
	georgia-3	1 772.88	**222 652**	269.99	219 891	101.22	**222 652**	0.09	218 573	15.81	218 610	0.77	**222 652**	7.55	**222 652**	6.13	**222 652**
	greenland-3	36 000.00	13 894	261.66	12 561	1 226.78	**14 011**	32.75	12 505	122.36	12 494	-	-	521.57	**14 011**	185.01	**14 011**
	hawaii-2	8.20	**125 284**	1 311.40	124 466	203.81	**125 284**	0.14	123 173	42.44	123 757	0.07	**125 284**	0.46	**125 284**	0.36	**125 284**
	hawaii-3	36 003.35	132 806	3 568.95	132 562	17 330.34	141 045	1 577.65	134 703	-	-	-	-	6 516.88	141 056	5 124.22	**141 058**
	idaho-3	36 000.00	77 122	180.84	75 831	1 549.78	**77 145**	52.16	75 527	42.76	75 901	-	-	2 142.01	**77 145**	410.29	**77 145**

Continued on next page

Table B.2 – *Continued from previous page*

		KaMIS BnR		GNN-VC		HILS		HtWIS		NuMWVC		Struction		M^2WIS+S		M^2WIS	
		t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
osm	kansas-3	36 001.02	87 963	23 741.49	87 688	759.69	**87 976**	2.43	87 424	137.89	87 497	16.79	**87 976**	65.05	**87 976**	104.24	**87 976**
	kentucky-2	63.34	**97 397**	3 595.24	96 550	319.15	**97 397**	0.74	97 362	141.39	96 838	0.20	**97 397**	0.83	**97 397**	0.68	**97 397**
	kentucky-3	36 001.57	100 311	2 502.83	98 310	28 313.04	100 508	3 508.49	97 906	-	-	-	-	13 121.68	100 508	8 865.42	**100 510**
	louisiana-3	22.52	**60 024**	7 238.25	59 473	62.47	**60 024**	0.02	59 040	7.55	58 854	0.05	**60 024**	0.60	**60 024**	0.47	**60 024**
	maryland-3	9.48	**45 496**	628.67	44 988	95.12	**45 496**	0.04	44 539	12.53	45 081	0.10	**45 496**	0.73	**45 496**	0.68	**45 496**
	massachusetts-2	0.37	**140 095**	1 186.63	140 051	51.59	**140 095**	0.02	139 799	6.05	139 697	0.04	**140 095**	0.09	**140 095**	0.09	**140 095**
	massachusetts-3	-	-	4 622.32	144 140	355.93	**145 866**	1.77	144 381	79.71	143 984	-	-	77.14	**145 866**	104.87	**145 866**
	mexico-3	921.72	**97 663**	2 969.21	95 902	90.92	**97 663**	0.05	96 700	1.38	95 999	0.86	**97 663**	14.73	**97 663**	14.07	**97 663**
	new-hampshire-3	14.46	**116 060**	40.64	115 734	51.78	**116 060**	0.01	115 161	2.76	112 771	0.04	**116 060**	1.69	**116 060**	1.40	**116 060**
	north-carolina-3	36 000.05	49 563	1 658.84	48 309	205.78	**49 720**	0.42	49 253	51.48	48 306	37.25	**49 720**	142.20	**49 720**	117.07	**49 720**
	oregon-2	0.03	**165 047**	91.84	164 144	74.19	**165 047**	0.03	164 786	7.18	164 006	0.01	**165 047**	0.02	**165 047**	0.01	**165 047**
	oregon-3	36 001.90	**175 078**	1 735.05	170 748	1 120.13	**175 078**	27.87	172 813	435.08	171 321	-	-	292.46	**175 078**	266.55	**175 078**
	pennsylvania-3	107.51	**143 870**	111.17	143 406	60.62	**143 870**	0.02	142 472	2.39	140 352	0.06	**143 870**	2.30	**143 870**	2.01	**143 870**
	rhode-i.-2	-	-	18 480.55	182 234	166.18	**184 596**	0.36	179 366	48.37	183 127	0.38	**184 596**	1.39	**184 596**	1.46	**184 596**
	rhode-i.-3	36 000.20	196 173	21 044.39	192 449	3 899.85	201 751	280.42	190 341	162.00	189 587	-	-	3 319.95	**201 769**	1 964.28	201 768
	utah-3	239.50	**98 847**	14 144.18	97 033	72.21	**98 847**	0.04	97 754	9.95	96 251	0.08	**98 847**	2.34	**98 847**	1.94	**98 847**
	vermont-3	36 000.45	63 305	25 708.04	59 773	842.04	63 304	3.81	60 518	167.97	60 454	-	-	1 855.99	**63 312**	142.43	**63 312**
	virginia-2	0.60	**295 867**	1 341.69	294 149	87.88	**295 867**	0.02	290 535	3.56	295 171	0.02	**295 867**	0.12	**295 867**	0.12	**295 867**
	virginia-3	36 000.80	307 981	35.87	296 735	482.02	**308 305**	1.14	300 335	82.45	298 038	-	-	1 021.68	**308 305**	172.77	**308 305**
	washington-2	5.67	**305 619**	2 794.95	301 157	199.38	**305 619**	0.06	300 195	18.68	303 632	0.05	**305 619**	0.23	**305 619**	0.19	**305 619**
	washington-3	-	-	9.58	295 610	1 946.06	**314 288**	11.74	305 019	168.96	297 908	-	-	2 731.64	**314 288**	327.63	**314 288**
	west-virginia-3	36 000.27	**47 927**	9 801.99	46 791	147.10	**47 927**	0.25	46 344	0.37	45 499	2.35	**47 927**	64.82	**47 927**	104.41	**47 927**
snap	as-skitter	-	-	1 166.25	124 155 737	36 000.25	123 994 141	1.04	124 141 373	34 265.33	123 594 776	-	-	3 422.79	**124 157 729**	621.62	**124 157 729**
	ca-AstroPh	0.02	**797 510**	1.02	**797 510**	924.29	797 508	0.02	797 363	315.84	796 125	0.02	**797 510**	0.04	**797 510**	0.04	**797 510**
	ca-CondMat	0.02	**1 147 950**	0.34	**1 147 950**	1 015.80	1 147 947	0.01	**1 147 950**	430.29	1 146 066	0.01	**1 147 950**	0.02	**1 147 950**	0.02	**1 147 950**
	ca-GrQc	<0.01	**287 919**	0.21	**287 919**	144.49	**287 919**	<0.01	287 850	59.07	287 638	<0.01	**287 919**	<0.01	**287 919**	<0.01	**287 919**
	ca-HepPh	0.01	**581 039**	18.03	**581 039**	589.17	**581 039**	0.01	580 864	212.89	580 439	0.01	**581 039**	0.02	**581 039**	0.02	**581 039**
	ca-HepTh	0.01	**562 004**	0.17	**562 004**	279.70	**562 004**	<0.01	561 736	150.18	561 331	<0.01	**562 004**	0.01	**562 004**	0.01	**562 004**
	com-amazon	0.48	**19 271 031**	2.64	**19 271 031**	33 178.00	19 270 284	0.14	19 270 078	10 316.17	19 083 928	0.36	**19 271 031**	0.56	**19 271 031**	0.53	**19 271 031**
	com-youtube	0.76	**90 295 294**	4.96	**90 295 294**	36 000.10	90 289 947	0.34	90 295 285	11 288.89	89 986 918	0.69	**90 295 294**	0.90	**90 295 294**	0.97	**90 295 294**
	email-Enron	0.02	**2 461 254**	0.40	**2 461 254**	1 432.11	2 461 242	0.01	**2 461 254**	587.52	2 459 380	0.02	**2 461 254**	0.03	**2 461 254**	0.03	**2 461 254**
	email-EuAll	0.07	**25 286 322**	0.73	**25 286 322**	17 439.93	**25 286 322**	0.03	25 265 214	2 089.93	25 285 560	0.04	**25 286 322**	0.11	**25 286 322**	0.11	**25 286 322**
	loc-gow.	-	-	985.50	12 276 922	17 018.50	12 275 375	0.08	12 276 781	3 886.18	12 203 398	1.32	**12 276 929**	3.89	**12 276 929**	4.71	**12 276 929**
	p2p-G.04	0.01	**679 111**	0.18	**679 111**	250.11	679 110	<0.01	679 085	123.87	678 920	0.01	**679 111**	0.01	**679 111**	0.01	**679 111**
	p2p-G.05	0.01	**554 943**	0.18	**554 943**	192.45	**554 943**	<0.01	**554 943**	76.17	554 757	<0.01	**554 943**	0.01	**554 943**	0.01	**554 943**
	p2p-G.06	0.01	**548 612**	0.20	**548 612**	183.91	**548 612**	<0.01	**548 612**	70.78	548 449	<0.01	**548 612**	0.01	**548 612**	0.01	**548 612**
	p2p-G.08	<0.01	**434 577**	0.13	**434 577**	109.72	**434 577**	<0.01	**434 577**	14.93	434 513	<0.01	**434 577**	<0.01	**434 577**	<0.01	**434 577**
	p2p-G.09	<0.01	**568 439**	0.16	**568 439**	152.25	**568 439**	<0.01	**568 439**	20.44	568 351	<0.01	**568 439**	0.01	**568 439**	0.01	**568 439**

