Engineering Good Upper Bounds for the Maximum Weight Independent Set Problem

Raphael Heuberger

September 25, 2025

4218120

Bachelor Thesis

at

Algorithm Engineering Group Heidelberg
Heidelberg University

Supervisor:
Univ.-Prof. PD. Dr. rer. nat. Christian Schulz

Co-Supervisors:
Dr. Ernestine Großmann
Dr. Kenneth Langedal

Acknowledgments

I would like to express my sincere gratitude to Prof. Christian Schulz. He offered me this wonderful insight into scientific algorithm engineering, and his teaching gave me a pleasant introduction to the subject of this thesis. I also want to thank my co-supervisors Ernestine Großmann and Kenneth Langedal. They supported me from start to finish and suggested this amazing topic to me. Without their ideas and detailed feedback, I would not have achieved this satisfactory result. Last but not least, I am deeply grateful to the people who shared the long days in the library with me. Together with them, I was able to enjoy this time even during the most stressful phases. They had a greater impact on the outcome of this work than they imagine.

Hiermit versichere ich, dass ich die Arbeit selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich aus fremden Werken Übernommenes als fremd kenntlich gemacht habe. Ferner versichere ich, dass die Übermittelte elektronische Version in Inhalt und Wortlaut mit der gedruckten Version meiner Arbeit vollständig Übereinstimmt. Ich bin einverstanden, dass diese elektronische Fassung universitätsintern anhand einer Plagiatssoftware auf Plagiate Überprüft wird.

Heidelberg, September 25, 2025

Raphael Heuberger

Abstract

In this thesis, we present four algorithms for calculating upper bounds for the MAXIMUM WEIGHT INDEPENDENT SET (MWIS) problem. This NP-complete problem consists of finding a set of nodes for a graph such that the nodes are not connected to each other via an edge. Additionally, the sum of the node weights must be as large as possible. To reach good upper bounds, we leverage the clique cover LP relaxation of the MWIS problem. Therefore, we apply the clique cover relaxation to the cover calculated by KaMIS and to our own exhaustive clique cover. KaMIS is a state-of-the-art, branch-and-bound algorithm for solving the MWIS problem optimally. For our exhaustive clique cover, we also present an algorithm that can iteratively improve the upper bound. In the evaluation, we examine different configurations for our algorithms in terms of solution quality and running time. Our comparison with the upper bounds of KaMIS shows that we obtain significantly better results. While the *best* bound of KaMIS is 17,71% larger than our comparison lower bound, our exhaustive clique cover approach reaches a *worst* gap of 3,284%. However, for some graphs, we have extremely long runtimes of up to 110 seconds for our non-iterative approach. This also applies to the iterative algorithm, although we show that with the right configuration, a small time saving is possible without compromising the solution quality. We also determine for vehicle routing graphs that two of our algorithms can compete with a state-of-the-art algorithm, even though the latter benefits from precomputed clique covers. Furthermore, we demonstrate that two of our algorithms are capable of finding the optimal solution for some of the vehicle routing instances.

Contents

CHAPTER 1

Introduction

1.1 Motivation

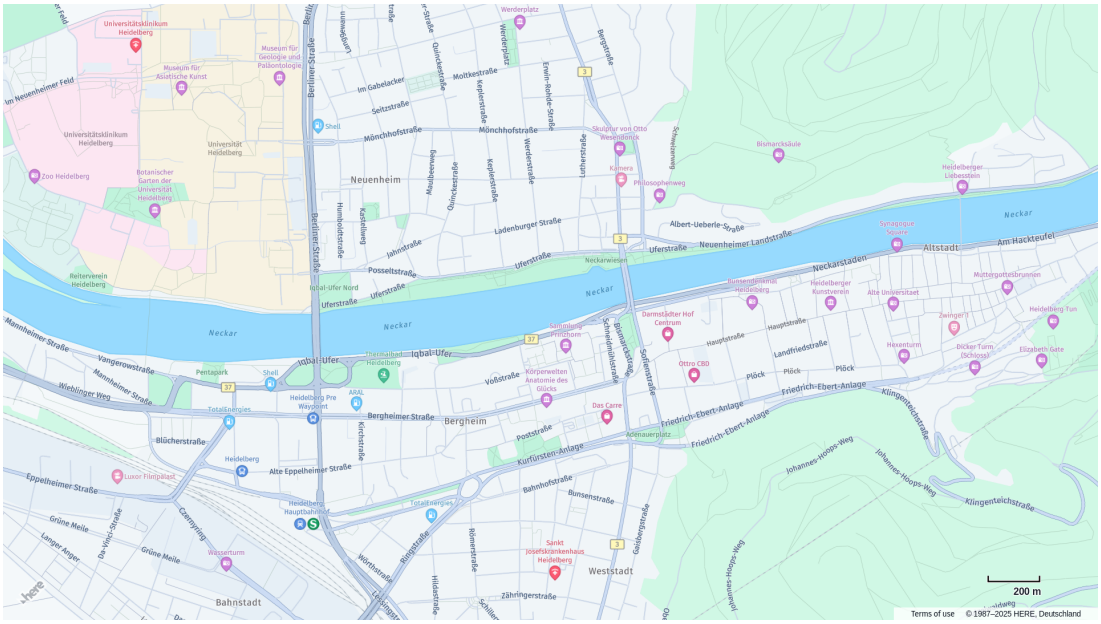

Figure 1.1: Screenshot from HERE WeGo [1] showing Heidelberg. The labels for street names or locations are shown depending on their relevance. Less important labels are not visible if their size conflicts with the other labels.

Mapmakers often face the problem that they want to provide more information about an area than can fit on the map. For small regions, it is easy to decide manually which information can be displayed so that the labels on the map do not overlap. However, this problem becomes much more difficult if you want to display as much relevant information

as possible for each area size. This label placement problem can be formally described by assigning each label a position, a minimum size and a weight that describe its relevance. From this information, you can build a graph with weighted vertices. Each label corresponds to a vertex, and all overlapping vertex-labels are connected by an edge. Solving the MAXIMUM WEIGHT INDEPENDENT SET (MWIS) problem on this conflict graph results in an optimal solution for the labeling problem [4, 13]. This is due to the definition of the MWIS problem, which desires a subset of non-adjacent nodes that has the largest possible sum of the node weights.

This example for the MAXIMUM WEIGHT INDEPENDENT SET problem is one of a wide field of real-world applications ranging from computer vision [5] to wireless network scheduling [25, 28, 29] to vehicle routing [7, 8]. Even in bioinformatics, the MWIS can be used to identify structurally similar proteins with different chain topologies [9]. Additionally, there are problems in graph theory that are closely related to the MWIS. The best known are the MINIMUM WEIGHT VERTEX COVER and the MAXIMUM WEIGHT CLIQUE problem [14, 27]. Due to the nature of these problems, it is possible to transform the graph so that a solution to one problem can be found by reinterpreting the solution to the other problem. Consequently, our contributions have an impact that exceeds the scope of the MWIS problem.

Since we know of the relevance of MWIS, there are numerous solution strategies for the problem [14]. The main reason for the wide variety of optimal and heuristic solvers is that the MWIS problem is NP-hard [11]. This also results in the demand for good upper bounds for the independent set. Exact solvers such as KaMIS [19] and other branch-and-bound-based approaches require an upper bound to improve their running times. On the other side, there are heuristics that are able to find (near) optimal solutions [8, 15, 18, 21]. To assess the results of these heuristics, we need either an optimal solution, which is very hard to compute for large graphs, or an upper bound that can usually be computed in polynomial time.

1.2 Our Contribution

In this thesis we present three new algorithms for the upper bound computation of the MAXIMUM WEIGHT INDEPENDENT SET problem. Our algorithms leverage the clique cover relaxation introduced by Haller et al. [18], which is the LP relaxation of the MWIS ILP formulated with clique constraints. The core of our approach is the computation of the clique cover that will be used for the relaxation. Our first algorithm applies LP relaxation to a small clique cover that previously served as an upper bound itself. In contrast, for the other two algorithms, we introduce a new computation of a large exhaustive clique cover that enables them to find near-optimal upper bounds. Finally, we propose a version that iteratively improves the solution by improving the exhaustive clique cover. The intermediate results could also be used by branch-and-bound algorithms to check earlier whether a branch cannot increase the independent set.

To evaluate the solution and running time quality of our algorithms, we compare each one with its previous version. Therefore, we use a subset of the Amazon Vehicle Routing instances [7] and the Meta-Segmentation for Cell Detection dataset provided by Haller et al. [18]. Additionally, we compare our results with two state-of-the-art algorithms, one providing an upper bound and the other a lower bound.

1.3 Structure

The remainder of this thesis is organized as follows: Chapter 2 consists of the necessary fundamentals of graph theory and linear programming. In Chapter 3, we go through the related work on solving approaches for the MAXIMUM WEIGHT INDEPENDENT SET problem, including strategies to compute upper bounds for this problem. Our underlying approach and the implementation are described in Chapter 4, where we provide a detailed overview of our algorithms. Chapter 5 provides the evaluation of our algorithms, and Chapter 6 closes this work with a conclusion and ideas for future work.

CHAPTER 2

Fundamentals

2.1 Graph Theory: Definitions

To convey a good understanding of this work, we provide an overview of the necessary basic knowledge. The main topics are graph theory and linear programs. Therefore, we present the definitions for weighted undirected graphs and the properties they can contain. Additionally, we specify graph problems like the Maximum Weight Independent Set problem and its formulations as an integer linear program.

Weighted Undirected Graphs

An *undirected graph* is a pair $G = (V, E)$, where V is a finite set of *vertices* (or *nodes*) and $E \subseteq \{\{u, v\} \subseteq V : u \neq v\}$ is a set of *edges*. A *weighted undirected graph* is a tuple $G = (V, E, \omega)$ with V and E defined like above. The function $\omega : V \to \mathbb{Z}$ assigns an integer weight to each vertex. In this work, we always consider undirected weighted graphs.

Properties of Undirected Graph

In an undirected graph $G = (V, E, \omega)$ two vertices $u, v \in V$ are *adjacent*, if there is an edge $e = \{u, v\} \in E$. For two vertices $v_1, v_k \in V$ there is a *path* in G from v_1 to v_k if there is a sequence of distinct vertices $v_1, ..., v_k$, so that $\{v_i, v_{i+1}\} \in E$ for all $i = 1..k$. A graph is *connected* if there is a path from v to u for all $v, u \in V : v \neq u$. The *degree* of a vertex $v \in V$ is defined by the number of edges of v and is denoted by *deg(v)*. The *neighborhood* of v is defined by $N(v) = \{u \in V : \{v, u\} \in E\}$ and describes all nodes that are directly connected by an edge with v. By this definition we get $deg(v) = |N(v)|$.

Clique

A *clique* is a set of vertices, so that each vertex has an edge to all other vertices in the clique. Formally, in a graph $G = (V, E)$ a clique is a subset $C \subseteq V$ such that $\{\forall u, v \in C : \{v, u\} \in E\}$.

Weighted Clique Cover

A *clique cover* is a collection of cliques $C_1, C_2, \ldots, C_k$ such that each vertex in the graph is contained in at least one of the cliques. Formally, a clique cover is a set of cliques $C_i \in V$ such that $\bigcup_{i=1}^{k} C_i = V$. A *weighted clique cover* is a clique cover of a weighted graph $G = (V, E, \omega)$ where each clique has an associated non-negative weight, so that for each vertex $v \in V$ the sum of the weights of the cliques containing v is at least the weight of v.

Maximum Weight Independent Set

An *independent set* (IS) of an undirected graph $G = (V, E)$ is a subset $I \subseteq V$ which only contains vertices that are not adjacent (connected by an edge). A *maximum independent set* (MIS) of the graph G is an independent set of G with the largest possible size. The size $|I|$ of an MIS *I* is defined by the number of nodes in the set and is denoted by the independence number $\alpha(G)$. Accordingly, there could exist multiple maximum independent sets of the same size for G. The size of a *weight independent set* (WIS) of an graph $G = (V, E, \omega)$ is defined by the sum of the node weights, instead of the number of nodes: $|I| = \sum_{v \in I} w(v)$. Therefore, the *maximum weight independent set* (MWIS) is an independent set with the largest sum of node weights, which is denoted by $\alpha_\omega(G)$.

Minimum Weight Vertex Cover

A *vertex cover* (VC) of an undirected graph $G = (V, E)$ is a subset $C \subseteq V$ such that every edge $\{u, v\} \in E$ has at least one vertex in C. A *minimum vertex cover* (MVC) of the graph G is a vertex cover of G with the smallest possible size. The size $|C|$ of an MVC C is defined by the number of vertices in the set. Like for MIS, there can exist multiple minimum vertex covers of the same size for G. The size of a *weight vertex cover* (WVC) of a graph $G = (V, E, \omega)$ is defined by the sum of the vertex weights, instead of the number of vertices: $|C| = \sum_{v \in C} \omega(v)$. Therefore, the *minimum weight vertex cover* (MWVC) is a vertex cover with the smallest sum of vertex weights.

Maximum Weight Clique

A *maximum clique* (MC) of an undirected graph $G = (V, E)$ is a clique of G that has the largest number of vertices. The size $|K|$ of an MC K is defined by the clique size. There can also be multiple maximum cliques of the same size for G. The size of a *weight clique*

(WC) of a graph $G = (V, E, \omega)$ is defined by the sum of the vertex weights, instead of clique size: $|K| = \sum_{v \in K} \omega(v)$. Therefore, the *maximum weight clique* (MWC) is a clique with the largest sum of vertex weights.