Continued on next page

Table B.2 – *Continued from previous page*

		KaMIS BnR		GNN-VC		HILS		HtWIS		NuMWVC		Struction		m^2wis+s		m^2wis	
		t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
snap	p2p-G.24	0.01	**1 984 567**	0.15	**1 984 567**	569.39	**1 984 567**	0.01	**1 984 567**	339.70	1 984 248	0.01	**1 984 567**	0.02	**1 984 567**	0.01	**1 984 567**
	p2p-G.25	0.01	**1 701 967**	0.13	**1 701 967**	467.31	**1 701 967**	0.01	**1 701 967**	129.05	1 701 819	0.01	**1 701 967**	0.01	**1 701 967**	0.01	**1 701 967**
	p2p-G.30	0.02	**2 787 907**	0.18	**2 787 907**	810.63	**2 787 907**	0.01	2 787 902	285.78	2 787 660	0.01	**2 787 907**	0.02	**2 787 907**	0.02	**2 787 907**
	p2p-G.31	0.03	**4 776 986**	0.28	**4 776 986**	1 795.27	4 776 969	0.01	4 776 925	789.92	4 776 386	0.02	**4 776 986**	0.04	**4 776 986**	0.04	**4 776 986**
	roadNet-CA	279.50	**111 360 828**	3 911.60	111 337 979	36 000.15	109 991 788	0.61	111 325 524	35 932.85	108 909 808	1.54	**111 360 828**	5.02	**111 360 828**	4.67	**111 360 828**
	roadNet-PA	16.44	**61 731 589**	7 394.67	61 719 900	36 000.07	61 549 659	0.33	61 710 606	23 724.64	60 461 602	0.85	**61 731 589**	2.19	**61 731 589**	2.38	**61 731 589**
	roadNet-TX	15.76	**78 599 946**	3 846.17	78 586 678	36 000.10	78 164 327	0.42	78 575 460	34 979.99	76 992 375	1.05	**78 599 946**	2.81	**78 599 946**	2.90	**78 599 946**
	soc-Epinions1	0.05	**5 690 970**	0.59	**5 690 970**	2 813.40	5 690 859	0.02	**5 690 970**	1 043.52	5 686 352	0.05	**5 690 970**	0.07	**5 690 970**	0.06	**5 690 970**
	soc-LiveJ.	36 002.35	284 008 877	16 424.34	284 026 908	36 000.67	281 688 778	12.20	283 922 214	-	-	-	-	779.88	**284 036 239**	1 738.22	**284 036 239**
	soc-p.-rel.	36 059.01	82 778 214	23 661.02	83 924 150	36 000.42	83 696 885	55.41	83 920 370	31 310.08	83 187 970	634.61	79 620 979	9 691.59	83 939 404	9 038.95	**83 944 926**
	soc-S.0811	0.06	**5 660 899**	0.63	**5 660 899**	4 106.47	5 660 734	0.02	**5 660 899**	1 091.76	5 655 800	0.06	**5 660 899**	0.08	**5 660 899**	0.08	**5 660 899**
	soc-S.0902	0.07	**5 971 849**	0.62	**5 971 849**	4 260.67	5 971 574	0.02	5 971 821	1 082.17	5 965 971	0.07	**5 971 849**	0.08	**5 971 849**	0.13	**5 971 849**
	web-BerkStan	36 000.12	43 891 206	472.62	43 904 999	36 000.10	43 888 267	9.94	43 889 843	12 844.27	43 473 969	6.52	**43 907 482**	8.75	**43 907 482**	9.32	**43 907 482**
	web-Google	2.33	**56 326 504**	45.40	56 326 476	36 000.15	56 319 614	0.65	56 323 382	9 398.74	56 023 547	1.52	**56 326 504**	2.40	**56 326 504**	2.47	**56 326 504**
	web-NotreD.	496.64	**26 016 941**	1 596.06	26 016 642	27 389.91	26 014 810	0.12	26 013 830	7 659.14	25 928 547	1.36	**26 016 941**	0.99	**26 016 941**	1.09	**26 016 941**
	web-Stanford	-	-	1 746.45	17 792 664	35 324.07	17 789 989	0.50	17 789 430	8 338.97	17 605 207	1.36	**17 792 930**	1.84	**17 792 930**	1.93	**17 792 930**
	wiki-Talk	1.08	**235 837 346**	16.94	**235 837 346**	36 000.12	235 837 287	0.41	**235 837 346**	34 505.67	235 822 182	0.95	**235 837 346**	1.39	**235 837 346**	1.53	**235 837 346**
	wiki-Vote	0.01	**500 079**	5.21	**500 079**	201.84	**500 079**	0.01	499 740	24.84	499 993	0.01	**500 079**	0.02	**500 079**	0.02	**500 079**
ssmc	ca2010	-	-	4 702.04	16 817 567	36 000.07	16 828 547	0.47	16 792 827	21 336.94	16 053 480	6.18	**16 869 550**	28.32	**16 869 550**	10 985.99	16 869 127
	fl2010	36 000.10	8 638 961	883.37	8 721 942	36 000.05	8 732 113	0.44	8 719 272	14 712.40	8 248 077	2.02	**8 743 506**	13.57	**8 743 506**	7 492.86	8 743 474
	ga2010	36 000.10	4 644 324	18 636.56	4 637 542	29 522.41	4 642 807	0.16	4 639 891	7 492.19	4 437 269	0.62	**4 644 417**	1.64	**4 644 417**	1.85	**4 644 417**
	il2010	36 000.10	5 852 296	4 106.19	5 981 072	36 000.00	5 983 871	0.31	5 963 974	13 341.29	5 776 584	2.33	**5 998 539**	22.98	**5 998 539**	13 136.79	5 998 288
	nh2010	36 000.00	581 637	2 328.87	586 642	2 163.80	588 797	0.03	587 059	758.10	568 188	0.11	**588 996**	0.47	**588 996**	0.47	**588 996**
	ri2010	36 000.00	447 427	717.01	457 288	782.49	458 489	0.02	457 108	371.27	454 423	0.09	**459 275**	0.62	**459 275**	0.49	**459 275**

Table B.3: Average solution weight ω and time t in seconds required to compute it on reduced instances using Reduction List 4.1. Only instances with $|V| \geq 1\,000$ are listed, the summary includes all. Bold numbers indicate the **best** solution among all algorithms.

	Reduced	KaMIS BnR		GNN-VC		HILS		HtWIS		NuMWVC		Struction		M^2WIS+S		M^2WIS	
	Instance	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
el.	body	-	-	717.66	223 946	74.81	224 550	0.01	222 320	45.36	224 168	-	-	60.30	**224 744**	100.23	**224 744**
el.	pwt	-	-	13 585.64	1 034 567	655.57	1 034 418	0.07	1 019 262	174.21	1 014 280	-	-	1 391.40	**1 038 139**	4 767.83	1 038 046
fin.	rotor	36 000.10	2 483 283	8 029.89	2 632 254	3 078.32	2 639 734	0.67	2 580 529	1 034.92	2 553 422	-	-	29 550.69	2 651 439	23 323.54	**2 651 735**
fin.	sphere	36 000.00	461 060	16.72	472 884	179.43	474 290	0.01	467 030	161.77	472 508	0.24	**475 344**	1.20	**475 344**	1.28	**475 344**
mesh	buddha	0.01	**23 026**	0.36	**23 026**	6.85	**23 026**	<0.01	22 726	0.86	22 937	<0.01	**23 026**	0.03	**23 026**	0.02	**23 026**
mesh	ecat	0.02	**31 254**	0.24	**31 254**	8.17	**31 254**	<0.01	30 789	1.12	31 121	<0.01	**31 254**	0.03	**31 254**	0.03	**31 254**
osm	alabama-3	36 000.00	35 944	1 780.24	34 715	107.88	**35 968**	0.23	34 667	4.58	33 440	1.12	**35 968**	30.52	**35 968**	29.27	**35 968**
osm	california-3	821.86	**13 689**	15 006.79	13 253	37.87	**13 689**	0.03	13 058	3.31	12 534	0.22	**13 689**	20.57	**13 689**	19.89	**13 689**
osm	canada-3	55.19	**17 591**	6 069.32	17 487	19.42	**17 591**	0.01	16 063	0.50	16 892	0.13	**17 591**	2.43	**17 591**	2.30	**17 591**
osm	colorado-3	0.14	**9 580**	0.17	**9 580**	7.11	**9 580**	<0.01	9 274	0.14	9 221	0.01	**9 580**	0.05	**9 580**	0.04	**9 580**
osm	d.o.c.-1	-	-	31.71	**55 063**	12.13	**55 063**	0.01	52 450	1.41	54 803	0.43	**55 063**	1.63	**55 063**	1.53	**55 063**
osm	d.o.c.-2	-	-	9 513.25	93 988	403.56	96 318	2.02	88 932	41.91	94 839	-	-	182.18	**96 322**	106.23	**96 322**
osm	d.o.c.-3	36 000.10	119 502	11 779.45	124 774	4 626.91	138 744	283.13	124 431	254.73	127 833	-	-	3 810.44	138 795	2 788.96	**138 797**
osm	florida-3	1 714.07	**25 992**	4 482.86	25 700	65.56	**25 992**	0.11	23 471	5.62	25 375	1.34	**25 992**	8.20	**25 992**	7.80	**25 992**
osm	georgia-3	763.74	**33 714**	14 126.14	31 934	54.48	**33 714**	0.08	30 570	5.13	32 412	0.76	**33 714**	6.78	**33 714**	6.63	**33 714**
osm	greenland-2	16.22	**3 537**	310.67	**3 537**	19.18	**3 537**	0.01	3 142	0.30	3 390	0.05	**3 537**	1.13	**3 537**	0.98	**3 537**
osm	greenland-3	36 000.00	11 419	495.96	10 304	836.96	**11 581**	43.14	9 905	33.53	10 508	-	-	478.88	**11 581**	161.78	**11 581**
osm	hawaii-2	1.42	**11 617**	203.72	11 595	16.62	**11 617**	0.01	11 410	0.81	11 452	0.01	**11 617**	0.22	**11 617**	0.22	**11 617**
osm	hawaii-3	36 001.92	50 612	460.52	52 089	13 683.70	58 812	1 747.34	52 595	424.67	52 169	-	-	4 965.81	58 858	3 845.37	**58 861**
osm	idaho-3	36 000.80	9 221	2 850.49	7 891	1 128.20	**9 224**	55.91	8 139	19.64	8 488	-	-	2 103.78	**9 224**	375.21	**9 224**
osm	kansas-3	36 001.72	5 681	8 074.38	5 512	410.05	**5 694**	1.80	5 173	19.38	5 459	12.08	**5 694**	62.66	**5 694**	102.59	**5 694**
osm	kentucky-2	17.08	**7 019**	8 979.49	6 981	30.49	**7 019**	0.02	6 979	0.26	6 816	0.04	**7 019**	0.52	**7 019**	0.43	**7 019**
osm	kentucky-3	36 001.50	26 198	6 103.84	24 580	23 202.28	26 397	4 335.68	24 614	-	-	-	-	9 536.24	26 397	5 679.08	**26 398**
osm	louisiana-3	2.52	**9 326**	5.30	9 094	16.06	**9 326**	<0.01	8 977	0.74	9 151	0.02	**9 326**	0.25	**9 326**	0.26	**9 326**
osm	maryland-3	1.83	**7 105**	466.86	7 070	13.94	**7 105**	<0.01	6 561	0.45	6 153	0.09	**7 105**	0.42	**7 105**	0.40	**7 105**
osm	massachusetts-2	0.32	**7 938**	14.41	**7 938**	9.80	**7 938**	<0.01	7 650	0.12	7 797	0.03	**7 938**	0.08	**7 938**	0.08	**7 938**
osm	massachusetts-3	36 000.45	14 610	13 976.78	13 673	247.66	**14 757**	1.79	13 285	29.15	14 085	-	-	74.22	**14 757**	102.60	**14 757**
osm	mexico-3	551.33	**16 137**	12 688.47	14 623	48.35	**16 137**	0.04	15 598	2.79	14 745	0.28	**16 137**	8.90	**16 137**	8.12	**16 137**
osm	minnesota-3	8.04	**4 343**	20.69	**4 343**	12.04	**4 343**	<0.01	4 104	0.14	4 281	0.02	**4 343**	0.50	**4 343**	0.50	**4 343**
osm	montana-3	874.67	**5 116**	10 322.49	4 991	66.85	**5 116**	0.10	5 094	7.32	5 015	0.19	**5 116**	3.85	**5 116**	3.72	**5 116**
osm	new-hampshire-3	6.69	**11 473**	13.97	**11 473**	14.40	**11 473**	<0.01	11 104	0.63	10 597	0.04	**11 473**	1.03	**11 473**	1.12	**11 473**
osm	new-york-2	0.22	**4 540**	3.77	**4 540**	8.43	**4 540**	<0.01	**4 540**	<0.01	4 163	0.01	**4 540**	0.03	**4 540**	0.04	**4 540**
osm	new-york-3	9 493.18	**5 897**	9 405.58	5 832	76.89	**5 897**	0.15	4 922	2.78	5 552	0.29	**5 897**	60.80	**5 897**	100.98	**5 897**

Continued on next page

Table B.3 – *Continued from previous page*

		KaMIS BnR		GNN-VC		HILS		HtWIS		NuMWVC		Struction		m^2wis+s		m^2wis	
		t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω	t	ω
osm	north-carolina-3	36 000.07	11 073	359.98	10 044	166.57	**11 191**	0.43	10 653	11.23	10 582	67.23	**11 191**	141.71	**11 191**	116.51	**11 191**
	ohio-3	7.08	**5 213**	157.84	5 170	13.05	**5 213**	<0.01	4 794	0.23	4 050	0.01	**5 213**	0.61	**5 213**	0.61	**5 213**
	oregon-3	36 002.17	23 422	17 211.78	21 920	804.84	**23 427**	28.97	20 386	98.83	22 395	-	-	249.18	**23 427**	268.10	**23 427**
	pennsylvania-3	11.45	**14 766**	27.10	**14 766**	15.90	**14 766**	<0.01	13 921	0.14	13 745	0.03	**14 766**	1.31	**14 766**	1.25	**14 766**
	puerto-rico-3	194.64	**3 544**	905.34	**3 544**	20.06	**3 544**	0.01	3 527	0.13	3 176	0.05	**3 544**	5.69	**3 544**	5.19	**3 544**
	rhode-i.-2	4 746.44	**42 587**	7 789.73	42 526	62.79	**42 587**	0.17	39 486	7.04	41 619	0.28	**42 587**	0.98	**42 587**	1.03	**42 587**
	rhode-i.-3	36 000.20	70 873	18 572.74	66 854	3 225.06	76 431	394.81	65 468	23.92	68 350	-	-	3 229.68	**76 458**	1 348.52	**76 458**
	utah-3	200.33	**16 090**	980.62	15 018	41.05	**16 090**	0.03	15 186	3.58	15 167	0.07	**16 090**	1.77	**16 090**	1.64	**16 090**
	vermont-2	17.92	**4 308**	5 549.13	4 290	25.14	**4 308**	0.01	3 343	0.94	3 862	0.02	**4 308**	1.04	**4 308**	0.99	**4 308**
	vermont-3	36 000.20	22 804	16 180.19	21 215	457.32	**22 813**	3.66	20 011	51.72	21 726	-	-	1 849.27	**22 813**	133.53	**22 813**
	virginia-2	0.26	**23 680**	2.80	**23 680**	9.70	**23 680**	<0.01	23 275	0.13	23 081	0.01	**23 680**	0.06	**23 680**	0.05	**23 680**
	virginia-3	36 000.77	90 854	291.78	80 459	285.12	**91 046**	1.37	83 975	21.52	86 081	-	-	1 014.80	**91 046**	304.16	**91 046**
	washington-2	4.24	**33 415**	37.83	**33 415**	20.71	**33 415**	0.01	32 185	0.15	32 965	0.02	**33 415**	0.17	**33 415**	0.19	**33 415**
	washington-3	-	-	534.33	98 789	1 592.22	113 514	23.64	104 692	71.86	103 006	-	-	2 101.05	**113 516**	289.58	**113 516**
	west-virginia-3	32 129.35	**16 666**	17 644.66	15 573	135.05	**16 666**	0.32	15 290	2.42	14 996	2.35	**16 666**	64.48	**16 666**	104.57	**16 666**
snap	as-skitter	-	-	21 606.10	135 910	45.35	135 976	0.04	134 500	26.37	135 783	-	-	1 904.18	**135 998**	287.87	**135 998**
	loc-gowalla_e.	294.11	**16 521**	25.68	**16 521**	8.40	**16 521**	<0.01	**16 521**	0.15	16 479	0.64	**16 521**	0.94	**16 521**	0.78	**16 521**
	soc-LiveJ.	-	-	5 669.30	277 920	180.88	**278 532**	0.13	273 672	76.47	277 988	-	-	308.08	**278 532**	145.28	**278 532**
	soc-pokec-rel.	36 038.14	34 227 442	18 389.52	35 219 027	36 000.15	**35 242 470**	81.18	35 205 509	7 035.18	34 828 738	644.98	32 989 873	8 238.05	35 225 724	7 259.99	35 230 752
	web-BerkStan	28.05	**135 172**	15.35	134 942	36.21	135 156	<0.01	133 309	14.97	134 676	0.04	**135 172**	0.17	**135 172**	0.15	**135 172**
	web-Google	0.13	**20 182**	1.69	**20 182**	9.22	**20 182**	<0.01	20 091	0.67	20 171	0.01	**20 182**	0.05	**20 182**	0.04	**20 182**
	web-NotreD.	0.33	**29 145**	22.52	**29 145**	11.67	**29 145**	<0.01	28 640	0.56	29 061	0.07	**29 145**	0.11	**29 145**	0.09	**29 145**
	web-Stanford	4 832.78	**31 695**	793.53	31 668	11.21	**31 695**	<0.01	31 427	2.36	31 617	0.01	**31 695**	0.04	**31 695**	0.05	**31 695**
ssmc	ca2010	-	-	2 873.13	1 979 294	1 470.66	1 980 551	0.16	1 951 444	916.03	1 951 636	1.47	**1 989 209**	5.79	**1 989 209**	8 652.80	1 989 106
	fl2010	36 000.00	649 704	84.54	682 793	932.09	684 214	0.10	673 852	563.46	676 284	0.83	**686 985**	3.82	**686 985**	5 130.54	686 946
	ga2010	159.38	**76 316**	492.05	76 214	59.10	76 297	0.01	75 201	26.71	76 001	0.07	**76 316**	0.34	**76 316**	0.33	**76 316**
	il2010	-	-	3 714.46	998 051	1 290.10	996 126	0.14	981 297	740.62	985 582	0.70	**1 001 624**	4.65	**1 001 624**	8 983.65	1 001 415
	nh2010	2.28	**26 770**	44.27	26 737	17.20	26 768	<0.01	26 477	4.56	26 687	0.01	**26 770**	0.07	**26 770**	0.05	**26 770**
	ri2010	6 477.22	**66 963**	1 008.11	66 672	32.71	66 960	<0.01	65 979	19.66	66 743	0.02	**66 963**	0.15	**66 963**	0.12	**66 963**

	KaMIS BnR	GNN-VC	HILS	HtWIS	NuMWVC	Struction	m^2wis+s	m^2wis
# best	67/93	46/93	75/93	14/93	0/93	75/93	88/93	88/93
gmean ω	-	16 357	16 683	15 964	-	-	16 688	16 688
gmean t	-	35.30	25.01	0.01	-	-	0.60	0.66

B.2 State-of-the-Art Results Comparing CHILS

Table B.4: Average solution weight ω and time t (in seconds) required to compute it for the *non reduced* instances from set one. The **best** solutions among all algorithms are marked bold.

		HtWIS		HILS		m²wis+s		BASELINE		CHILS	
	Instance	ω	t	ω	t	ω	t	ω	t	ω	t
fe	body	1 645 650	0.06	1 678 383	3 471.02	**1 680 182**	2 958.88	**1 680 182**	1 824.22	**1 680 182**	165.03
	ocean	6 803 672	0.12	7 081 151	3 384.46	**7 248 581**	3.38	7 139 076	16.75	7 210 830	2 863.50
	pwt	1 153 600	0.04	1 175 710	3 580.37	1 176 685	3 601.03	1 178 479	49.24	**1 178 708**	537.13
	rotor	2 591 456	0.31	2 651 471	3 255.65	2 596 217	3 601.17	2 663 757	880.70	**2 665 805**	3 179.83
	sphere	608 401	0.02	617 139	3 166.86	**617 816**	1.95	617 738	6.05	**617 816**	125.55
mesh	blob	854 484	< 0.01	855 004	252.71	**855 547**	0.04	855 544	0.26	**855 547**	45.37
	buddha	57 508 556	0.52	56 086 272	3 600.05	**57 555 880**	4.26	57 554 770	3 380.94	57 555 764	2 571.52
	bunny	3 682 356	0.04	3 681 110	3 415.81	**3 686 960**	0.14	3 686 896	15.13	**3 686 960**	167.07
	dragonsub	32 163 872	0.28	31 833 085	3 599.67	**32 213 898**	1.62	32 212 574	3 409.68	32 213 769	2 857.15
	dragon	7 950 526	0.07	7 946 347	3 485.87	**7 956 530**	0.27	7 956 496	32.35	7 956 526	160.64
	ecat	36 606 394	0.48	36 122 506	3 600.03	**36 650 298**	2.60	36 648 982	3 279.35	36 650 039	2 155.87
	fandisk	462 765	< 0.01	463 049	3 212.81	**463 288**	0.02	463 280	2.37	**463 288**	21.82
	feline	2 204 947	0.03	2 204 823	3 055.87	**2 207 219**	0.10	2 207 213	1.60	**2 207 219**	105.20
	gameguy	2 324 088	0.02	2 323 182	3 553.14	**2 325 878**	0.06	2 325 802	5.25	**2 325 878**	42.04
	gargoyle	1 058 656	< 0.01	1 059 102	2 337.95	**1 059 559**	0.05	**1 059 559**	1.35	**1 059 559**	1.35
	turtle	14 247 883	0.14	14 238 437	3 597.64	**14 263 005**	0.58	14 262 790	2 753.32	14 262 993	1 665.35