2.2 Linear Programming

Linear Programming (LP) is a method to solve optimization problems that can be translated into a linear objective function with linear constraints.

Linear Program

A *linear program* (also LP) is the description of an optimization problem in the following form:

$$\begin{aligned} \text{maximize} \quad & c^\top x \qquad \text{(or minimize)} \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0, \end{aligned}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables a solver has to find,
- $c \in \mathbb{R}^n$ is the vector of objective coefficients,
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix,
- $a_{ij} \in A$ are the constraint coefficients,
- $b \in \mathbb{R}^m$ is the vector of constraint bounds

For better readability, we will prefer this more general form to describe our LPs:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \qquad \text{(or minimize)} \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \forall i = 1, \ldots, n \\ & l \leq x_j \leq u \qquad \forall j = 1, \ldots, n \\ \text{with} \quad & l, u \in \mathbb{R} \end{aligned}$$

If there is a feasible solution, an LP solver will find a x that maximizes (or minimizes) the objective function $c^\top x$ while satisfying the constraints $Ax \leq b$ and $x \geq 0$.

Integer Linear Program

An *integer linear program* (ILP) is a linear program where the decision variables x_j are restricted to integer values. Accordingly, the difference is that $x \in \mathbb{Z}^n$ instead of $x \in \mathbb{R}^n$. The MWIS problem of a graph $G = (V, E, \omega)$ can be formulated as the following two ILPs. However, the ILP using clique constraints depends on a clique cover K, which has for each edge $e = \{u, v\} \in E$ a clique $C \in K$ so that $e \in C$.

Using Edge Constrains

$$\begin{aligned} \text{maximize} \quad & \sum_{u \in V} \omega(u) \cdot x_u \\ \text{subject to} \quad & x_u + x_v \leq 1, \quad \forall (u, v) \in E \\ & x_u \in \{0, 1\}, \quad \forall u \in V \end{aligned} \tag{2.1}$$

Using Clique Constraints

$$\begin{aligned} \text{maximize} \quad & \sum_{u \in V} \omega(u) \cdot x_u \\ \text{subject to} \quad & \sum_{u \in C_i} x_u \leq 1, \quad \forall C_i \in K \\ & x_u \in \{0, 1\}, \quad \forall u \in V \\ \text{where} \quad & K \text{ is a clique cover that covers all edges} \end{aligned} \tag{2.2}$$

CHAPTER 3

Related Work

The MAXIMUM WEIGHT INDEPENDENT SET (MWIS) problem is one of the most well-studied NP-hard problems. Therefore, there is a huge set of exact and heuristic solvers. In the following, we provide an overview of the most important related work, starting with different solving strategies. Since the main topic of our work is the upper bound computation, we also consider how to obtain such a bound by using MWIS-related problems. Thereafter, we present approaches that directly compute an upper bound for the maximum independent set.

3.1 Maximum Weight Independent Set

We start by introducing exact solving strategies that guarantee to find the maximum weight independent set. Later, we focus on heuristic solving approaches, which generally offer no guarantee but often deliver near-optimal weighted independent sets.

Exact Solver

As stated in Chapter 2.2, the MAXIMUM WEIGHT INDEPENDENT SET problem can be formulated as an integer linear program (ILP). Both the ILP with edge constraints and the ILP with clique constraints can be used to obtain a maximum independent set [8, 18]. There are ILP solvers like the open source GNU Linear Programming Kit (GLPK) [10] and the commercial Gurobi [17]. However, those solvers are designed to solve any kind of linear problem, not only the MWIS problem.

That aside, there are branch-and-bound algorithms, dealing explicitly with the MWIS problem [26]. The *branching* part of these algorithms recursively divides the problem into two subproblems. For the first subproblem, it is assumed that a node is part of the independent set, while for the second subproblem, it is excluded. Overall, the algorithm keeps track of the best solution found. To improve the running time, a lower and upper bound is used for *pruning*. Therefore, the algorithm quits the branch of a subproblem if

its upper bound is less than or equal to the current best solution. The procedure of upper bound computation for branch-and-bound algorithms is described in Section 3.3. Lower bounds are provided by heuristics discussed in the next subsection.

The branch-and-bound approach was extended to a branch-and-reduce search [2]. This method utilizes *kernalization* by applying reduction rules to the initial graph as well as during branching. For the MWIS problem, several reductions were developed [14, 20]. From the optimal solution of the reduced graph, the original MWIS can be derived. In addition to reduction rules, there are transformations, which increase the graph [12]. However, it is possible to get overall smaller reduced graphs after applying reductions to it. The current state-of-the-art branch-and-reduce algorithm for the MWIS problem is KaMIS [19, 12], which implements reductions and increasing transformations.

Heuristic Solver

Branch-and-reduce algorithms have the downside of an exponential running time. Heuristic algorithms, on the other hand, aim to find near-optimal solutions in a short period of time. A common approach for the independent set problem is the iterative local search (ILS). This metaheuristic uses (x, y)-swaps to iteratively improve the initial solution. Therefore, it considers the neighborhood, removes $x \in \mathbb{N}$ nodes from the solution, and adds $y \in \mathbb{N}$ instead. For the MWIS problem, Nogueira et al. developed the hybrid iterated local search heuristic ILS-VND [22] (often called HILS). It is derived from the ARW heuristic [3] for the unweighted problem. Dong et al. further improved the HILS approach. They present METAMIS [8], a heuristic that focuses on the Amazon Vehicle Routing (AVR) instances, which are especially hard to reduce and to solve. However, the state-of-the-art algorithms for this dataset are the Concurrent Hybrid Iterated Local Search heuristic (CHILS) presented by Großmann et al. [15] and the Bregman-Sinkhorn algorithm (BSA) [18]. CHILS also expands on the HILS heuristic and is explicitly designed to handle large graphs of varying densities. The approach of Haller et al. for their Bregman-Sinkhorn algorithm differs from the rest of the outlined heuristics. Instead of performing local search, they address a family of clique cover LP relaxations. However, they also introduce a new upper bound computation for the MWIS, which will be presented in the corresponding Section 3.3.

3.2 Related Problems

This section addresses related problems of MWIS and their connection to the upper bounds of this problem. Two such complementary problems are the MINIMUM WEIGHT VERTEX COVER (MWVC) and the MAXIMUM WEIGHT CLIQUE (MWC) problem [14]. Both problems have the property that their results can be directly transferred to an MWIS. By solving one of the problems, also an MWIS problem is solved. However, it is important to mention that only in the case of the vertex cover it is possible to receive the corresponding independent set for the same graph. In the case of the MWC problem, the MWC in a graph

G corresponds to an MWIS in the complement graph[1] $\overline{G}$ [27]. Consequently, it is possible to compute an upper bound for the MWIS by computing an MWC upper bound in the complement graph. However, this is not practical for very large sparse graphs, since the complement would contain too many edges for an efficient computation.

The MWVC, on the other hand, is for all practical purposes equivalent to the MWIS, since the solution L of the MWVC problem in graph G corresponds to an MWIS $I = V \setminus L$ in the same graph [14]. Since the MWVC problem is a minimization problem, a lower bound there corresponds to an upper bound of the MWIS. The calculation of such a lower bound is presented, for example, by Wang et al. in their paper about a branch-and-bound algorithm for the MWVC problem [24].

3.3 Upper Bound Computation

For their branch-and-bound approach, Warren and Hicks take advantage of the circumstance that every weighted clique cover yields an upper bound for the maximum weight independent set [26]. The bound is computed by the sum of the clique weights. Accordingly, the resulting quality of the bound depends on the weight of the clique cover. This approach is still used in practice, since the state-of-the-art branch-and-reduce algorithm KaMIS utilizes the same method.

Another opportunity is offered by the ILP representation of the MWIS problem. Haller and Savchynskyy compute an upper bound by relaxing the ILP to an LP by removing the integer constraints of the decision variable. They call this LP relaxation *edge relaxation*, which is depicted in Figure 3.1. One advantage of the edge relaxation is the *persistency property* [18]. This means that for all nodes with integer values in the LP solution, there is an optimal ILP solution in which these nodes take on the same values. However, Haller

Edge Relaxation

$$\begin{aligned} \max \quad & \sum_{u \in V} \omega(u) \cdot x_u \\ \text{s. t.} \quad & x_u + x_v \leq 1, \quad \forall (u,v) \in E \\ & x_u \in [0,1], \quad \forall u \in V \end{aligned} \tag{3.1}$$

Clique Cover Relaxation

$$\begin{aligned} \max \quad & \sum_{u \in V} \omega(u) \cdot x_u \\ \text{s. t.} \quad & \sum_{u \in C_i} x_u \leq 1, \quad \forall C_i \in K \\ & x_u \in [0,1], \quad \forall u \in V \\ \text{where} \quad & K \text{ is clique cover covering} \\ & \text{all nodes and edges} \end{aligned} \tag{3.2}$$

[1] The complement graph $\overline{G}$ of a graph G has the same vertices like G, but only then an edge between two nodes iff there was no edge between these vertices in G.

and Savchynskyy also show the limitations of the edge relaxation by creating an example in which the LP solution does not contain a single integer value.

To further improve the upper bound, they combine the edge relaxation approach with that of the clique cover by using clique constraints instead of edge constraints 3.2. However, their definition of this *clique cover relaxation* requests a clique cover that covers not only all nodes *but also all edges*. In addition, they show that the clique cover relaxation produces at least as good results as edge relaxation by proving that for two clique cover relaxations A and B with two different clique covers, the solution of A is at least as tight to the optimal solution as that of B if every clique from B is a subset of a clique from A.

CHAPTER 4

Upper Bound Engineering

This chapter introduces two new algorithms for calculating a high-quality upper bound for the MAXIMUM WEIGHT INDEPENDENT SET problem. To ensure a good understanding of the algorithms, we first present the underlying approach before we show a high-level overview. Thereafter, we provide a detailed description of the implementation, starting with a previous version of the two algorithms that will be used for the evaluation.

4.1 Algorithm Design

4.1.1 Underlying Approach

The two algorithms introduced here are based on the same LP relaxation approach as in the Bregman-Sinkhorn Algorithm (BSA) by Haller et al. [18]. This approach is based on the fact that the maximum independent set problem can be formulated as an integer linear program (2.1) and that the LP relaxation of this ILP yields an upper bound [7, 18]. The proof for this is quite intuitive: The only difference between an integer linear program and its LP relaxation is that the integer constraint of the decision variables is removed. Since all decision variables can still take integer values in the linear program relaxation, the solution set of the ILP is a subset of the solution set of the LP relaxation. If we now consider maximization problems, we will never obtain an optimal LP relaxation solution that is smaller than the optimal ILP solution. Thus, the optimal solution of the MWIS LP relaxation is an upper bound.

Since the MWIS problem can be formulated as ILP in two ways (2.1 and 2.2), we obtain the edge relaxation and clique cover relaxation proposed by Haller et al. As they also show that clique cover relaxation is tighter than edge relaxation, this thesis focuses on clique cover relaxation. The difference between the BSA approach and the approach presented in this thesis lies in the clique cover used for the clique cover relaxation. Since BSA does not compute its own clique cover but uses graphs represented as clique covers, the main

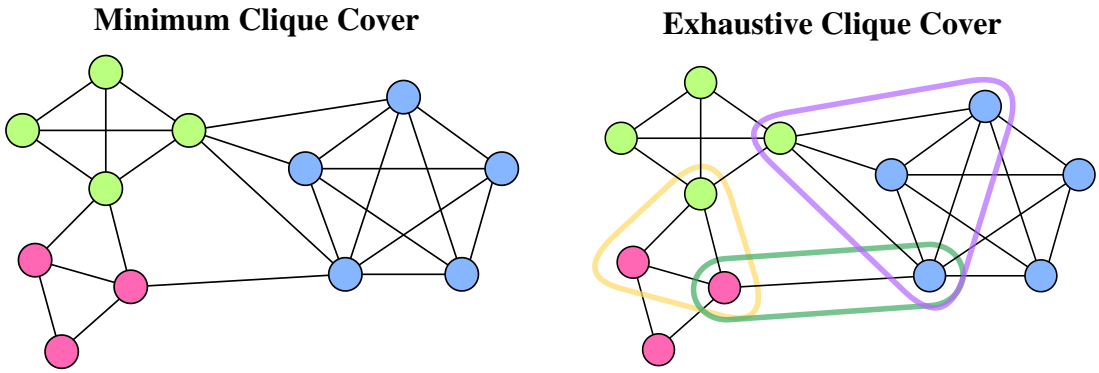

Figure 4.1: Minimum clique cover (left) containing three cliques vs. maximum exhaustive clique cover (right) containing six cliques. Cliques are indicated by node colors or surrounding colored borders.

innovation of our algorithm comes from the clique cover computation. Usually, minimal weight clique covers are desired. This is because a smaller cover leads to a smaller upper bound, if you directly derive the bound from the clique weights [26]. However, our approach contains the intermediate step of the LP relaxation, and adding more cliques as additional constraints leads to a more constrained search space. Therefore, we aim to add further cliques to our clique cover, even if it already covers all vertices. The desired clique cover is referred to as *exhaustive clique cover* and is defined below. The difference between a minimal clique cover and an exhaustive clique cover is depicted in Figure 4.1.

Definition 1: Exhaustive Clique Cover

An *exhaustive clique cover* is a set of maximal cliques that cover all nodes and edges of a graph G. It is characterized by containing far more cliques than are necessary to satisfy these conditions.

Definition 2: Maximum Exhaustive Clique Cover

A *maximum exhaustive clique cover* is an exhaustive clique cover that contains all maximal cliques of the graph G.