Continued on next page

Table B.4 – *Continued from previous page*

		HtWIS		HILS		m²wis+s		BASELINE		CHILS	
	Instance	ω	t	ω	t	ω	t	ω	t	ω	t
osm	alabama-2	172 797	0.01	**174 309**	0.11	**174 309**	0.03	**174 309**	0.10	**174 309**	0.10
	alabama-3	182 667	0.52	**185 744**	0.30	**185 744**	28.95	**185 744**	14.44	**185 744**	16.45
	california-2	45 377	< 0.01	**47 153**	< 0.01	**47 153**	0.03	**47 153**	< 0.01	**47 153**	< 0.01
	california-3	48 356	0.04	**49 365**	0.08	**49 365**	19.04	**49 365**	0.05	**49 365**	0.05
	canada-3	84 503	0.02	**86 018**	0.01	**86 018**	2.07	**86 018**	< 0.01	**86 018**	< 0.01
	colorado-3	54 435	< 0.01	**54 741**	0.03	**54 741**	0.07	**54 741**	0.02	**54 741**	0.02
	d.-of-c.-1	193 364	0.02	**196 475**	1.05	**196 475**	1.06	**196 475**	0.07	**196 475**	0.04
	d.-of-c.-2	198 327	2.40	**209 132**	476.45	**209 132**	1 643.20	**209 132**	64.28	**209 132**	160.18
	d.-of-c.-3	210 461	330.18	227 652	939.72	227 681	1 890.32	**227 683**	948.94	**227 683**	258.43
	florida-3	234 218	0.20	**237 333**	1.94	**237 333**	6.91	**237 333**	1.14	**237 333**	1.13
	georgia-3	218 573	0.13	**222 652**	0.04	**222 652**	5.99	**222 652**	0.15	**222 652**	0.15
	greenland-2	10 179	0.06	**10 718**	0.13	**10 718**	0.68	**10 718**	0.24	**10 718**	0.23
	greenland-3	12 505	34.22	**14 012**	3 131.30	14 011	1 360.74	**14 012**	180.88	**14 012**	161.13
	hawaii-2	123 173	0.19	**125 284**	0.12	**125 284**	0.31	**125 284**	1.03	**125 284**	1.04
	hawaii-3	134 703	1 423.03	141 041	3 213.04	141 070	2 611.52	141 074	34.12	**141 095**	160.02
	idaho-3	75 527	52.15	**77 145**	115.88	**77 145**	1 795.12	**77 145**	20.52	**77 145**	12.02
	kansas-3	87 424	3.25	**87 976**	2.26	**87 976**	61.39	**87 976**	10.76	**87 976**	36.08
	kentucky-2	97 362	0.93	**97 397**	0.45	**97 397**	0.46	**97 397**	4.19	**97 397**	4.25
	kentucky-3	97 906	3 077.14	100 510	1 292.90	99 918	3 684.53	100 507	35.39	**100 511**	160.03
	louisiana-3	59 040	0.03	**60 024**	0.02	**60 024**	0.44	**60 024**	0.03	**60 024**	0.03
	maine-3	26 231	< 0.01	**26 734**	< 0.01	**26 734**	< 0.01	**26 734**	< 0.01	**26 734**	< 0.01
	maryland-3	44 539	0.07	**45 496**	0.04	**45 496**	0.54	**45 496**	0.03	**45 496**	0.03
	massachusetts-2	139 799	0.02	**140 095**	0.05	**140 095**	0.10	**140 095**	0.03	**140 095**	0.03

Continued on next page

Table B.4 – *Continued from previous page*

Instance	HtWIS		HILS		m²wis+s		BASELINE		CHILS	
	ω	t	ω	t	ω	t	ω	t	ω	t
massachusetts-3	144 381	2.22	**145 866**	0.41	**145 866**	1 292.74	**145 866**	11.06	**145 866**	11.58
mexico-2	94 820	< 0.01	**94 834**	< 0.01	**94 834**	0.01	**94 834**	< 0.01	**94 834**	< 0.01
mexico-3	96 700	0.07	**97 663**	0.02	**97 663**	8.94	**97 663**	0.10	**97 663**	0.10
minnesota-3	32 318	0.04	**32 787**	< 0.01	**32 787**	0.46	**32 787**	< 0.01	**32 787**	< 0.01
montana-3	59 521	0.18	**59 822**	0.10	**59 822**	3.37	59 753	0.02	59 753	0.02
nevada-3	51 269	0.01	**52 036**	< 0.01	**52 036**	0.19	**52 036**	< 0.01	**52 036**	< 0.01
new-hampshire-3	115 161	0.02	**116 060**	0.04	**116 060**	1.06	**116 060**	< 0.01	**116 060**	< 0.01
new-york-2	**14 330**	< 0.01	**14 330**	< 0.01	**14 330**	0.03	**14 330**	< 0.01	**14 330**	< 0.01
new-york-3	15 293	0.19	**16 268**	0.01	**16 268**	60.59	**16 268**	0.11	**16 268**	0.11
north-carolina-3	49 253	0.52	**49 720**	0.05	**49 720**	1 277.21	**49 720**	0.23	**49 720**	0.27
ohio-3	52 199	0.01	**52 634**	< 0.01	**52 634**	0.71	**52 634**	0.02	**52 634**	0.02
oregon-3	172 813	29.73	**175 078**	5.54	**175 078**	1 396.74	**175 078**	64.38	**175 078**	161.51
pennsylvania-3	142 472	0.02	**143 870**	0.02	**143 870**	1.33	**143 870**	< 0.01	**143 870**	0.01
puerto-rico-3	33 556	0.03	**33 590**	< 0.01	**33 590**	3.54	**33 590**	< 0.01	**33 590**	< 0.01
rhode-i.-2	179 366	0.46	**184 596**	0.14	**184 596**	0.86	**184 596**	0.60	**184 596**	0.18
rhode-i.-3	190 341	255.58	201 753	1 190.44	**201 771**	1 886.87	**201 771**	366.29	**201 771**	15.34
tennessee-3	32 232	< 0.01	**32 276**	< 0.01	**32 276**	0.07	**32 276**	< 0.01	**32 276**	< 0.01
utah-2	95 080	< 0.01	**95 087**	< 0.01	**95 087**	< 0.01	**95 087**	< 0.01	**95 087**	< 0.01
utah-3	97 754	0.05	**98 847**	0.06	**98 847**	1.34	**98 847**	0.10	**98 847**	0.10
vermont-2	57 563	0.04	**59 310**	1.91	**59 310**	1.07	59 284	< 0.01	**59 310**	42.52
vermont-3	60 518	4.89	63 304	667.25	**63 312**	1 317.01	63 302	4.38	63 302	4.41
virginia-2	290 535	0.03	**295 867**	0.13	**295 867**	0.11	**295 867**	0.12	**295 867**	0.12
virginia-3	300 335	1.38	**308 305**	1.34	**308 305**	1 297.87	**308 305**	3.63	**308 305**	3.61

Continued on next page

Table B.4 – *Continued from previous page*

	Instance	HtWIS ω	HtWIS t	HILS ω	HILS t	m^{2}wis+s ω	m^{2}wis+s t	BASELINE ω	BASELINE t	CHILS ω	CHILS t
osm	washington-2	300 195	0.10	**305 619**	0.17	**305 619**	0.17	**305 619**	0.12	**305 619**	0.12
osm	washington-3	305 019	14.41	**314 288**	546.60	**314 288**	1 373.62	**314 288**	115.04	**314 288**	161.81
osm	west-virginia-3	46 344	0.32	**47 927**	0.31	**47 927**	62.72	**47 927**	0.74	**47 927**	0.74
snap	as-skitter	124 141 373	1.17	123 207 288	3 597.32	**124 157 729**	2 828.19	124 141 585	3 545.45	124 111 689	3 491.32
snap	com-amazon	19 270 078	0.18	19 266 084	3 590.70	**19 271 031**	0.45	19 270 569	3 571.05	19 270 939	3 556.41
snap	loc-gowalla-edges	12 276 781	0.12	12 275 221	3 372.84	**12 276 929**	4.98	12 275 141	1 714.74	12 275 967	3 598.85
snap	roadNet-CA	111 325 524	0.72	108 370 752	3 600.18	**111 360 828**	4.12	111 355 248	3 496.83	111 358 047	3 579.38
snap	roadNet-PA	61 710 606	0.41	60 504 456	3 599.52	**61 731 589**	2.01	61 730 745	1 264.12	61 731 458	3 347.56
snap	roadNet-TX	78 575 460	0.49	76 796 646	3 600.03	**78 599 946**	2.39	78 598 317	3 121.37	78 599 552	3 398.95
snap	soc-LiveJ.	283 922 214	9.22	279 386 267	< 0.01	**284 036 239**	1 737.39	283 951 703	3 599.46	283 924 551	3 562.85
snap	soc-p.-rel	83 920 370	39.06	82 984 397	3 598.22	81 112 447	3 637.19	83 971 405	3 369.22	**83 982 856**	3 520.91
snap	web-BerkStan	43 889 843	14.87	43 694 574	3 599.51	**43 907 482**	4.29	43 895 988	2 790.96	43 901 482	3 600.08
snap	web-Google	56 323 382	0.74	56 193 137	3 599.88	**56 326 504**	2.16	56 320 795	2 539.95	56 321 677	3 580.05
snap	web-NotreDame	26 013 830	0.20	26 014 630	3 593.50	**26 016 941**	0.99	26 014 581	3 425.73	26 016 396	3 051.54
snap	web-Stanford	17 789 430	0.57	17 779 603	3 595.78	**17 792 930**	1.31	17 792 392	2 784.70	17 792 474	3 491.05
ssmc	ca2010	16 792 827	0.55	16 591 401	3 597.70	16 867 553	3 601.26	16 867 987	2 650.77	**16 869 278**	2 613.13
ssmc	fl2010	8 719 272	0.40	8 694 303	3 599.61	**8 743 506**	183.77	8 742 384	2 941.69	8 743 309	2 832.06
ssmc	ga2010	4 639 891	0.21	4 640 168	3 595.16	**4 644 417**	61.28	4 644 398	579.13	**4 644 417**	650.90
ssmc	il2010	5 963 974	0.39	5 965 735	3 599.95	**5 998 539**	17.93	5 998 017	2 516.42	5 998 520	3 314.47
ssmc	nh2010	587 059	0.04	588 843	2 811.97	**588 996**	0.44	**588 996**	15.57	**588 996**	28.91
ssmc	ri2010	457 108	0.03	458 983	191.53	**459 275**	0.49	459 270	5.23	**459 275**	51.36

Table B.5: Solution weight ω and time t (in seconds) required to compute it for the *reduced* instances from Instance Set 2. The **best** solutions among all algorithms are marked bold.