4.1.2 Algorithm Overview

An overview of the two algorithms *Exhaustive Clique Cover Relaxation* (ECCR) and *Iterative Exhaustive Clique Cover Relaxation* (I-ECCR) follows. As the name suggests, the main difference between the two algorithms is that the first one calculates an upper bound once, while the second one improves the result iteratively.

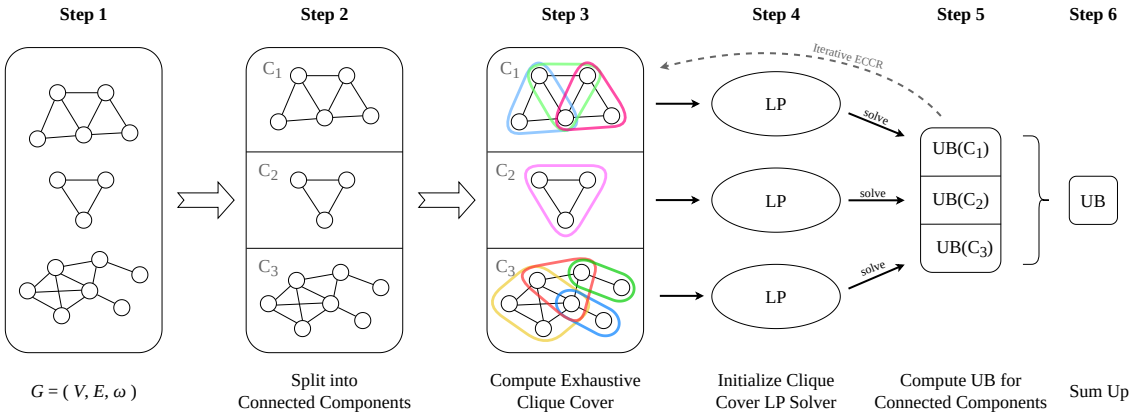

Figure 4.2: High-level overview of the ECCR and I-ECCR algorithms for the MWIS upper bound computation.

The structure of both algorithms is shown in figure 4.2. They each consist of 6 steps. First, the graph G is read in and then divided into its connected components. This offers the advantage that in the following steps, the components can be regarded as individual connected graphs $C_1, \ldots, C_n$. In steps three to five, all components are iterated through one by one. In this process, all three steps are executed on one component before moving on to the next.

Starting with step three, an exhaustive clique cover is calculated, which does not necessarily have to be a maximum exhaustive clique cover. Using this cover, the MWIS problem is formulated as a clique cover relaxation in step four, and an LP solver is initialized with it. The fifth step consists of finding an optimal solution for the LP. This value is then stored as an upper bound for the component, and the non-iterative ECCR algorithm moves on to the next component. The I-ECCR algorithm, on the other hand, jumps back to step three and attempts to add further cliques to the exhaustive clique cover. After extending the cover, steps four and five follow as usual, the upper bound gets updated, and the next iteration starts. Finally, if no new cliques were found or the new LP solution did not improve the upper bound, the iterative algorithm moves on to the next component.

After a bound has been calculated for all components, these are summed up, resulting in an upper bound of the MWIS for the entire graph G. This is allowed without further action, since no vertex from one component can influence the solution value of a vertex from another component, as there are no edges between the connected components.

Looking at the two algorithms, it becomes clear that the I-ECCR algorithm is an extension of the ECCR algorithm. If I-ECCR is forced to perform only a single iteration per component, its workflow is the same as that of ECCR. Therefore, it should always be possible for the iterative algorithm to calculate an upper bound that is at least as good as the non-iterative algorithm.

4.2 Implementation

The next three subsections are based on the development of the algorithms. For comparison, we created an improved version of the KaMIS upper bound computation by applying the clique cover relaxation on its clique cover. That way, we can isolate the importance of our exhaustive clique cover strategy. Accordingly, Section 4.2.1 focuses on the implementation of the clique cover relaxation using the KaMIS clique cover. The next Section 4.2.2 addresses the computation of an own exhaustive clique cover. In combination with the implementation of the clique cover relaxation from the previous section, we obtain the Exhaustive Clique Cover Relaxation (ECCR) algorithm, which will be discussed in detail. Lastly, the third Section 4.2.3 describes the extension of the ECCR to the Iterative Exhaustive Clique Cover Relaxation (I-ECCR) algorithm.

4.2.1 KaMIS Clique Cover Relaxation

KaMIS is an exact MWIS solver that computes a weighted clique cover to obtain an upper bound for its branch-and-bound algorithm. We first start by introducing the KaMIS. In addition, there is a separate paragraph on the LP solvers, which we use for the clique cover relaxation.

Algorithm 1: KaMIS Clique Cover (KCC)

Input: Connected graph $G = (V, E, \omega)$
Output: Clique cover and MWIS upper bound of G

Function `KaMISCliqueCover` ($G = (V, E, \omega)$)**:**
- $U \leftarrow V$ sorted descending by node weight (and degree on equality)
- $K \leftarrow \emptyset$ `// clique cover`
- $W \leftarrow 0$ `// vector of clique weights`
- **foreach** $v \in U$ *(in the chosen order)* **do**
  - $\mathcal{F} \leftarrow \{ C \in K \mid v$ is adjacent to every $u \in C \}$ `// candidates`
  - **if** $\mathcal{F} = \emptyset$ **then**
    - create new clique $C_{\text{new}} \leftarrow \{v\}$
    - Add C_{new} to K
    - $W(C_{\text{new}}) \leftarrow \omega(v)$
  - **else**
    - add v to $C^\star \in \mathcal{F}$ with largest weight
  - **end**
- **end**
- **return** K and $\sum_{C \in K} W(C)$

KaMIS Clique Cover

The calculation of clique cover is shown in the KaMIS Clique Cover (KCC) Algorithm. This function returns the upper bound computed by KaMIS and the KaMIS clique cover, since we use it later for clique cover relaxation.

The KCC algorithm iterates over all nodes that have been previously sorted in descending order by node weight or degree on equality. If cliques have already been found that can be expanded by the current node, the node is added to the clique with the highest weight. If this is not the case, the algorithm creates a new clique that only contains the current node. The new clique receives the weight of its node. After the algorithm has traversed all nodes, it returns the clique cover and the sum of the weights of all cliques in the cover. This sum yields the upper bound for the maximum independent set.

KaMIS Clique Cover Relaxation

Although the KCC algorithm returns a clique cover, we do not use it immediately for our clique cover relaxation because it does not include all edges. Therefore, the missing edges are added. This clique cover is passed together with the graph $G = (V, E, \omega)$ to the solver for the clique cover relaxation. This solver is referred to as `CliqueCoverRelaxation` in all subsequent sections. This class builds an LP model using the GNU Linear Programming Kit (GLPK) [10] or Gurobi [17]. Creating an LP model for the clique cover relaxation requires the transformation of the clique cover into a constraint matrix. An example of this conversion is shown in Figure 4.3. The upper bound for the MWIS of graph G is obtained by triggering the LP model to find a solution. In the case of GLPK, the *primal simplex algorithm* is executed. Gurobi runs its standard solver, the *concurrent optimizer*, which executes multiple methods in parallel [16]. The resulting algorithm is called KaMIS Clique Cover Relaxation (KCCR).

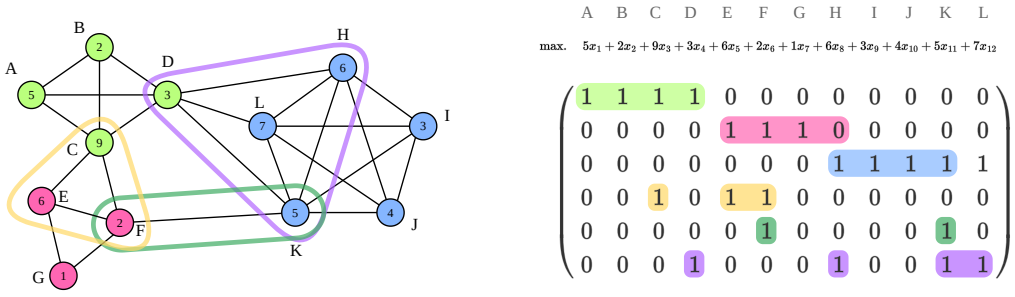

Figure 4.3: Transformation of an exhaustive clique cover (left) to an MWIS objective function and constraint matrix (right). The values within the nodes represent the node weights, while the letters are the node identifiers. The colors indicate cliques.

4.2.2 Exhaustive Clique Cover Relaxation

In this section, we consider the implementation of the Exhaustive Clique Cover Relaxation (ECCR) Algorithm (2). Since this algorithm uses the same `CliqueCoverRelaxation` class described in the previous section, the focus of this section is on the computation of an exhaustive clique cover for an arbitrary connected graph.

Exhaustive Clique Cover Computation

The computation of the exhaustive clique cover is split into two steps. The first one is displayed by the `ComputeUniqueMaximalCliques` function of the ECCR algorithm. This function returns a clique cover that has to be extended to an exhaustive clique cover in the next step. We choose this order due to the properties we want the exhaustive clique cover to fulfill: Often, people desire to compute clique covers that consist of as few cliques as possible, for example, because one can obtain an upper bound for the MIS based on the number of cliques. However, we formulate an LP with the clique cover, and adding further constraints to the LP can only lower our upper bound. Accordingly, we focus in the first step on computing as many unique and maximal cliques as possible, expecting that this will result in a valid exhaustive clique cover. But since we cannot guarantee this, afterwards, all uncovered edges are added to the clique cover, which converts it to an exhaustive clique cover.

Next, the behaviour of the `ComputeUniqueMaximalCliques` function is explained. For a better understanding of the single steps, they are depicted for a single clique in Figure 4.4. The goal of the function is to find a certain number of *cliques per vertex*. This is why there is the CPV parameter that defines a clique limit for each node. To find at most $|V| \cdot CPV$ many cliques, the procedure iterates over all $v \in V$ and shuffles all neighbors of the current node v. To ensure that we obtain unique cliques, a node $u \in \text{shuffled } N(v)$ is detected, that does not share a common clique with v. Both nodes $\{v, u\}$ build the base for the new clique C_{new}. Now only the nodes of the cut $N(v) \cap N(u)$ can be potentially added to C_{new}. But since we want to avoid computing the cut, we consider all nodes $w \in N(v)$ and add them to C_{new} if they are adjacent to all nodes in C_{new}. Checking all $w \in N(v)$ also ensures that we obtain maximal cliques. After all w were considered, the algorithm tries to find a new clique $\{v, u\}$ with the same v but different u. If there is no further u fulfilling the requirements or if CPV cliques were found for v, the next $v \in V$ is considered.

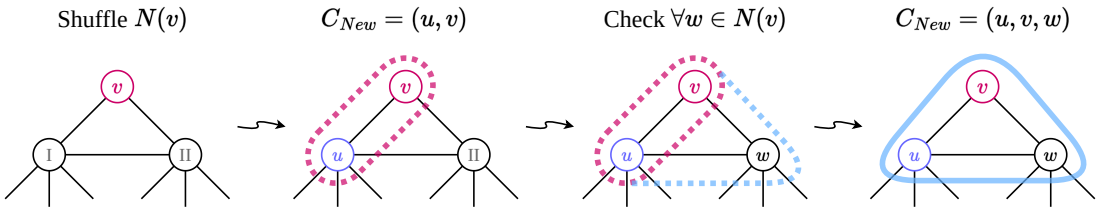

Figure 4.4: Single steps of the clique computation for a node v.

Algorithm 2: Exhaustive Clique Cover Relaxation (ECCR)

Input: Graph $G = (V, E, \omega)$
Input: CPV: Number of **c**liques to find **p**er **v**ertex
Output: Upper Bound for $\alpha_w(G)$

Function `ECCR` ($G = (V, E, \omega)$, *CPV:* $\mathbb{N}^+$)**:**
- $upper\_bound \leftarrow 0$
- $components \leftarrow$ `ConnectedComponents`(G)
- **foreach** $c \in components$ **do**
  - $cliques \leftarrow$ `ComputeUniqueMaximalCliques`(c)
  - **foreach** $e \in E$ **do**
    - **if** e not covered by $cliques$ **then** add e to $cliques$
  - **end**
  - $lp \leftarrow$ `CliqueCoverRelaxation`(G, $cliques$)
  - $upper\_bound \mathrel{+}= \lfloor lp.solve() \rfloor$
- **end**
- **return** $upper\_bound$

Input: Connected graph $G = (V, E, \omega)$ and CPV
Output: Set of cliques

Function `ComputeUniqueMaximalCliques` ($G = (V, E, \omega)$, *CPV:* $\mathbb{N}^+$)**:**
- **if** $|V|$ == 1 **then return** V
- $cliques \leftarrow \emptyset$
- **foreach** $v \in V$ **do**
  - $N(v) \leftarrow$ randomly shuffled neighbors of v
  - **for** $k \leftarrow 1$ **to** CPV **do**
    - Find $u \in N(v)$ s.t. $\deg(u) > 1$ *and* $u \notin$ any clique(v)
    - **if** u *not exists* **then**
      - **break**
    - **end**
    - Create clique $C \leftarrow \{v, u\}$
    - **foreach** $w \in N(v)$ **do**
      - **if** w is adjacent to all nodes in C **then**
        - Add w to C
      - **end**
    - **end**
    - Add C to $cliques$
  - **end**
- **end**
- **return** $cliques$