		HtWIS		HILS		m²wis+s		BASELINE		CHILS	
	Instance	ω	t	ω	t	ω	t	ω	t	ω	t
fe	body	1 680 066	0.60	**1 680 182**	0.61	**1 680 182**	60.63	**1 680 182**	0.69	**1 680 182**	0.68
fe	pwt	1 169 768	2.21	1 178 273	1 946.51	**1 178 735**	3 196.24	**1 178 735**	9.31	**1 178 735**	9.84
fe	rotor	2 599 908	5.41	2 653 478	2 764.03	2 617 855	3 606.23	2 664 500	347.16	**2 666 222**	3 251.99
osm	alabama-3	184 863	1.52	**185 744**	1.46	**185 744**	304.35	**185 744**	1.47	**185 744**	1.46
osm	california-3	48 244	0.23	**49 365**	0.21	**49 365**	37.22	**49 365**	0.24	**49 365**	0.23
osm	d.-of-c.-2	206 445	6.24	**209 132**	5.97	**209 132**	1 284.78	**209 132**	5.90	**209 132**	5.86
osm	d.-of-c.-3	217 052	395.81	227 586	734.53	227 622	3 450.79	227 205	350.73	**227 656**	450.50
osm	florida-3	235 834	0.57	**237 333**	0.53	**237 333**	14.07	**237 333**	0.71	**237 333**	0.69
osm	georgia-3	220 585	0.42	**222 652**	0.39	**222 652**	8.90	**222 652**	0.40	**222 652**	0.40
osm	greenland-3	13 532	41.89	**14 012**	416.04	**14 012**	1 406.56	**14 012**	28.34	**14 012**	28.25
osm	hawaii-3	135 846	1 659.20	140 963	1 068.11	141 076	4 427.01	140 976	1 023.18	**141 095**	1 785.05
osm	idaho-3	76 227	59.61	**77 145**	52.11	**77 145**	1 499.48	**77 145**	185.03	**77 145**	200.46
osm	kansas-3	87 725	4.51	**87 976**	4.05	**87 976**	64.44	**87 976**	5.48	**87 976**	5.48
osm	kentucky-3	99 310	3 421.08	100 475	2 582.95	100 193	5 125.45	**100 511**	2 167.95	**100 511**	2 385.14
osm	massachusetts-3	144 574	4.01	**145 866**	3.09	**145 866**	1 293.42	**145 866**	3.57	**145 866**	3.57
osm	mexico-3	96 866	0.31	**97 663**	0.29	**97 663**	10.77	**97 663**	0.31	**97 663**	0.31
osm	montana-3	59 679	0.32	**59 822**	0.25	**59 822**	8.95	**59 822**	0.30	**59 822**	0.30
osm	new-york-3	15 666	0.49	**16 268**	0.46	**16 268**	62.24	**16 268**	0.44	**16 268**	0.44

Continued on next page

Table B.5 – *Continued from previous page*

		HtWIS		HILS		m²wis+s		BASELINE		CHILS	
	Instance	ω	t	ω	t	ω	t	ω	t	ω	t
osm	north-carolina-3	48 897	1.08	**49 720**	51.58	**49 720**	1 285.56	**49 720**	1.45	**49 720**	1.42
	oregon-3	173 454	35.28	**175 078**	24.02	**175 078**	1 410.40	**175 078**	25.52	**175 078**	25.46
	rhode-i.-2	184 554	0.57	**184 596**	0.52	**184 596**	5.79	**184 596**	0.55	**184 596**	0.55
	rhode-i.-3	197 126	306.81	201 654	184.23	201 767	3 171.02	201 480	167.51	**201 771**	333.07
	vermont-3	61 795	7.01	**63 312**	5.58	**63 312**	1 317.28	**63 312**	7.44	**63 312**	7.42
	virginia-3	306 026	3.05	**308 305**	3.93	**308 305**	1 294.69	**308 305**	5.13	**308 305**	5.11
	washington-3	307 176	22.98	**314 288**	127.80	**314 288**	1 406.80	**314 288**	27.60	**314 288**	130.21
	west-virginia-3	47 074	0.70	**47 927**	0.69	**47 927**	68.22	**47 927**	0.59	**47 927**	0.59
snap	as-skitter	124 156 703	3.46	**124 157 729**	8.63	**124 157 729**	1 300.24	124 157 700	3.45	**124 157 729**	33.49
	loc-gowalla-edges	12 276 850	0.33	**12 276 929**	0.33	**12 276 929**	3.76	**12 276 929**	0.34	**12 276 929**	0.34
	soc-LiveJ.	284 035 711	23.49	**284 036 239**	23.65	**284 036 239**	1 291.80	**284 036 239**	23.55	**284 036 239**	23.55
	soc-p.-rel	83 913 297	669.24	83 762 908	4 243.55	81 480 804	4 311.48	83 986 271	4 175.34	**83 991 718**	3 975.30
	web-NotreDame	**26 016 941**	0.62	**26 016 941**	0.62	**26 016 941**	0.73	**26 016 941**	0.62	**26 016 941**	0.62
ssmc	fl2010	8 742 642	3.62	**8 743 506**	37.95	**8 743 506**	1 479.03	**8 743 506**	3.65	**8 743 506**	3.65

Table B.6: Detailed per instance results. Best solution found out of ten repetitions for each algorithm after all updates have been performed. The **best** result is highlighted in bold font. Additionally, we give the optimum value computed by KaMIS BnR, if found within 48h by the exact solver. Otherwise we mark it with a × in the respective field.

	Instance	Greedy	DegGreedy	DyTwoSwap	DgOracleTwo	DynOneFast	DynOneStrong	KaMIS BnR
Mesh Networks	3elt	1 240	1 349	1 442	78	1 471	**1 475**	×
	4elt	4 048	4 389	4 777	420	4 905	**4 918**	×
	add20	1 033	1 100	**1 130**	958	1 121	**1 130**	1 130
	add32	2 141	2 269	**2 286**	1 984	2 269	**2 286**	2 286
	crack	4 128	4 572	4 602	4 572	4 572	**4 603**	4 603
	cs4	7 708	7 769	8 767	7 536	8 964	**9 138**	×
	cti	6 155	7 552	7 856	1 998	**8 088**	**8 088**	×
	data	637	678	**685**	105	678	683	×
	fe_4elt2	2 904	3 008	3 448	1 166	3 505	**3 562**	×
	fe_body	12 012	12 760	13 465	4 207	13 620	**13 730**	×
	fe_ocean	53 444	70 456	70 514	6 914	**71 664**	71 629	71 716
	fe_pwt	7 806	8 683	9 033	3 135	9 038	**9 219**	×
	fe_sphere	4 230	**5 462**	**5 462**	655	**5 462**	**5 462**	×
	t60k	24 158	29 403	29 621	29 432	29 437	**29 644**	×
	whitaker3	2 515	2 646	3 002	612	3 095	**3 113**	×
	wing	21 089	20 784	23 967	20 868	24 612	**25 144**	×
Road Networks	asia.osm	5 337 245	5 806 366	5 951 596	5 767 404	5 874 844	**5 980 482**	5 998 335
	belgium.osm	645 260	689 483	716 522	693 765	703 848	**721 749**	724 487
	germany.osm	5 210 948	5 569 050	5 780 252	5 575 465	5 687 811	**5 820 972**	5 841 600
	great-britain.osm	3 535 628	3 789 940	3 921 417	3 782 883	3 849 252	**3 946 495**	3 958 619
	italy.osm	2 983 420	3 253 962	3 328 962	3 257 039	3 265 390	**3 342 945**	3 353 582
	luxembourg.osm	51 428	55 476	57 123	55 923	56 167	**57 424**	57 663
	netherlands.osm	994 501	1 042 427	1 102 700	1 050 646	1 084 286	**1 113 242**	1 116 770
	uk	1 937	2 006	2 140	2 063	2 170	**2 192**	2 198
Social Networks	amazon-ratings	1 982 026	1 991 436	**1 992 583**	1 992 071	1 991 669	1 992 582	1 992 590
	citeulike_ui	708 835	709 906	**710 065**	709 841	709 985	**710 065**	710 067
	dnc-temporalGraph	1 773	1 780	**1 781**	1 775	1 780	**1 781**	1 781
	facebook-wosn-wall	23 651	24 498	25 054	24 358	24 945	**25 121**	25 126
	haggle	**234**	**234**	**234**	**234**	**234**	**234**	234
	lastfm_band	**173 087**	173 066	**173 087**	173 086	**173 087**	**173 087**	173 087
	lkml-reply	53 418	53 926	54 053	53 909	54 024	**54 063**	54 065
	sociopatterns-infections	110	114	**118**	101	**118**	**118**	118
	topology	29 224	29 504	**29 565**	29 548	29 504	**29 565**	29 565

B.3 State-of-the-Art Results Comparing dynamicOne

Table B.7: Detailed per instance results (*higher* is better) on weighted instances. Each entry corresponds to the average result for each algorithm after all updates have been performed. The **best** result is highlighted in bold font.