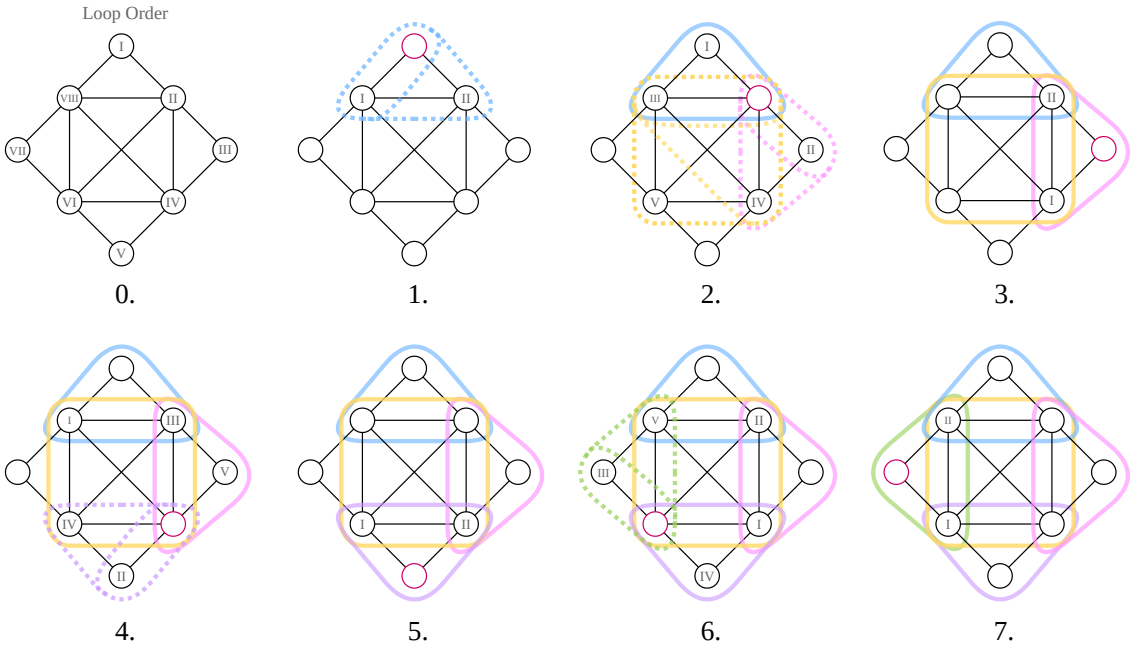

Figure 4.5: Application of the `ComputeUniqueMaximalClique` function to a graph, resulting in a maximum exhaustive clique cover. Roman numerals indicate the order in which the vertices are considered by the algorithm. Cliques are denoted by colored borders.

Computation Example

Applying the described clique computation to the graph in the example Figure 4.5, we obtain the iteration order depicted in step 0. In the other steps, the Roman numbering represents the ordering of $N(v)$ after the shuffling. The dashed surrounding of the nodes represents the clique. For each clique, there is also a dashed surrounding of only two nodes. Those two nodes are the basis of C_{new} before expanding. In step 2 of the figure, you can observe that two cliques were computed. This is only possible if the CPV parameter is larger than one. Also, the pink clique is computed first, because the node II is ordered before node III, which is part of the basis of the yellow clique. After Step 7, all nodes were considered, and the algorithm terminates.

Upper Bound Calculation

Since we are currently only able to compute upper bounds for connected graphs using the exhaustive clique cover and the clique cover relaxation, we split the graph into its connected components. After an upper bound was produced, it is added to the overall upper bound of the graph. A little improvement of the bound is achieved by flooring the value of the connected component. This is allowed because the node weights must be integers. So the MWIS must also be an integer.

Limits of the Clique Cover Computation

Even if it is possible to compute a maximum exhaustive clique cover with this algorithm (see Figure 4.5), there is no guarantee that we receive one. This can be due to a CPV value that is too small or bad luck during shuffling or iterating over V. An example for such a case is depicted in Figure 4.6. There, the algorithm only finds three cliques for any $CPV \geq 1$, because the outermost vertices are considered first. After each vertex shares a clique with every other vertex, the algorithm cannot find any further cliques. Therefore, the clique containing the yellow vertices is not found.

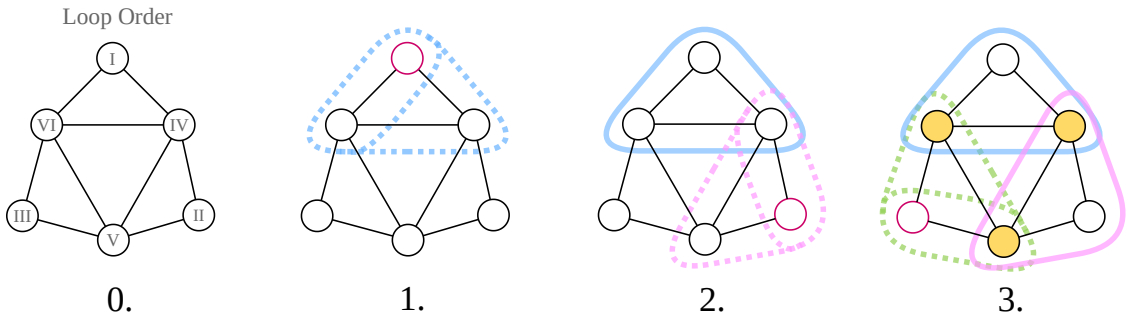

Figure 4.6: Application of the `ComputeUniqueMaximalClique` function to a graph, not resulting in a maximum exhaustive clique cover. The nodes mark those for which the algorithm is currently trying to find cliques. The clique containing the yellow nodes is not found.

4.2.3 Iterative Exhaustive Clique Cover Relaxation

This subsection comprises the extension of the ECCR algorithm to an iterative version. The main innovation of the Iterative Exhaustive Clique Cover Relaxation (I-ECCR) is the extension of the clique cover in each iteration. This also means that the clique cover relaxation must be solved multiple times. To avoid very large LP running times, only a custom subset of the expanded clique cover is used. The next two paragraphs describe the implementation of the functions `IterativeECCR` (3) and `ExtendCliqueCover` (4) which constitute the I-ECCR algorithm (3).

IterativeECCR Function

Compared to the pseudo code of the ECCR (2) algorithm, the I-ECCR (3) algorithm contains a few more variables. Since those variables rule the behaviour of the algorithm, they will be described first:

In addition to the $cliques$ variable, there is also a $trash$ variable. As already mentioned, the algorithm splits the exhaustive clique cover into a subset to save time on solving the LP. This subset is stored in $cliques$, while the rest of the cover is preserved in $trash$. This backup is necessary to avoid recomputing the same cliques in each iteration. Also, it is

possible that a clique in $trash$ becomes important in a future iteration. In such a case, it will be added to $cliques$. Furthermore, there is a variable $improved$. This boolean value indicates whether the clique cover got extended in a way that it certainly will improve the LP solution. This is determined by the solution from the last LP run. So if there is a new clique that would not fulfill the clique constraint $\sum_{u \in C} x_u \leq 1$ using the old solution values, the new LP solution will change.

To iteratively lower the upper bound, the I-ECCR algorithm calls in each iteration the $ExtendCliqueCover$ function (4). If this call results in an improved clique cover (indicated by the $improved$ variable), the cover is transformed to an exhaustive clique cover in the same way as in the non-iterative approach. Using this cover, the upper bound for the current connected component is updated. If the clique cover was not improved, the iteration stops except there are cliques in $trash$ that will change the LP solution. Those cliques will be added to $cliques$, and the algorithm continues. There are two other conditions that stop the iteration, forcing the algorithm to continue with the next component. The first one is a missing improvement of the bound, despite the clique cover being marked as $improved$. This can be the case because the bound is floored after each LP run, like in the ECCR algorithm. The second condition only allows a certain number of iterations for each component. This value is defined by the input parameter I.

ExtendCliqueCover Function

The differences between the `ComputeUniqueMaximalCliques` (2) function and the `ExtendCliqueCover` (4) function are highlighted in red. Thereby, it is easy to recognize that the core of the clique computation did not change. Since the new algorithm extends the clique cover in-place, the return value changed. Instead of returning a clique cover, the new algorithm returns the $improved$ value discussed in the previous paragraph. Accordingly, the $improvement$ variable gets true if there is a clique added to the cover, which certainly will change the LP solution. One more difference is the classification of a clique as useful or redundant. Useful cliques are added to $cliques$, and redundant cliques move to the $trash$. It is important to note that not only cliques are useful, which certainly impact the LP solution. Instead, all cliques that fulfill the constraint $\sum_{u \in C} x_u \leq 0{,}7$ are added to the cover. The *threshold* of 0,7 was picked without good evidence, but a short test showed that the upper bounds get worse by using a too large value.

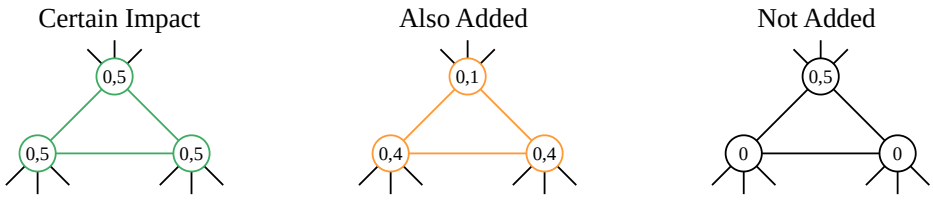

Figure 4.7: Three cliques with associated vertex solution values from last LP run. Only the left clique will certainly impact the new LP solution, but the middle one will also be added.

Algorithm 3: Iterative Exhaustive Clique Cover Relaxation (I-ECCR)

Input: Graph $G = (V, E, \omega)$
Input: CPV: Number of **c**liques to find **p**er **v**ertex
Input: I: Maximal number of **i**terations
Input: $Timeout$: Omitted from the code for better readability
Output: Upper Bound for $\alpha_w(G)$

Function `IterativeECCR` ($G = (V, E, \omega)$, *CPV:* $\mathbb{N}^+$, *I:* $\mathbb{N}^+$)**:**

```
upper_bound ← 0
components ← ConnectedComponents(G)
foreach c ∈ components do
    component_upper_bound ← 0
    cliques ← ∅
    trash ← ∅                                   // unused cliques
    lp ← CliqueCoverRelaxation()
    for i ← 1 to I do
        improved ← ExtendCliqueCover(c, cliques, trash, lp)
        if not improved then
            foreach c ∈ trash s.t. lp.solutionImpactGuaranteed(c) do
                Add c to cliques
            end
            if cliques did not increase then
                break
            end
        end
        exhaustive_clique_cover ← AddUncoveredEdges(cliques)
        lp ← CliqueCoverRelaxation(G, exhaustive_clique_cover)
        if component_upper_bound ≤ ⌊lp.solve()⌋ then
            break
        end
        component_upper_bound ← ⌊lp.solve()⌋
    end
    upper_bound += component_upper_bound
end
return upper_bound
```

Algorithm 4: Extend Clique Cover (Auxiliary function for I-ECCR)

Input: Connected graph $G = (V, E, \omega)$ and CPV
Input: $cliques$: Set of cliques that will be used for the LP
Input: $trash$: Set of cliques that are unlikely to affect the LP solution
Input: lp: LP solver instance to check if clique would have affected the solution
Output: Boolean: True if new cliques will improve the LP solution

```
// This function operates inplace on cliques and trash
// Red highlights indicate changes to the ECCR version
```

Function `ExtendCliqueCover` ($G = (V, E, \omega)$, *CPV:* $\mathbb{N}^+$, *cliques, trash, lp*)**:**

- $improved \leftarrow$ False
- **foreach** $v \in V$ **do**
  - $N(v) \leftarrow$ randomly shuffled neighbors of v
  - **for** $k \leftarrow 1$ **to** CPV **do**
    - Find $u \in N(v)$ s.t. $u \notin$ any clique(v) `// also check trash`
    - **if** u *not exists* **then**
      - **break**
    - **end**
    - Create clique $C \leftarrow \{v, u\}$
    - **foreach** $w \in N(v)$ **do**
      - **if** w *is adjacent to all nodes in* C **then**
        - Add w to C
      - **end**
    - **end**
    - **if** *lp.solutionImpactPossible(C)* **then**
      - Add C to $cliques$
    - **else**
      - Add C to $trash$
    - **end**
    - **if** *lp.solutionImpactGuaranteed(C)* **then**
      - $improved \leftarrow$ True
    - **end**
  - **end**
- **end**
- **return** $improved$

CHAPTER 5

Experimental Evaluation

In the following section, we introduce the experimental setup as well as the evaluation method. Before proceeding with the evaluation, we specify the datasets that are used to evaluate the proposed approaches.

5.1 Methodology

All algorithms were implemented in C++ 14 and compiled with g++ (GCC) Version 14.2.1 using the Release mode with the -O3 flag. The experiments were run on a machine with an Intel Core i5-1235U (10 cores, 1,30 GHz to 4,40 GHz) with 12 MB of L3 cache and 16 GB of main memory. The machine was running Manjaro 25.0 with Linux kernel 6.14.0-1. The ECCR and I-ECCR algorithms were run with five different seeds (1 to 5) that are used for the random number generator. Mean values always refer to the arithmetic mean over those seeds. The running time refers to the time spent on calculating the upper bounds, without reading in the graph or performing any logging.