	Instance	Greedy	DegGreedy	DynOneFast	DynOneStrong
Mesh Networks	3elt	80 427	75 635	90 112	**90 169**
	4elt	263 809	245 426	296 244	**296 484**
	add20	66 379	67 344	69 128	**69 357**
	add32	133 653	138 037	140 735	**141 074**
	crack	201 774	235 131	240 666	**241 510**
	cs4	505 079	493 489	545 842	**549 020**
	cti	339 123	350 491	414 558	**416 153**
	data	34 847	35 189	41 696	**43 954**
	fe_4elt2	191 496	179 536	211 390	**214 539**
	fe_body	734 141	703 940	834 557	**838 087**
	fe_ocean	2 910 036	3 054 326	3 584 670	**3 610 588**
	fe_pwt	515 032	472 184	584 843	**620 905**
	fe_sphere	272 751	257 150	308 532	**308 627**
	t60k	1 498 465	1 514 036	1 623 871	**1 626 909**
	whitaker3	164 853	152 057	185 313	**185 694**
	wing	1 388 751	1 360 853	1 499 895	**1 512 689**
Road Networks	asia.osm	337 151 643	337 879 210	350 337 476	**353 127 303**
	belgium.osm	40 625 661	40 687 449	42 226 435	**42 590 971**
	germany.osm	327 151 986	327 763 791	339 946 811	**342 976 042**
	great-britain.osm	220 872 346	221 522 336	229 334 872	**231 482 437**
	italy.osm	188 795 346	189 286 893	196 001 471	**197 612 299**
	luxembourg.osm	3 241 919	3 248 700	3 359 823	**3 393 258**
	netherlands.osm	62 461 275	62 507 042	64 922 456	**65 544 509**
	uk	122 900	122 093	130 282	**130 875**
Social Networks	amazon-ratings	101 520 569	102 066 076	102 124 693	**102 237 718**
	citeulike_ui	35 600 375	35 951 479	35 970 737	**35 979 609**
	dnc-temporalGraph	89 498	91 019	91 369	**91 428**
	facebook-wosn-wall	1 290 379	1 333 313	1 381 791	**1 389 705**
	haggle	10 980	11 800	11 800	**11 839**
	lastfm_band	8 714 534	8 730 188	8 731 008	**8 731 010**
	lkml-reply	2 758 583	2 776 325	2 796 716	**2 799 473**
	sociopatterns-infections	6 018	6 277	7 015	**7 032**
	topology	1 473 077	1 516 556	1 523 308	**1 525 097**

Table B.8: Detailed per instance results on weighted instances. Each entry corresponds to the first seed value result for each algorithm after all updates have been performed. The **best** result is highlighted in bold font. Additionally, we give the optimum value computed by KaMIS BnR, if found within 48h by the exact solver. Otherwise we mark it with a × in the respective field.

	Instance	Greedy	DegGreedy	DynOneFast	DynOneStrong	KaMIS BnR
Mesh Networks	3elt	80 433	76 471	90 825	**90 909**	90 924
	4elt	262 673	244 985	295 890	**296 240**	296 316
	add20	66 623	67 807	69 524	**69 672**	69 672
	add32	132 463	138 179	140 864	**141 182**	141 259
	crack	204 346	236 599	241 800	**242 904**	242 922
	cs4	503 367	494 685	546 236	**549 342**	×
	cti	339 217	349 883	416 277	**420 197**	×
	data	34 914	34 035	41 839	**44 602**	×
	fe_4elt2	192 996	181 627	211 970	**215 739**	215 755
	fe_body	730 297	708 021	836 208	**840 744**	×
	fe_ocean	2 908 069	3 034 363	3 575 131	**3 601 335**	×
	fe_pwt	516 239	472 353	584 247	**645 158**	×
	fe_sphere	274 290	258 583	309 310	**309 338**	×
	t60k	1 494 694	1 508 797	1 622 343	**1 624 919**	1 625 636
	whitaker3	166 549	152 877	186 574	**187 048**	187 150
	wing	1 389 365	1 360 105	1 500 520	**1 513 556**	×
Road Networks	asia.osm	337 172 858	337 881 640	350 340 186	**353 137 274**	353 174 593
	belgium.osm	40 641 709	40 701 734	42 240 786	**42 605 423**	42 610 605
	germany.osm	327 166 065	327 791 692	339 953 614	**342 990 960**	343 041 226
	great-britain.osm	220 893 807	221 517 185	229 324 772	**231 483 948**	231 529 608
	italy.osm	188 808 339	189 295 075	196 007 403	**197 630 568**	197 651 869
	luxembourg.osm	3 229 282	3 235 616	3 351 233	**3 384 719**	3 384 957
	netherlands.osm	62 504 555	62 538 118	64 944 070	**65 568 780**	65 578 800
	uk	122 753	122 449	130 285	**130 899**	130 946
Social Networks	amazon-ratings	101 602 864	102 115 654	102 186 265	**102 287 992**	102 293 304
	citeulike_ui	35 631 253	35 964 477	35 983 929	**35 994 003**	35 994 022
	dnc-temporalGraph	88 238	90 040	90 388	**90 424**	90 424
	facebook-wosn-wall	1 289 402	1 330 707	1 378 783	**1 388 061**	×
	haggle	9 852	**11 647**	**11 647**	**11 647**	11 647
	lastfm_band	8 698 740	8 717 211	**8 719 546**	**8 719 546**	8 719 546
	lkml-reply	2 757 010	2 775 771	2 796 309	**2 799 487**	2 799 682
	sociopatterns-infections	5 926	6 410	7 022	**7 125**	7 125
	topology	1 463 982	1 519 086	1 525 776	**1 527 491**	1 527 512

B.4 Detailed Results on 2-Packing Set

Table B.9: Percentage of remaining vertices ($\tilde{n} = 100 \cdot n(\mathcal{K}^2/n)$) and edges ($\tilde{m} = 100 \cdot m(\mathcal{K}^2/m)$) for different reduction variants after transformation. Bold numbers indicate the **best** results, gray background marks empty kernels.

		TRANSFORM		MAIN		FAST	
	Instances	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$
social small	PGPgiantcompo	100.00	873.91	**0.00**	**0.00**	**0.00**	**0.00**
	adjnoun	100.00	725.18	**0.00**	**0.00**	**0.00**	**0.00**
	as-22july06	100.00	22 941.92	**0.00**	**0.00**	**0.00**	**0.00**
	celegans_metabolic	100.00	2 239.56	**0.00**	**0.00**	**0.00**	**0.00**
	celegansneural	100.00	1 123.00	**30.64**	**98.46**	**30.64**	**98.46**
	chesapeake	100.00	407.65	**0.00**	**0.00**	**0.00**	**0.00**
	cond-mat	100.00	678.06	**0.00**	**0.00**	**0.00**	**0.00**
	dolphins	100.00	381.76	**0.00**	**0.00**	**0.00**	**0.00**
	email	100.00	1 114.57	**0.00**	**0.00**	**0.00**	**0.00**
	email-EuAll	100.00	16 882.03	**0.14**	**0.07**	**0.14**	**0.07**
	football	**100.00**	**476.18**	**100.00**	**476.18**	**100.00**	**476.18**
	hep-th	100.00	535.64	**0.00**	**0.00**	**0.00**	**0.00**
	jazz	100.00	488.48	**6.57**	**1.79**	**6.57**	**1.79**
	lesmis	100.00	491.73	**0.00**	**0.00**	**0.00**	**0.00**
	netscience	100.00	245.15	**0.00**	**0.00**	**0.00**	**0.00**
	p2p-Gnutella04	100.00	1 227.55	**74.85**	**638.90**	**74.85**	**638.90**
	polbooks	100.00	453.97	**50.48**	**105.67**	**50.48**	**105.67**
	power	100.00	343.18	**2.83**	**4.88**	**2.83**	**4.88**
	soc-Slashdot0902	100.00	8 678.73	**24.12**	**79.04**	**24.12**	79.07
	wordassociation-2011	100.00	2 771.82	**0.00**	**0.00**	**0.00**	**0.00**
social large	G_n_pin_pout	100.00	1 088.81	**99.35**	**1 075.38**	**99.35**	**1 075.38**
	amazon-2008	100.00	901.89	**11.27**	**35.64**	**11.27**	35.66
	astro-ph	100.00	1 469.69	**0.02**	**0.00**	**0.02**	**0.00**
	caidaRouterLevel	100.00	1 843.47	**0.67**	**0.67**	**0.67**	**0.67**
	citationCiteseer	100.00	2 980.56	**0.08**	**0.03**	**0.08**	**0.03**
	cnr-2000	100.00	20 072.23	**4.32**	**378.64**	4.34	378.94
	coAuthorsCiteseer	100.00	1 005.95	**0.00**	**0.00**	**0.00**	**0.00**
	coAuthorsDBLP	100.00	1 262.80	**0.01**	**0.01**	**0.01**	**0.01**
	coPapersCiteseer	100.00	823.64	**0.01**	**0.00**	**0.01**	**0.00**
	coPapersDBLP	100.00	1 572.96	**0.01**	**0.00**	**0.01**	**0.00**
	cond-mat-2003	100.00	1 149.60	**0.03**	**0.02**	**0.03**	**0.02**
	cond-mat-2005	100.00	1 467.49	**0.00**	**0.00**	**0.00**	**0.00**
	enron	100.00	9 684.16	**0.00**	**0.00**	**0.00**	**0.00**
	loc-brightkite_edges	100.00	4 026.75	**0.00**	**0.00**	**0.00**	**0.00**
	loc-gowalla_edges	100.00	25 177.85	**0.03**	**0.01**	**0.03**	**0.01**
	preferentialAttachment	**100.00**	**3 386.36**	**100.00**	**3 386.36**	**100.00**	**3 386.36**
	road_central	100.00	261.10	**13.55**	**35.31**	**13.55**	**35.31**
	road_usa	100.00	261.17	**13.46**	**35.13**	**13.46**	**35.13**
	smallworld	100.00	550.88	**99.20**	**543.05**	**99.20**	**543.05**
	web-Google	100.00	7 560.20	**0.41**	**0.87**	0.42	0.94

Continued on next page

Table B.9 – *Continued from previous page*

		TRANSFORM		MAIN		FAST	
	Instances	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$
planar	outP500_1	100.00	213.51	**99.94**	**213.20**	**99.94**	**213.20**
	outP500_2	100.00	212.22	**99.93**	**211.87**	**99.93**	**211.87**
	outP1000_1	100.00	212.19	**99.85**	**211.53**	**99.85**	**211.53**
	outP1000_2	100.00	211.92	**99.87**	**211.29**	**99.87**	**211.29**
	outP1500_1	100.00	211.86	**99.95**	**211.60**	**99.95**	**211.60**
	outP1500_2	100.00	211.97	**99.94**	**211.68**	**99.94**	**211.68**
	outP2000_1	100.00	211.93	**99.93**	**211.58**	**99.93**	**211.58**
	outP2000_2	100.00	211.97	**99.91**	**211.53**	**99.91**	**211.53**
	outP2500_1	100.00	212.10	**99.86**	**211.45**	**99.86**	**211.45**
	outP2500_2	100.00	212.14	**99.91**	**211.72**	**99.91**	**211.72**
	outP3000_1	100.00	212.12	**99.89**	**211.62**	**99.89**	**211.62**
	outP3000_2	100.00	212.11	**99.88**	**211.56**	**99.88**	**211.56**
	outP3500_1	100.00	212.13	**99.88**	**211.55**	**99.88**	**211.55**
	outP3500_2	100.00	212.00	**99.91**	**211.58**	**99.91**	**211.58**
	outP4000_1	100.00	212.01	**99.90**	**211.55**	**99.90**	**211.55**
	outP4000_2	100.00	211.99	**99.92**	**211.61**	**99.92**	**211.61**
	outP4500_1	100.00	212.04	**99.88**	**211.50**	**99.88**	**211.50**
	outP4500_2	100.00	212.05	**99.92**	**211.69**	**99.92**	**211.69**
	outP5000_1	100.00	212.06	**99.91**	**211.64**	**99.91**	**211.64**
	outP5000_2	100.00	211.96	**99.92**	**211.56**	**99.92**	**211.56**
	overall	100.00	2 564.50	**43.84**	**185.49**	**43.84**	185.50

Table B.10: Solution size $|S|$ and time t (in seconds) needed to find it using the exact independent set solver. The time t_p is the time needed to prove the optimality. Bolt numbers indicate **best** results, gray background optimally solved instances.

		TRANSFORM			MAIN			FAST		
	Instance	$\|S\|$	t	t_p	$\|S\|$	t	t_p	$\|S\|$	t	t_p
social small	PGPgiantcompo	**2 708**	0.02	0.02	**2 708**	0.01	0.01	**2 708**	**0.01**	**0.01**
	adjnoun	**18**	<0.01	<0.01	**18**	**<0.01**	**<0.01**	**18**	<0.01	<0.01
	as-22july06	**2 026**	4.26	4.26	**2 026**	**0.95**	**0.96**	**2 026**	1.74	1.74
	celegans_metabolic	**29**	0.01	0.01	**29**	**<0.01**	**<0.01**	**29**	<0.01	<0.01
	celegansneural	**14**	0.06	0.06	**14**	**0.05**	**0.06**	**14**	0.06	0.06
	chesapeake	**3**	<0.01	<0.01	**3**	**<0.01**	**<0.01**	**3**	<0.01	<0.01
	cond-mat	**3 391**	0.03	0.03	**3 391**	0.02	0.02	**3 391**	**0.02**	**0.02**
	dolphins	**13**	<0.01	<0.01	**13**	<0.01	<0.01	**13**	**<0.01**	**<0.01**
	email	**209**	0.01	0.01	**209**	0.01	0.01	**209**	**<0.01**	**<0.01**
	email-EuAll	**696**	2.74	2.74	**696**	1.83	1.83	**696**	**1.56**	**1.56**
	football	**7**	1.36	1.39	**7**	1.36	1.39	**7**	**1.35**	**1.39**
	hep-th	**2 611**	0.01	0.01	**2 611**	0.01	0.01	**2 611**	**<0.01**	**0.01**
	jazz	**13**	<0.01	<0.01	**13**	**<0.01**	**<0.01**	**13**	<0.01	<0.01
	lesmis	**10**	<0.01	<0.01	**10**	<0.01	<0.01	**10**	**<0.01**	**<0.01**
	netscience	**477**	<0.01	<0.01	**477**	**<0.01**	**<0.01**	**477**	<0.01	<0.01
	p2p-Gnutella04	**825**	0.35	m.o.	**825**	0.35	m.o.	**825**	**0.33**	m.o.
	polbooks	**12**	**<0.01**	**<0.01**	**12**	<0.01	<0.01	**12**	<0.01	<0.01
	power	**1 465**	**<0.01**	<0.01	**1 465**	<0.01	<0.01	**1 465**	<0.01	**<0.01**
	soc-Slashdot0902	3 280	27.43	m.o.	**3 282**	19.77	m.o.	**3 282**	**3.53**	m.o.
	wordassociation-2011	**2 473**	0.28	0.28	**2 473**	0.04	0.04	**2 473**	**0.02**	**0.02**
	overall	**159**	0.02	-	**159**	0.01	-	**159**	**0.01**	-
social large	G_n_pin_pout	**7 116**	**8.18**	m.o.	**7 116**	8.46	m.o.	**7 116**	8.64	m.o.
	amazon-2008	106 533	71.17	m.o.	**106 558**	**68.58**	m.o.	106 556	70.55	m.o.
	astro-ph	**2 926**	0.28	0.28	**2 926**	0.09	0.09	**2 926**	**0.05**	**0.05**
	caidaRouterLevel	**40 138**	3.11	3.21	**40 138**	1.49	1.51	**40 138**	**0.73**	**0.74**
	citationCiteseer	**43 238**	10.39	10.39	**43 238**	2.51	2.51	**43 238**	**1.72**	**1.72**
	cnr-2000	**21 897**	1 031.59	m.o.	21 896	986.12	m.o.	**21 897**	**1 030.06**	m.o.
	coAuthorsCiteseer	**33 167**	1.22	1.22	**33 167**	0.56	0.56	**33 167**	**0.40**	**0.40**
	coAuthorsDBLP	**43 960**	2.37	2.37	**43 960**	0.73	0.73	**43 960**	**0.67**	**0.67**
	coPapersCiteseer	**26 001**	47.88	47.88	**26 001**	33.65	33.65	**26 001**	**14.70**	**14.70**
	coPapersDBLP	**35 529**	121.92	121.92	**35 529**	90.89	90.89	**35 529**	**18.21**	**18.21**
	cond-mat-2003	**5 374**	0.17	0.17	**5 374**	0.08	0.08	**5 374**	**0.06**	**0.06**
	cond-mat-2005	**6 505**	0.39	0.39	**6 505**	0.16	0.16	**6 505**	**0.10**	**0.10**
	enron	**4 090**	11.21	11.21	**4 090**	1.42	1.43	**4 090**	**1.25**	**1.26**
	loc-brightkite_edges	**12 940**	2.18	2.18	**12 940**	0.40	0.41	**12 940**	**0.18**	**0.19**
	loc-gowalla_edges	**41 590**	350.55	351.65	**41 590**	76.80	76.82	**41 590**	**70.81**	**70.83**
	preferentialAttachment	**6 397**	**15.35**	m.o.	**6 397**	15.54	m.o.	**6 397**	15.95	m.o.
	road_central	4 289 510	2 566.67	m.o.	4 289 578	2 299.59	m.o.	**4 289 639**	**2 441.79**	m.o.
	road_usa	7 296 706	3 697.55	m.o.	7 296 913	3 043.52	m.o.	**7 297 028**	**3 733.19**	m.o.
	smallworld	**6 872**	**5.19**	m.o.	**6 872**	5.36	m.o.	**6 872**	5.38	m.o.
	web-Google	**30 296**	62.47	63.40	**30 296**	**34.77**	**34.83**	**30 296**	68.16	68.22
	overall	30 065	16.58	-	**30 066**	8.38	-	**30 066**	**6.44**	-
planar	outP500_1	**19 140**	**1.64**	m.o.	**19 140**	1.75	m.o.	**19 140**	1.77	m.o.
	outP500_2	**21 894**	**2.16**	m.o.	**21 894**	2.27	m.o.	**21 894**	2.25	m.o.
	outP1000_1	**43 855**	**4.28**	m.o.	**43 855**	4.73	m.o.	**43 855**	4.76	m.o.
	outP1000_2	**45 418**	**4.10**	m.o.	**45 418**	4.53	m.o.	**45 418**	4.65	m.o.
	outP1500_1	**69 368**	**6.33**	m.o.	**69 368**	6.59	m.o.	**69 368**	6.71	m.o.
	outP1500_2	**68 571**	**6.45**	m.o.	**68 571**	6.69	m.o.	**68 571**	6.84	m.o.
	outP2000_1	**90 550**	**8.26**	m.o.	**90 550**	8.60	m.o.	**90 550**	8.76	m.o.
	outP2000_2	**90 507**	**8.44**	m.o.	90 505	9.08	m.o.	90 505	9.18	m.o.
	outP2500_1	**113 643**	**10.83**	m.o.	113 635	11.72	m.o.	113 635	11.99	m.o.
	outP2500_2	**113 073**	**10.73**	m.o.	113 066	11.30	m.o.	113 066	11.55	m.o.
	outP3000_1	135 658	13.22	m.o.	**135 659**	**14.57**	m.o.	**135 659**	14.95	m.o.
	outP3000_2	**136 101**	**13.45**	m.o.	**136 101**	15.11	m.o.	**136 101**	15.67	m.o.
	outP3500_1	161 706	13.85	m.o.	**161 709**	**15.14**	m.o.	**161 709**	15.42	m.o.
	outP3500_2	**162 533**	**14.23**	m.o.	162 528	15.73	m.o.	162 528	16.17	m.o.
	outP4000_1	178 655	17.55	m.o.	**178 671**	**18.77**	m.o.	**178 671**	19.36	m.o.
	outP4000_2	**178 862**	**17.64**	m.o.	178 858	19.12	m.o.	178 858	19.69	m.o.
	outP4500_1	198 172	21.96	m.o.	**198 176**	**23.59**	m.o.	**198 176**	24.17	m.o.
	outP4500_2	**198 440**	**21.81**	m.o.	**198 440**	22.78	m.o.	**198 440**	23.26	m.o.
	outP5000_1	**226 396**	**21.77**	m.o.	226 395	24.31	m.o.	226 395	25.15	m.o.
	outP5000_2	**33 663**	**13.39**	m.o.	**33 663**	15.23	m.o.	**33 663**	15.84	m.o.
	overall	**92 135**	**9.43**	-	**92 135**	10.19	-	**92 135**	10.41	-
	`social` and `planar`	**7 604**	1.50	-	**7 604**	1.04	-	**7 604**	**0.91**	-

Table B.11: Solution size $|S|$ and time t (in milliseconds) needed to find it for `social` graphs. The time t_p is the time needed to prove the optimality. Bolt numbers indicate **best** results in comparison. Gray background marks optimally solved instances. No competitor is able to solve these instances.