To examine whether we improve the upper bounds with our algorithms, we compare each algorithm with its previous version. Accordingly, we start with the KCC (4.2.1) and KCCR (4.2.1) algorithms and end with the comparison between the ECCR (4.2.2) and I-ECCR (4.2.3). Thereafter, we select the best configuration of each algorithm to provide a benchmark against existing work. The existing work comprises the original KaMIS upper bound (without relaxation) and the two state-of-the-art algorithms, CHILS [15] and the Bregman-Sinkhorn algorithm BSA [18]. CHILS is a heuristic to compute lower bounds, whereas BSA provides a lower bound as well as an upper bound for the MWIS. We run CHILS parallel in 10 threads, calculating 10 concurrent solutions using a time limit of 6 minutes[1]. For the Bregman-Sinkhorn algorithm BSA, we used the same recommended

[1] `CHILS -c 10 -p 10 -t 360`

configuration[2] that Großmann et al. used for their CHILS evaluation [15], also with a time limit of 6 minutes. This configuration produces integer solutions only after reaching 0,1% relative duality gap for the LP relaxation [15]. All upper bounds were rounded down, and all lower bounds were rounded up, since we only use integer weights in the instances.

For the comparison of the algorithms, we use performance profiles [6]. These plots allow a high-level comparison between any number of algorithms by visualizing how far each algorithm's measured value (e.g., running time or solution quality) is from the best one. In the following, the "solution" of an algorithm refers to the measured value, even if it is the running time or another unit. To describe the distance to the best solution, each algorithm is assigned a ratio $\tau_{A,i}$. The value $\tau_{A,i}$ multiplied by the best solution gives the solution of the algorithm A. This ratio is calculated for each instance i of the dataset using all algorithms A. The performance profile plot assigns a piecewise constant function to each algorithm. The x-axis shows a general τ, which is denoted as the *performance ratio*. The corresponding y-value indicates the fraction of instances i for which algorithm A has a $\tau_{A,i}$ that is smaller than or equal to the performance ratio τ. In other words: The performance ratio indicates how many instances can be solved at least as well as with the best instance-specific algorithm when the solution is divided by the performance ratio. Since the amount of $\tau_{A,i}$ that are smaller or equal τ increases with increasing τ, the functions of the performance profiles are non-decreasing.

Considering minimization problems, such as running time or the upper bound of the MWIS, $\tau_{A,i}$ will always be greater than or equal to one, since the best algorithm will have the smallest solution value ($\tau_{A,i} = 1$). For maximization problems like the MWIS, the value will be between one and zero, because by dividing by $\tau_{A,i}$ the worse solutions must increase.

5.2 Instances

Since we compare our results with the BSA results, we use the dataset that Haller et al. republished along with their paper [18]. The dataset is available on their website[3] and consists of two subsets. The first set contains 38 *Amazon Vehicle Routing (AVR)* instances that were originally presented by Dong et al. [7]. These instances are based on real-life long-haul vehicle routing problems and can be downloaded there [4]. The dataset comes with precomputed clique information for all graphs, which is required by BSA. The AVR instances are significantly harder to solve than previous datasets [15] and are by far the largest publicly available MWIS instances [18]. However, we do not use the largest 19 instances, because they are too large for our main memory of 16 GB. The second collection of 21 instances belongs to the *Meta-Segmentation for Cell Detection (MSCD)* dataset. It is

[2] `mwis_json -l temp_cont -B 50 -initial-temperature 0.01 -g 50 -b 100000000 -t [seconds] [instance]`

[3] https://vislearn.github.io/libmpopt/mwis2024/

[4] https://registry.opendata.aws/mwis-vr-instances/

dataset	#instances	#nodes	#edges	#cliques	avg. clique size
AVR-large	17	127k – 882k	43M – 344M	5.4k – 38k	120.2
AVR-medium	14 + 2	10k – 84k	126k – 39M	1.8k – 48k	24.4
AVR-small	5	979 – 14k	2.4k – 44k	805 – 15k	3.2
MSCD	21	5k – 8k	66k – 198k	22k – 36k	8.8

Table 5.1: The BSA dataset. The instances that we do not use for evaluation are shown in gray. Here **#nodes** and **#edges** stand for the number of nodes and edges in the graph, **#cliques** gives the number of clique constraints in the respective ILP representation Haller et al. used to compute the upper bounds, and **avg. clique size** is the average clique size per instance, averaged over the number of instances in the dataset.

derived from semi-automated labeling problems for cell segmentations [23]. The dataset is only publicly available by Haller et al., as the author of [23] did not publish it.

In total, we use 40 instances, 19 from the AVR and 21 from the MSCD dataset. The authors assigned the AVR and MSCD to four groups, describing the size of the instances. This categorization is depicted in Table 5.1, which is an adapted version of Table 1 in the BSA paper [18]. All instances are available once as a graph representation and once as a clique cover representation. Both formats are required because CHILS and our algorithms work with the graph representation, whereas the Bregman-Sinkhorn algorithm requires a precalculated clique cover. Interestingly, the graph representation is provided in a scaled-down version, while the clique cover is unscaled. Actually, it is an advantage that at least the graphs are scaled, since KaMIS can only process nodes up to a certain node weight. In order to calculate the KaMIS upper bounds (with and without LP relaxation), we therefore rely on the scaled graphs. To make the results comparable, we created a scaled version of the clique cover representation. As we figured out that the scaling factors provided by Haller et al. are not precise enough, we extracted the node weights from the graph representation and replaced the corresponding value in the clique cover representation. Nevertheless, the scaling factors from the dataset are inserted in Table A.1 of the Appendix.

5.3 Experiments

In this chapter, we compare our presented algorithms with their respective previous versions. Therefore, we first determine a good configuration for the new algorithm that will be used for the evaluation. The concluding comparison with existing work follows in a separate chapter.

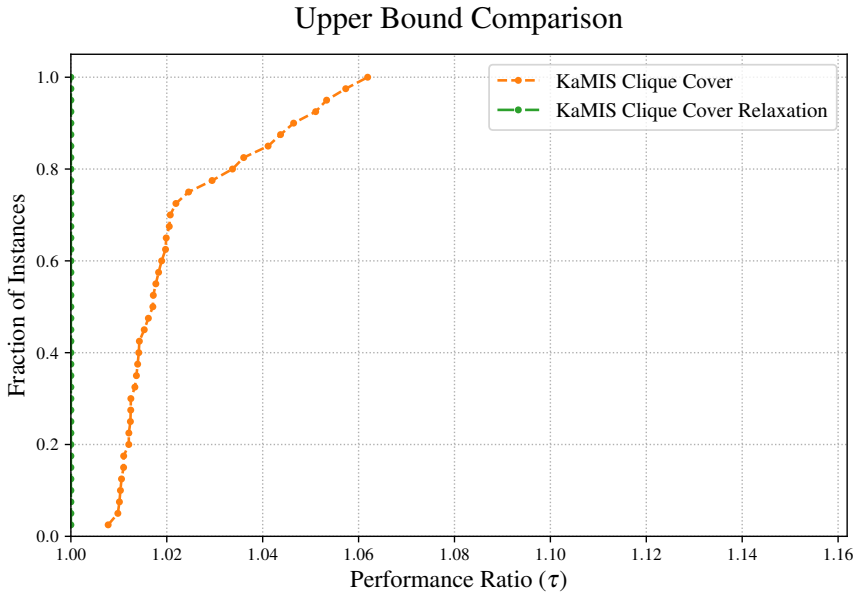

Figure 5.1: The original KaMIS Clique Cover upper bound (KCC) vs. KaMIS Clique Cover Relaxation (KCCR).

5.3.1 KaMIS Clique Cover Relaxation

We begin with the changes between the original KaMIS upper bound and our KaMIS Clique Cover Relaxation. As in the following sections, we split the evaluation into two subsections. The first one considers the improvement of the upper bound, and the second one the progress of the running time.

Upper Bound Comparison

The first question we investigated was whether we could improve the upper bound of KaMIS by applying clique cover relaxation to the computed clique cover. The assumption is that the results with LP relaxation should be at least as good as those without, since we use the same clique cover but try to extract more information from it by adding edge constraints. To check this, we compare the results of both algorithms described in Section 4.2.1. To obtain the original KaMIS upper bound, we modify the code of KaMIS version 3.0 so that no reductions and MWIS computations are performed, but only the pruning upper bound is returned. In Figure 5.1, we present a performance profile comparing the produced upper bounds by the original KaMIS Clique Cover (KCC) and by the KaMIS Clique Cover Relaxation (KCCR). The profile shows that we are able to improve the bound *on each instance*. The KCCR achieves on 75% of the instances at most an upper bound reduction of 2,396%. Overall, it is able to produce reductions up to 5,828%. The worst upper bound of the KCCR is a bound that is 0,771% smaller than the original KaMIS bound.

Observation 1

By adding edge constraints to the KaMIS clique cover and performing the clique cover relaxation, we can generate upper bounds that are up to 5,828% smaller than the original ones. Additionally, the KaMIS Clique Cover Relaxation algorithm *always* produces better results than KaMIS without LP relaxation.

Time Comparison

The next evaluation step covers the running time of the algorithms. First, we compare two variants of the KCCR, one using GLPK, the other one Gurobi. Afterwards, we pick the better one and compare its running time with the KaMIS cover without LP relaxation. Nevertheless, the KCC algorithm should be even faster, since it does not run any LP.

The running time shift between Gurobi and GLPK is shown in the performance profile of Figure 5.2. The Gurobi implementation is up to a factor of 42 times faster than the GLPK implementation. Since GLPK cannot compete with Gurobi on any instance, we use only Gurobi for the upcoming experiments. But even if Gurobi is always faster, it never matches the KaMIS running time without LP. As displayed in the plot of Figure 5.3, the smallest running time gap amounts to a factor of 1,858. Also, 65% of the instances are solved within a maximum time difference of a factor 7,788. This percentage is equal to the sum of all MSCD and AVR_small instances, which are in general the smaller. Since the solution time for the remaining instances increases considerably, this likely means that larger instances take longer to solve. According to this explanation, the running time ratio for the instances from AVR_large would deteriorate even further. However, the performance profile does not

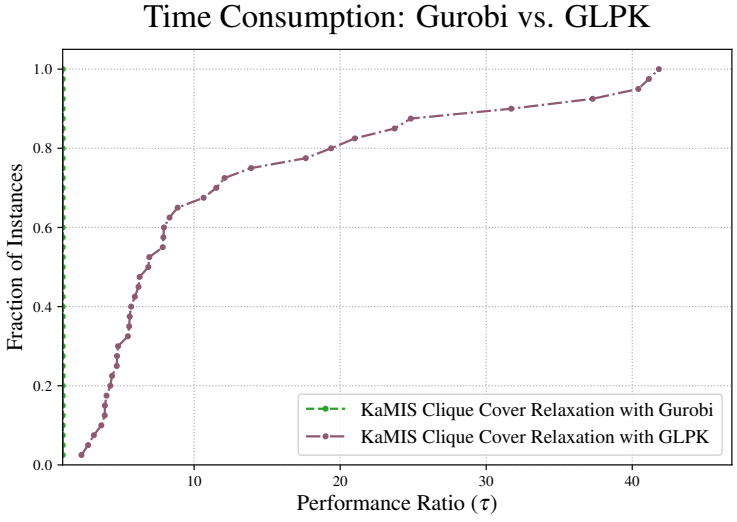

Figure 5.2: Runtime between KCCR with Gurobi and KCCR with GLPK.

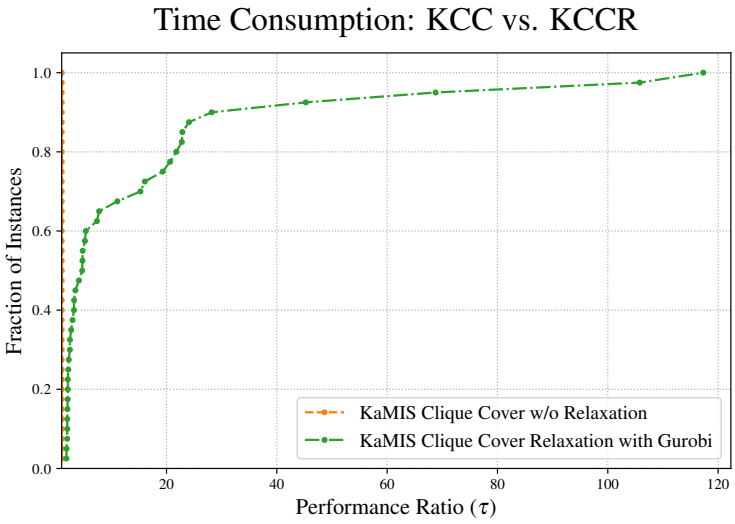

Figure 5.3: Runtime between KCC and KCCR with Gurobi.

contain an exact mapping between performance ratio and instance. Concrete solving 80% of the graphs requires about 20x more time, and the four worst instances need between 45x and 118x the time of the version without LP. But even if a factor of 118 seems very bad, this means in practice that we only required 13,813 seconds to solve the hardest instance using Gurobi. This is because only once the KaMIS clique cover computation took more than half a second. All times can be seen in Table A.3 in the Appendix.

Observation 2

The Gurobi implementation is significantly faster than those using GLPK. Comparing KaMIS with Gurobi to the running time of KaMIS without LP reveals a nonlinear relationship. The Gurobi implementation requires 1,858 x to 118 x more time, with the longest running time being 13,813 seconds.

5.3.2 Exhaustive Clique Cover Relaxation

Since the Exhaustive Clique Cover Relaxation (ECCR) algorithm can run with different CPV (cliques per vertex) values, the first experiment detects a good configuration for this parameter. Afterwards, we pick the best one to compare the solution quality and running time with the KaMIS Clique Cover Relaxation (KCCR).

Best CPV Configuration

We expect a smaller upper bound with an increased CPV value, as it should control how many cliques are found in total. To verify this hypothesis, we not only need to prove that

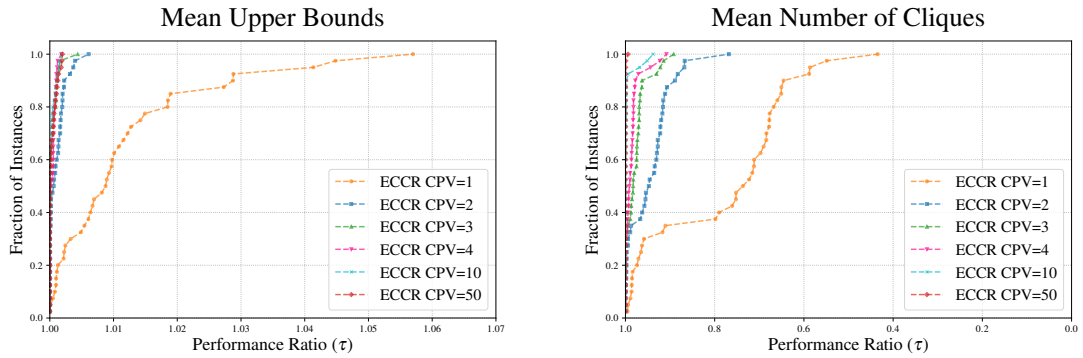

Figure 5.4: Comparison of ECCR upper bound for different CPV (left) and comparison of clique cover size for different CPV (right). For each algorithm configuration, the mean result of seeds 1 to 5 is used.

there is a connection between bound and CPV, but also that there is a positive relation between CPV and the real amount of cliques found. Therefore, we provide two performance profiles in Figure 5.4. On the left side, the mean upper bound over five seeds is plotted for six CPV configurations. We decided to display a selected subset of all CPV values because we want to improve readability. Furthermore, no fundamental changes occurred to the upper bounds after a value of $CPV = 4$. Therefore, $CPV = 50$ is the largest parameter considered in this evaluation. On the right side of Figure 5.4, the ratio of the number of cliques is presented for the same algorithm parametrization. This profile is adjusted for maximization problem, because we want more cliques.

Starting with the right plot, we can see that indeed the number of cliques rises with the CPV value. However, the cliques per vertex value enables the algorithm to find n times more cliques, comparing $CPV = n$ with $CPV = 1$. With regard to the performance profile, this is never the case for $n > 2$. Instead, we observe that the fraction of instances, those clique cover *cannot* be extended by CPV increase, goes up with each increment of the parameter. Consequently, there is a limit to the number of cliques that the algorithm can find. This bound exists due to these two reasons:

1. Obviously, the number of unique maximal cliques is limited by the graph. But even if a graph has enough cliques, it can contain vertices that are part of less than n maximal and unique cliques.

2. There is no guarantee that the ECCR algorithm finds all cliques of a vertex. An example for this is shown in Figure 4.6 of the implementation chapter.

Since there is only a small clique number increase on the four instances for $CPV = 10$ to 50, it is not surprising that we do not get an upper bound improvement for these values. But especially on the lower CPV, we have the expected progress. There, the upper bound decrease fits very well with the corresponding number of cliques that were used for the clique cover relaxation. The most significant improvement occurs between $CPV = 1$ and

$CPV = 2$. Whereas in the first variant the worst upper bound is 5,699% larger than that of the best solution found, the second variant only produces bounds up to 0,609% larger. On the other hand, there is no perfect relation between the number of cliques and solution size. This is well seen on the $CPV = 4$ and $CPV = 20$. Latter produces at almost all instances the largest clique covers, but its overall solution quality is not significantly better than those of $CPV = 4$, even if this variant has the noticeable smaller clique covers. Also, there seems to be some noise, because there is no algorithm configuration that performs on each instance at least as good as the others. This noise comes from the CPV parameter and the random shuffling of the neighbors in the clique cover computation. Even if all algorithms use the same seed, the CPV parameter decides when a clique may be created. For example, we could have two adjacent vertices v and u, where one algorithm configuration might create the clique $\{v, u, x\}$ from v. The other one, however, may have to leave v too early due to CPV, so this clique is not found. Instead, it creates the clique $\{u, v, y\}$ from u, because $N(u)$ is shuffled in a way that y comes before x. If both algorithms also have bad luck with x and y, it can happen that we receive two different clique covers. This is also the reason why it is possible to receive a larger clique cover with a smaller CPV (see Table A.4 in the Appendix). Based on the performance profile, the noise level can lead up to 0,198%[5] worse upper bounds. In the following sections, we refer to the best ECCR configuration as that with $CPV = 50$. Since there is no one that outperforms all others, this is justified by its largest size of the clique cover. Especially on larger graphs with an average degree larger than 50, we expect that this algorithm would produce better results than the others.

Observation 3

Increasing the CPV value leads to better upper bounds, as long as the algorithm can find significantly more cliques. However, the algorithm with the highest CPV value does not always compute the best solution, even if it generally produces the largest clique cover. This is due to random noise, which affected the solution size by up to 0,2%.

Upper Bound Comparison

To assess whether the exhaustive clique cover from ECCR leads to an improvement over the KaMIS clique cover, we compare the best and worst ECCR configurations with the KaMIS Clique Cover Relaxation. Therefore, we provide a performance profile in Figure 5.5 that comprises the upper bounds of the KCCR, the worst ECCR with $CPV = 1$, and ECCR with $CPV = 50$. As already determined, the 50 configuration is better than the 1 configuration and results in up to 1,122% smaller bounds compared to it. However, even the weaker ECCR version already produces way better solutions for all instances than the KCCR algorithm. While the smallest improvement is a reduction by 10,159%, the best

[5] We used the largest τ value of ECCR with $CPV = 4, 10, 20$ and 50

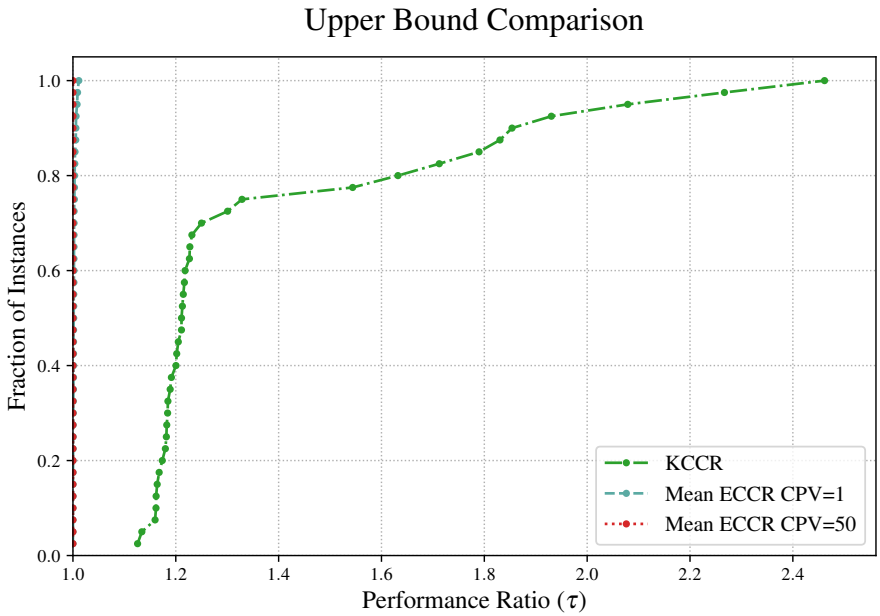

Figure 5.5: KaMIS Cliques Cover Relaxation compared to Exhaustive Clique Cover Relaxation (CCLP) with $CPV = 1$ and $CPV = 50$. For the ECCR configurations, the mean result of the seeds 1 to 5 is used.

does more than half the size of the upper bound by 58,915%. If we compare KCCR and ECCR with $CPV = 50$, we achieve a peak improvement of 59,376%. However, 70 percent of all instances reported only up to 20,002% smaller bounds, with a least reduction of 11,167%. However, a small improvement does not necessarily mean that the algorithm did not perform well on the instance, since it is possible that the previous version already computed good upper bounds. For a comparison with the CHILS lower bound, which gives an impression of the real quality of the upper bounds, see Section 5.4 Comparison With Existing Work. There, we also compare the ECCR algorithm with the original KaMIS upper bounds.

Observation 4

The solutions from the exhaustive clique cover calculated by the ECCR algorithm outperform by far *all* the solutions generated by the KCCR algorithm. This is even the case for the cover that contains at most one clique per vertex ($CPV = 1$). Accordingly, the improvements from KCCR to ECCR, ranging from 11,167% to 59,376%, exceed the best improvement of 5,828% that we achieved by applying clique cover relaxation to the KaMIS clique cover.

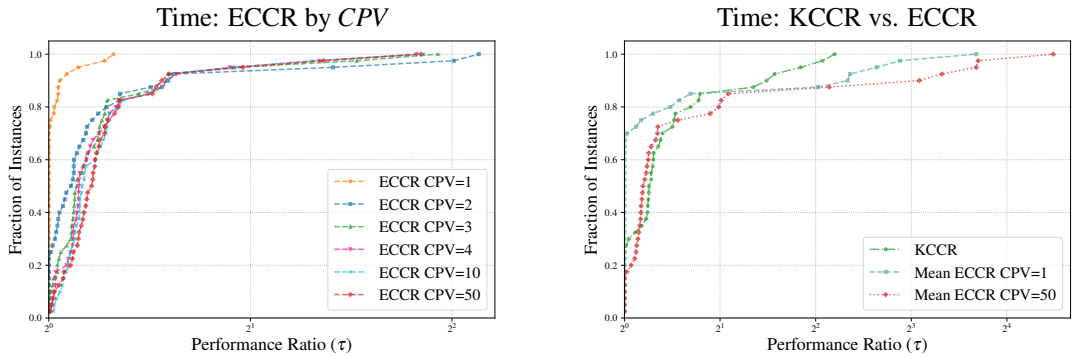

Figure 5.6: Time comparison of ECCR Gurobi with different CPV values (left) and KCCR (right)

Time Comparison

To prove that the algorithm configuration with the smallest resulting exhaustive clique cover has, in general, the shortest running time, we provide the left plot of Figure 5.6. This plot reveals that the running time increases in general with increasing CPV. In Section 5.3.2, we determined that there is, in general, also a positive relation between this parameter and the clique cover size. Consequently, there is also the assumed relationship between the CPV value and running time. Nevertheless, in the performance profile, you can observe exceptions to this rule. But since the running time depends much on the LP solver, we will not attempt to explain these exceptions. Instead, we compare the running time of the worst ($CPV = 50$) and best ($CPV = 1$) ECCR config with the running time of the KCCR algorithm. Therefore, we also provide a performance profile on the right side of Figure 5.6, which shows the mean running time over five seeds. Since the KaMIS Clique Cover Relaxation uses a clique cover, which was originally designed to be as small as possible, we would expect that it outperforms the ECCR algorithm. Contrary to this assumption, this is not the case. Instead, the $CPV = 1$ configuration is the fastest on 60% of the instances and the KCCR algorithm only on 25%. However, this comparison is not completely fair, because we run the KCCR only once, while we use the mean running time for the ECCR. Interestingly, the $CPV = 50$ config is also on 15% of the instances faster than the other two. This could be due to the solving strategy of Gurobi, but also due to measurement inaccuracy. Nevertheless, the largest running time differences happen for the $CPV = 50$ variant. It is the only one that achieves running times up to 22,387 longer than the best variant. With respect to Table A.3, we can find out that the four instances that need the most time for $CPV = 50$ have a performance ratio of 9,997 or larger. Considering Table A.1, we see that these four instances are part of the AVR-medium group, which represents the largest graphs in our dataset. Exactly the same instances also take the most time on the $CPV = 1$ algorithm, which is not the case for the KCCR algorithm. Consequently, explicitly some large graphs are very hard to solve for the ECCR algorithm but not for the KaMIS Clique Cover Relaxation. However, the KCCR algorithm also took between 1,186 and 9,674 seconds for these four hard instances. Summarized this leads for the ECCR to running times up to 110,85 seconds.

Observation 5

The KaMIS Clique Cover Relaxation does unlike expected not outperform both ECCR configurations. But even if ECCR with $CPV = 1$ is the fastest on 60% of the instances, it performed like the slower $CPV = 50$ variant very badly on four of the largest graphs. The worst running time of 110 seconds was measured not only for the slower configuration, but also for the faster one on the same graph. Consequently, this means that there will be other large graphs where both variants will perform equally badly.

5.3.3 Iterative Exhaustive Clique Cover Relaxation

To compare the Iterative Exhaustive Clique Cover Relaxation with the Exhaustive Clique Cover Relaxation, we run them with the same CPV parametrization (1, 2, 3, 4, 5, 7, 9, 10, 11, 13, 15, 20 and 50) and seeds (1, 2, 3, 4, 5). The difference between the algorithms is the *I* parameter of the new version, which defines the maximal number of iterations. This value is set to 50 for all experiments. So in total, we run the I-ECCR exactly as often as the non-iterative version.

Upper Bound Comparison

In Section 4.2.3, we described the I-ECCR algorithm as an extension of the Exhaustive Clique Cover Relaxation algorithm. To show that the new version actually achieves at least as good results, we focus on the lowest upper bounds calculated across all configurations. The results are composed in the left performance profile of Figure 5.7. Since the I-ECCR is capable of improving its solution up to 50 times, we would expect it to calculate better bounds for most instances. Considering the performance profile, we observe that the I-ECCR always provides the best bound. However, it does not bring the supposed improvement. The non-iterative version also finds the same best solution for 57,5% of the graphs. And even for the other instances, the I-ECCR improvement is very small, with a maximum reduction of the bounds by 0,178%.