		RED2PACK_B&R			RED2PACK_HEURISTIC	
	Instance	$\|S\|$	t	t_p	$\|S\|$	t
social small	PGPgiantcompo	**2 708**	8.34	**8.74**	**2 708**	**8.31**
	adjnoun	**18**	**0.43**	**0.48**	**18**	**0.43**
	as-22july06	**2 026**	**1 737.58**	**1 739.08**	**2 026**	1 774.51
	celegans_metabolic	**29**	**4.77**	**4.84**	**29**	4.81
	celegansneural	**14**	**55.94**	**58.29**	**14**	57.74
	chesapeake	**3**	**0.13**	**0.18**	**3**	**0.13**
	cond-mat	**3 391**	**16.42**	**17.09**	**3 391**	16.92
	dolphins	**13**	0.12	**0.16**	**13**	**0.11**
	email	**209**	**3.44**	**3.58**	**209**	3.49
	email-EuAll	**696**	1 561.44	**1 561.44**	**696**	**1 463.69**
	football	**7**	1 351.83	**1 388.30**	**7**	**223.50**
	hep-th	**2 611**	**4.77**	**5.07**	**2 611**	4.85
	jazz	**13**	**2.18**	**2.18**	**13**	7.12
	lesmis	**10**	**0.16**	**0.20**	**10**	**0.16**
	netscience	**477**	**1.24**	**1.38**	**477**	1.31
	p2p-Gnutella04	825	332.37	m.o.	**837**	**1 695.12**
	polbooks	**12**	**1.97**	**2.62**	**12**	5.57
	power	**1 465**	**3.67**	**3.92**	**1 465**	17.06
	soc-Slashdot0902	3 282	3 532.80	m.o.	**3 288**	**3 314.43**
	wordassociation-2011	**2 473**	**22.04**	**22.50**	**2 473**	22.43
	overall	**159**	**11.32**	-	**159**	13.53
social large	G_n_pin_pout	7 116	8 640.57	m.o.	**7 970**	**1 706 124.41**
	amazon-2008	106 556	70 549.59	m.o.	**107 165**	**15 503 659.26**
	astro-ph	**2 926**	51.34	**51.34**	**2 926**	**50.86**
	caidaRouterLevel	**40 138**	**732.05**	**742.47**	**40 138**	799.62
	citationCiteseer	**43 238**	1 719.69	**1 719.69**	**43 238**	**1 658.55**
	cnr-2000	21 897	1 030 063.00	m.o.	**21 898**	**931 990.73**
	coAuthorsCiteseer	**33 167**	402.64	**402.64**	**33 167**	**370.94**
	coAuthorsDBLP	**43 960**	672.32	**672.32**	**43 960**	**630.44**
	coPapersCiteseer	**26 001**	**14 702.18**	**14 702.18**	**26 001**	14 979.70
	coPapersDBLP	**35 529**	18 210.66	**18 210.66**	**35 529**	**17 866.85**
	cond-mat-2003	**5 374**	**59.71**	**59.71**	**5 374**	61.45
	cond-mat-2005	**6 505**	**99.69**	**101.85**	**6 505**	100.22
	enron	**4 090**	1 254.90	**1 258.90**	**4 090**	**1 233.50**
	loc-brightkite_edges	**12 940**	182.96	**185.44**	**12 940**	**182.93**
	loc-gowalla_edges	**41 590**	**70 812.67**	**70 825.69**	**41 590**	71 102.93
	preferentialAttachment	6 397	15 947.59	m.o.	**7 034**	**2 608 316.99**
	road_central	4 289 639	2 441 793.42	m.o.	**4 499 839**	**35 841 231.91**
	road_usa	7 297 028	3 733 186.22	m.o.	**7 647 882**	**35 970 721.50**
	smallworld	6 872	5 379.45	m.o.	**7 946**	**11 329 895.89**
	web-Google	**30 296**	68 160.24	**68 222.93**	**30 296**	**67 864.36**
	overall	30 066	6 442.49	-	**30 756**	**26 377.56**
	overall `social`	2 184	270.05	-	**2 210**	**597.37**

Table B.12: Solution size $|S|$ and time t (in milliseconds) needed to find it for `erdos` and `cactus` graphs. The time t_p is the time needed to prove the optimality. Bolt numbers indicate **best** results in comparison. Gray background marks optimally solved instances.

		GEN2PACK		RED2PACK_B&R			RED2PACK_HEURISTIC	
	Instance	$\|S\|$	t	$\|S\|$	t	t_p	$\|S\|$	t
cactus	cac50	15	25 291.12	**17**	**0.05**	**0.09**	**17**	**0.05**
	cac100	28	85 674.26	**31**	**0.09**	**0.14**	**31**	**0.09**
	cac150	42	181 973.07	**49**	**0.31**	**0.31**	**49**	13.49
	cac200	55	313 878.10	**65**	**10.27**	**10.70**	**65**	14.47
	cac250	68	484 267.30	**82**	0.69	**0.79**	**82**	**0.25**
	cac300	80	712 988.40	**100**	**0.83**	**0.83**	**100**	13.07
	cac350	92	963 595.66	**116**	**12.89**	**13.35**	**116**	13.66
	cac400	103	1 230 210.54	**133**	**14.58**	**15.26**	**133**	30.00
	cac450	114	1 574 182.46	**148**	**12.22**	**12.66**	**148**	13.70
	cac500	126	1 987 312.28	**166**	23.73	**25.24**	**166**	**17.65**
	cac550	136	2 389 870.11	**179**	**15.61**	**15.98**	**179**	18.59
	cac600	146	2 779 427.76	**199**	0.24	**0.30**	**199**	**0.23**
	cac650	158	3 328 099.16	**214**	**28.90**	**29.19**	**214**	58.65
	cac700	165	3 859 839.95	**232**	**7.68**	**7.72**	**232**	8.82
	cac750	172	4 477 170.66	**250**	**11.58**	**11.58**	**250**	19.19
	cac800	184	5 088 716.64	**264**	**18.26**	**18.87**	**264**	19.84
	cac850	196	5 648 560.45	**282**	26.46	**27.04**	**282**	**26.27**
	cac900	205	6 490 942.35	**300**	**10.92**	**11.35**	**300**	29.14
	cac950	214	7 281 058.31	**315**	**10.18**	**10.18**	**315**	17.17
	cac1000	223	8 084 805.96	**332**	**17.02**	**17.46**	**332**	58.03
	overall	104	1 384 007.11	**137**	**4.26**	**4.66**	**137**	7.30
erdos	Erdos37-2	**9**	21 118.31	**9**	0.07	**0.11**	**9**	**0.06**
	Erdos37-16	8	21 119.42	**9**	**1.13**	**1.25**	**9**	2.31
	Erdos37-23	9	21 104.45	**10**	**0.09**	**0.13**	**10**	**0.09**
	Erdos37-44	**7**	21 318.79	**7**	0.99	**1.19**	**7**	**0.14**
	Erdos37-45	10	21 328.35	**11**	0.09	**0.13**	**11**	**0.08**
	Erdos38-2	**9**	21 706.27	**9**	**0.07**	**0.11**	**9**	**0.07**
	Erdos38-14	8	21 915.50	**9**	**0.59**	**0.68**	**9**	2.50
	Erdos38-18	6	21 969.36	**7**	**1.25**	**1.35**	**7**	4.59
	Erdos38-46	8	21 927.82	**9**	**1.05**	**1.19**	**9**	14.31
	Erdos38-48	8	21 643.77	**9**	**0.96**	**1.17**	**9**	1.30
	Erdos39-14	**9**	22 997.65	**9**	0.16	**0.16**	**9**	**0.08**
	Erdos39-22	10	22 626.53	**11**	**0.06**	**0.10**	**11**	**0.06**
	Erdos39-25	**8**	23 005.90	**8**	1.30	**1.69**	**8**	**0.48**
	Erdos39-29	9	22 800.22	**10**	**0.07**	**0.12**	**10**	**0.07**
	Erdos39-44	8	22 758.11	**9**	0.11	**0.15**	**9**	**0.10**
	Erdos40-0	9	23 138.60	**10**	**0.18**	**0.18**	**10**	1.79
	Erdos40-4	8	19 244.44	**9**	**0.26**	**0.26**	**9**	1.21
	Erdos40-8	7	19 161.29	**8**	**1.87**	**2.02**	**8**	12.74
	Erdos40-10	**10**	23 474.02	**10**	**0.22**	**0.22**	**10**	1.04
	Erdos40-43	8	19 918.83	**9**	**1.07**	**1.18**	**9**	3.51
	overall	8	21 679.67	**9**	**0.31**	**0.38**	**9**	0.53

Table B.13: Solution size $|S|$ and time t (in seconds) needed to find it for `planar` graphs. The time t_p is the time needed to prove the optimality. Bolt numbers indicate **best** results in comparison. Gray background marks optimally solved instances.

		Apx-2P+Imp2P ($h = 100$)		Apx-2P+Imp2P ($h = 50$)		red2pack b&r		red2pack heuristic	
	Instance	$\|S\|$	t	$\|S\|$	t	$\|S\|$	t	$\|S\|$	t
planar	outP500_1	**20 333**	**97.51**	20 327	22.99	19 140	1.77	20 332	26 166.03
	outP500_2	**24 187**	**1 978.02**	24 144	58.42	21 894	2.25	24 181	12 631.72
	outP1000_1	**48 575**	**1 066.37**	48 472	145.79	43 855	4.76	48 567	29 215.74
	outP1000_2	49 429	692.06	**49 430**	**111.35**	45 418	4.65	49 416	28 336.12
	outP1500_1	74 232	794.47	74 170	166.89	69 368	6.71	**74 292**	**31 577.14**
	outP1500_2	73 901	2 638.12	73 813	161.62	68 571	6.84	**73 922**	**31 015.92**
	outP2000_1	-	t.o.	98 519	333.63	90 550	8.76	**98 555**	**32 735.67**
	outP2000_2	98 643	2 062.41	98 558	251.90	90 505	9.18	**98 669**	**33 103.01**
	outP2500_1	122 826	1 092.08	122 761	233.20	113 635	11.99	**122 831**	**34 384.01**
	outP2500_2	**122 322**	**4 897.24**	122 275	319.30	113 066	11.55	122 282	34 018.53
	outP3000_1	**146 748**	**1 739.13**	146 622	258.05	135 659	14.95	146 696	35 371.22
	outP3000_2	**147 544**	**1 633.60**	147 360	341.67	136 101	15.67	147 522	35 483.06
	outP3500_1	**171 303**	**14 025.36**	171 127	490.08	161 709	15.42	171 264	34 414.78
	outP3500_2	-	t.o.	172 720	351.27	162 528	16.17	**172 956**	**35 585.91**
	outP4000_1	-	t.o.	196 172	490.55	178 671	19.36	**196 189**	**35 262.82**
	outP4000_2	196 218	19 284.45	196 076	540.26	178 858	19.69	**196 243**	**35 367.19**
	outP4500_1	**220 799**	**4 932.77**	220 677	505.56	198 176	24.17	220 737	35 650.42
	outP4500_2	**221 406**	**11 015.21**	221 283	503.34	198 440	23.26	221 345	35 643.68
	outP5000_1	**244 641**	**12 413.34**	244 350	595.94	226 395	25.15	244 548	35 605.20
	outP5000_2	245 031	6 309.16	245 038	603.84	33 663	15.84	**245 223**	**35 789.86**
	overall	-	-	110 009	255.04	92 135	10.41	**110 095**	**31 706.65**