From Section 5.3.2, we know that there is a positive relationship between the solution quality and the number of cliques found by the algorithm. In addition, we have seen that for large CPV, there is only a little or no increase in cliques. This can explain both observations from the last performance profile. Since the clique calculation strategy for ECCR and I-ECCR is the same, ECCR can keep up with the iterative version if the CPV value is large enough to find the largest clique cover.

Figure 5.7 also depicts a diagram on the right side in which only small CPV values are used for the iterative algorithm version. This configuration was chosen because small CPV values lead to faster intermediate results. This is advantageous for branch-and-bound algorithms, for example. Note that the used I-ECCR configuration should still find at least as many cliques as ECCR, since it has $I \cdot CPV$ attempts. However, the performance profile

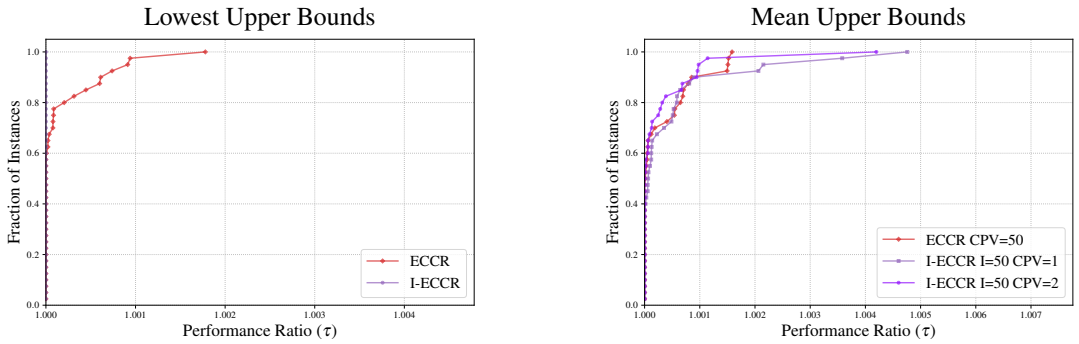

Figure 5.7: I-ECCR vs. ECCR

shows that I-ECCR does not always find the best solution for this configuration. One reason for this is the noise discussed in Section 5.3.2, which was also the reason why $CPV = 10$ solutions can be better than $CPV = 50$ solutions. Another possible reason is that the I-ECCR terminates too early, because there was an iteration where it could not improve the clique cover, even if there was a clique that certainly would affect the LP solution. The latter problem can be fixed by increasing the CPV. Therefore, the iterative $CPV = 2$ variant has smaller performance ratios than the $CPV = 1$ and mostly even produces better results than the ECCR $CPV = 50$ configuration.

While the ECCR algorithm leads to poorer results for small CPV, we wanted to eliminate this problem with the iterative version. Even though the latest result already reveals that there is an enhancement, we provide a performance profile in Figure 5.8 comparing the I-ECCR bounds for different CPV. The plot is the exact counterpart to the performance profile for the ECCR algorithm, which is shown again on the right side of Figure 5.8 for a better overview. Comparing both diagrams, we observe the desired result. The worst upper bound is only 0,476%. larger than the best one. The ECCR had a maximal gap of 5,699%. However, the 0,476% are still double the detected noise.

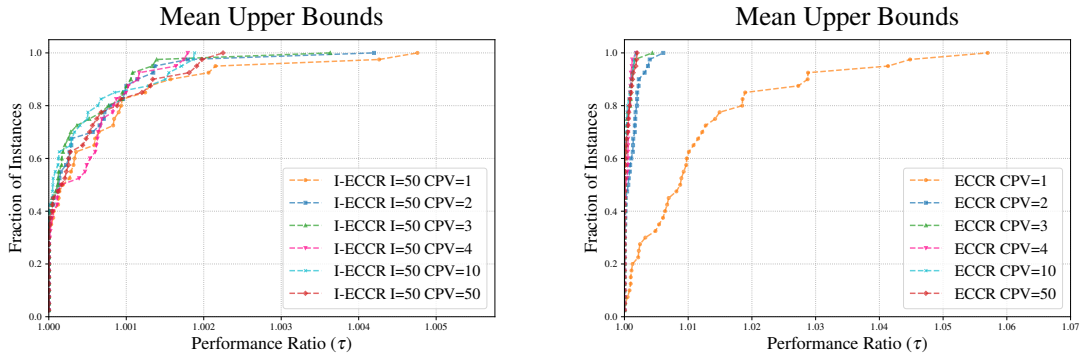

Figure 5.8: I-ECCR (left) and ECCR (right) with different CPV configurations. The mean results of seeds 1 to 5 are used.

Observation 6

The Iterative Exhaustive Clique Cover Relaxation does not bring a notable improvement to the upper bound. There is also for small CPV no guarantee that the bound will reach the best solution that can be calculated with the ECCR. Regardless, the solution quality is largely independent of the CPV parameter if sufficient iterations are used.

Time Comparison

Before we begin with the time comparison between ECCR and its iterative version, we have determined a good I-ECCR configuration regarding the running time. This is done like for the ECCR algorithm. Accordingly, Figure 5.9 presents a performance profile of nine variants with different CPV. We stay on $I = 50$ for all variants, since the final comparison should only contain algorithms with similar solution quality. From the profile, we extracted the configuration with $CPV = 10$ as best. Even if it only provides for 35% of the graphs the fastest solution, there is no other algorithm that beats it on more instances. Also, its performance ratio is the smallest on most of the graphs. Although the following assumption is not based on available data, it is very likely that this variant is the fastest

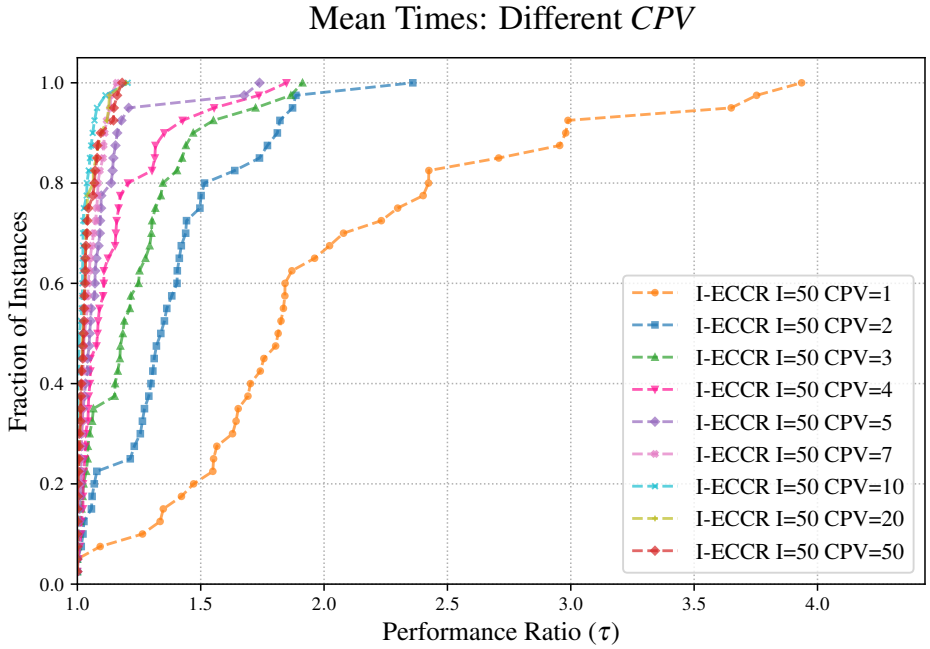

Figure 5.9: Runtime comparison of I-ECCR with different CPV. For the results, the mean time over the seeds 1 to 5 is used.

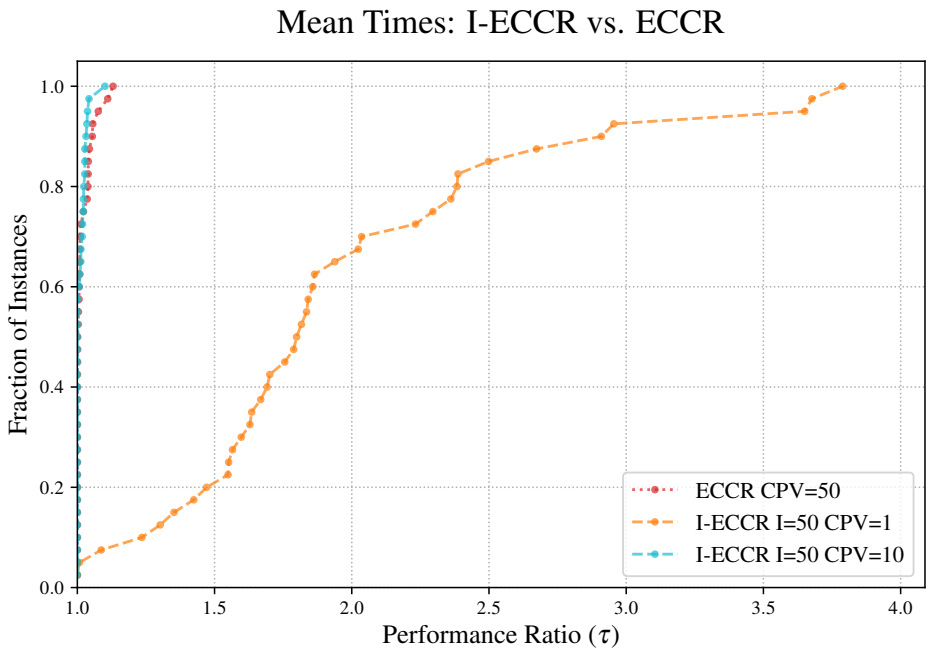

Figure 5.10: I-ECCR vs. ECCR

because it only requires one iteration in most instances and therefore only has to solve the LP once. Because this is not how we want to use the iterative algorithm, we compare ECCR not only with I-ECCR with $CPV = 10$, but also with $CPV = 1$. For the ECCR algorithm, we use the $CPV = 50$ variant due to its good solutions. The performance profile that contains these three algorithm variants is depicted in Figure 5.10. There, it can be seen that the slower I-ECCR variant is significantly outperformed by the other two. Compared to the fastest solution, it takes up to 3,789 times as long. The faster $CPV = 10$ variant, on the other hand, solves, with 55% of the dataset, more instances faster than the ECCR algorithm. In addition, it has a lower average performance ratio, which means that it provides faster results overall. However, the I-ECCR algorithm also has a maximum deceleration of 10,05% to the best solution, which is not particularly lower compared to the 13,013% of the non-iterative algorithm.

Observation 7

Focussing on algorithm configurations that produce very similar solutions, we could find out that the I-ECCR configuration with $CPV = 10$ and $I = 50$ leads to a small improvement in running time compared to the $CPV = 50$ ECCR algorithm.

5.4 Comparison With Existing Work

This section provides an overall comparison of our best algorithm configurations to the original KaMIS upper bound. In addition, we use the state-of-the-art heuristics CHILS [15] and BSA (Bregman-Sinkhorn algorithm) [18] for a final evaluation. CHILS provides lower bounds for the MWIS. Based on these values, we can estimate how close our upper bounds are to the optimal solution size. Even though BSA also returns lower bounds, we only use its upper bounds for a state-of-the-art comparison with our algorithms, because CHILS provides better lower bounds for all instances except four. BSA is run on the precalculated clique cover that was provided by the author. However, we had to scale this part of the data set ourselves (see Section 5.2 Instances). CHILS and BSA were executed with a time limit of 6 minutes.

Bound Comparison

First, we discuss Figure 5.11, containing the bounds of CHILS, BSA, KCC, KCCR, ECCR, and I-ECCR. We choose the ECCR version with $CPV = 50$ and the I-ECCR with $CPV = 10$ and $I = 50$, since these are our recommended configurations to run these algorithms. For both variants, the mean values from seeds 1 to 5 are used. Since CHILS is the only one that provides lower bounds, this algorithm is the "best" for all instances. This has the advantage that the functions of the other algorithms can be directly compared to CHILS.

Again, we see that KCC and KCCR perform very badly. The KCC results are between 1,198 and 2,574 times larger than those of CHILS. If we raise the CHILS solutions by the largest performance ratio of BSA (1,018) so that they are definitely above the optimal solution, the KCC results are still between 17,71% and 152,892% worse than the CHILS. For the KCCR, the outcome is barely better with performance ratios from 1,128 to 2,542. Between ECCR and I-ECCR, it is difficult to see any differences just by looking at the plot. However, this is not the case when comparing BSA and the two algorithms. It can be seen that the BSA function always appears along or to the left of our algorithms. Therefore, it computes on most instances the better upper bounds. BSA solves 80% of the instances with a performance ratio of 1,003, which is by a factor of 6 smaller than its overall worst ratio of 1,018. This means that BSA provides near-optimal solutions for most graphs. Both ECCR and I-ECCR have a ratio of slightly above 1,02 also for a fraction of 80%, which is more than BSA needs for all instances. Finally, our (I)-ECCR configurations have the same (rounded) worst ratio of 1,033. Using the larger unrounded ECCR value, we obtain a solution that is 3,284% larger than its CHILS solution. In contrast, the worst BSA solution is only 1,783% larger, which is about half. However, this does not mean that BSA outperforms our algorithms on all graphs, as shown below.

To give an overview of the best solutions calculated by each algorithm, we created Table 5.2. So now we go back from the (I)-ECCR mean values to the lowest upper bound we found using the configurations from Section 5.3.3. The table also contains the BSA and CHILS bounds, but only the lowest upper bounds are highlighted in bold. The CHILS

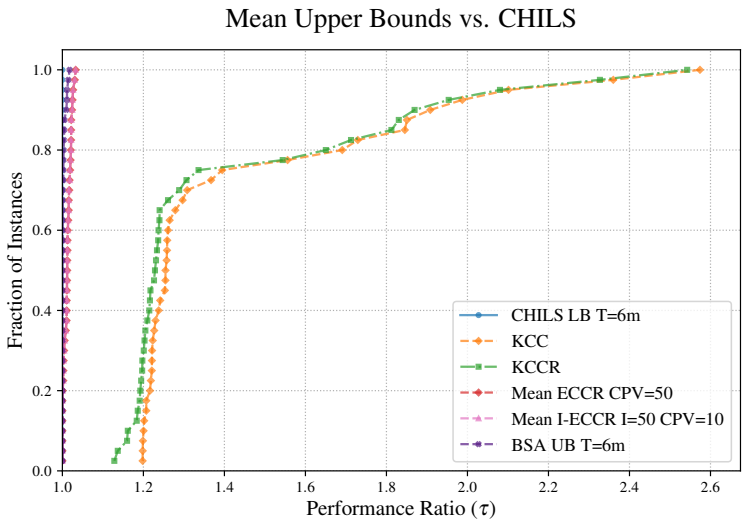

Figure 5.11: Comparison between CHILS lower bound, BSA upper bound, the KaMIS upper bound (KCC), and our best algorithm configuration for each version. The ECCR and I-ECCR are overlapping.

lower bound is marked **green** if it matches the lowest upper bound. From the table, it is easy to read that BSA finds the best solution for all MSCD instances. However, this is not the case for the AVR dataset. There, BSA only computes the best bounds on half of the instances. Interestingly, on all 10 instances, which we solved better than BSA, our ECCR algorithm produced an equally good solution as the iterative version. Since in one case BSA and our algorithm found the same solution, it also provides the best upper bounds on 10 AVR graphs, even though there are only 19 in this dataset. Nevertheless, our algorithms were able to provide the optimal solution for at least five graphs (12,5% of all datasets). This can be proven by the CHILS lower bound, which must have the same value as the upper bound. BSA has not returned a single bound that fulfills this condition.

Observation 8

Even if the BSA outperforms our approach on all MSCD instances, we could find better upper bounds for half of the AVR instances without using the advantages of the precomputed clique covers required by BSA. Additionally, only our algorithms were able to provide solutions, where we can prove that they are optimal. In total, our ECCR and I-ECCR algorithms were able to compute on each instance a close upper bound. The largest deviation from the CHILS lower bound was a 3,284% larger result.

Table 5.2: CHILS lower bounds together with the best upper bounds for the BSA, KCC, KCCR, ECCR, and I-ECCR algorithms. The (I)-ECCR were executed with the configurations described in Section 5.3.3. The lowest upper bound on each instance is highlighted in bold. The CHILS bound is marked **green** if it is equal to the lowest upper bound.

Instance	KCC	KCCR	Best ECCR	Best I-ECCR	BSA	CHILS
MR-D-01	1 863 354 499	1 754 754 382	**1 558 702 057**	**1 558 702 057**	1 558 837 933	1 555 272 030
MR-D-03	1 399 168 695	1 328 329 087	1 082 163 610	1 082 078 096	**1 081 890 341**	1 078 928 116
MR-D-05	1 186 713 710	1 134 033 844	871 634 378	871 566 281	**871 382 459**	868 086 806
MR-D-FN	1 109 245 970	1 062 773 797	799 732 117	799 664 265	**799 435 902**	795 413 712
MR-W-FN	1 359 332 691	1 312 046 268	**1 080 255 273**	**1 080 255 273**	1 080 349 170	**1 080 255 273**
MT-D-01	290 754 796	276 633 077	**238 166 485**	**238 166 485**	238 190 197	**238 166 485**
MT-D-200	326 647 135	323 374 852	**155 491 414**	**155 491 414**	155 524 943	155 448 042
MT-D-FN	308 369 298	305 011 029	**166 634 431**	**166 634 431**	**166 634 431**	166 626 056
MT-W-01	374 175 909	361 976 773	**312 121 568**	**312 121 568**	312 152 337	**312 121 568**
MT-W-200	267 867 951	265 269 369	**154 899 257**	**154 899 257**	154 916 643	154 897 702
MT-W-FN	245 852 556	243 956 023	**158 005 943**	**158 005 943**	158 021 534	**158 005 943**
MW-D-01	572 634 406	541 590 471	**477 467 196**	**477 467 196**	477 566 115	476 437 726
MW-D-20	593 451 405	579 229 940	354 582 923	354 582 923	**354 354 403**	350 959 680
MW-D-40	397 696 097	390 552 652	218 064 368	218 045 481	**217 955 374**	215 499 651
MW-D-FN	307 610 229	302 412 610	156 598 159	156 594 579	**156 561 937**	154 789 973
MW-W-01	1 520 639 965	1 460 570 893	**1 212 205 854**	**1 212 205 854**	1 212 304 599	**1 212 205 854**
MW-W-05	685 920 157	671 992 200	361 983 579	361 970 268	**361 028 951**	359 448 366
MW-W-10	511 884 047	504 737 963	222 271 930	222 271 930	**220 095 070**	216 901 878
MW-W-FN	452 822 276	447 270 740	181 436 575	181 436 575	**179 057 525**	175 920 756
MSCD_000	733 511 697	712 526 475	586 281 125	586 281 125	**579 998 912**	579 941 037
MSCD_001	507 549 567	502 359 258	419 620 169	419 620 169	**412 657 530**	412 621 002
MSCD_002	662 812 674	651 671 964	548 906 023	548 902 864	**542 695 962**	542 641 794
MSCD_003	764 555 528	749 650 503	628 901 353	628 901 353	**623 616 851**	623 246 490
MSCD_004	730 085 676	718 476 642	605 549 959	605 360 111	**600 825 661**	600 237 047
MSCD_005	582 810 239	576 473 437	484 865 497	484 865 497	**476 576 370**	476 528 726
MSCD_006	436 191 231	430 050 231	347 703 648	347 703 648	**341 180 519**	341 098 573
MSCD_007	751 743 946	737 194 040	621 924 548	621 357 294	**616 203 775**	615 643 728
MSCD_008	657 052 392	649 204 576	553 635 953	553 635 953	**547 408 921**	547 354 264
MSCD_009	770 303 501	754 816 853	617 904 321	617 628 507	**611 962 315**	611 905 129
MSCD_010	566 164 897	559 170 497	464 621 556	463 796 312	**456 057 433**	456 011 861
MSCD_011	607 990 485	600 737 327	488 620 373	488 620 373	**485 001 429**	484 605 284
MSCD_012	782 897 268	766 116 540	647 363 476	647 363 476	**642 575 105**	642 511 000
MSCD_013	564 400 939	556 650 339	455 699 149	455 420 989	**449 372 168**	448 876 895
MSCD_014	698 045 278	685 073 796	563 468 466	563 353 713	**553 686 064**	553 634 101
MSCD_015	428 350 477	421 896 204	335 537 677	335 537 677	**328 473 920**	327 468 394
MSCD_016	649 352 601	638 048 350	528 665 091	528 348 364	**525 007 532**	524 650 259
MSCD_017	628 428 927	621 863 390	527 862 717	527 367 116	**520 652 792**	520 600 877
MSCD_018	518 652 004	511 660 252	438 673 826	438 350 078	**429 705 769**	429 662 819
MSCD_019	647 100 082	639 122 660	546 687 116	546 674 987	**540 059 757**	540 004 600
MSCD_020	536 625 031	529 557 465	435 281 292	435 281 292	**428 304 211**	428 261 405

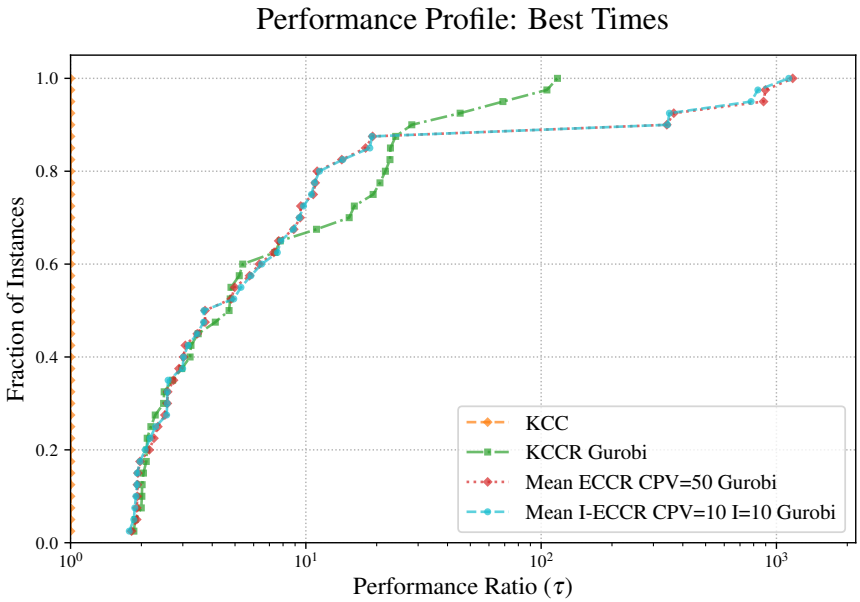

Figure 5.12: Performance profile of the running times KCC, KCCR, and the best ECCR and I-ECCR configuration. For better readability, a log scale is used.

Time Comparison

It does not make much sense to compare the running time between BSA and our algorithms, as BSA does not generate its own clique cover. Analyzing the time consumption of the two clique cover relaxation implementations would also only be a comparison between BSA and Gurobi, which is not our area of interest. Therefore, in the following, we only compare the running time of KCC, the original upper bound calculation from KaMIS, with KCCR, ECCR and I-ECCR.

The performance profile in Figure 5.12 uses a log scale to display the four configurations. Considering the curve shapes, we observe that KCC is the fastest on all instances. This is not surprising, since it is the only algorithm that does not solve any LP. Accordingly, the performance ratios of all other algorithms display a direct relation to KCC. As a result, we see an extreme increase in running time. Each algorithm requires at least 77,674% more time than the KCC. Both (I)-ECCR algorithms show the same largest rounded running time gap with an increase of 117 178,712%, which is more than a thousand times longer. The KCCR algorithm performs better in the worst case, but provides much worse upper bounds.

Observation 9

Applying the clique cover relaxation leads to an immense increase in running time due to the LP that needs to be solved.

CHAPTER 6

Discussion

6.1 Conclusion

In this thesis, we developed and implemented three algorithms to compute good upper bounds for the MAXIMUM WEIGHT INDEPENDENT SET problem. This was achieved by leveraging the clique cover LP relaxation of the MWIS. The key component of our algorithms is the computation of an exhaustive clique cover, a cover that is designed to contain as many cliques as possible. This enabled our Exhaustive Clique Cover Relaxation (ECCR) algorithm to compute much better upper bounds than the pruning bound from the KaMIS Clique Cover (KCC). We also compared the ECCR to our KaMIS Clique Cover Relaxation (KCCR), finding out that our cover leads to significantly smaller upper bounds. Finally, we extended the ECCR to an iterative version. From our algorithms, this Iterative Exhaustive Clique Cover Relaxation (I-ECCR) found the best solution for all graphs in the dataset. The Bregman-Sinkhorn algorithm (BSA) computed a better upper bound on 75% of the instances, containing all MSCD instances. However, this state-of-the-art algorithm takes advantage of a precomputed clique cover, whereas our algorithm calculates its own. Furthermore, we were able to prove that our ECCR and I-ECCR algorithms were able to find the optimal solution for 12,5% of the graphs. All of those are part of the AVR dataset. Although BSA had a time limit of more than twice the time required by our algorithm, we were unable to show optimality for a single instance. However, all of the presented algorithms, that solve an LP, have a massive running time increase compared to the original KaMIS clique cover bound.

6.2 Future Work

Our approach offers multiple opportunities for future work. First, the evaluation of the I-ECCR can be significantly expanded. The use of the threshold value of 0,7 is not supported

by evidence that this value leads to the best solution or running time, or to a balance between both. Furthermore, we provide less information about the quality of the solution, in terms of the graph size or the number of iterations involved. On the other side, it would be interesting how our algorithms perform on reduced graphs (graphs where reduction rules are applied [12, 14, 20]). Another area of interest is whether the use of our algorithms as upper pruning bounds of branch-and-bound algorithms can achieve a performance increase in exact state-of-the-art solvers for MWIS.

For further improvements to the upper bound, we see two areas with potential. Since we are not limited by the number of cliques used for the clique cover relaxation, we can extend the clique cover computation. In the best case, the algorithm should be capable of finding a maximum exhaustive clique cover that contains all cliques of the graph. For instance, we considered the introduction of a depth parameter that determines how often an alternative clique for each clique-base $C_{New} = \{v, u\}$ (see Figure 4.4) should be found. Another, less concrete, possibility for improvement could arise from examining the solution values of the decision variables from the LP. Due to the good upper bounds, we saw in our private experiments that most of the values have near-integer values. However, it is interesting if it is possible to detect weak points, so that we can focus on them.

Finally, we present some suggestions for optimizing our running times. One approach could be to determine a smaller but still good basis clique cover for the first iteration of the I-ECCR algorithm. Here, it could help to observe what the BSA clique cover makes better than our $CPV = 1$ cover. Since the LP is the most time-consuming part, our iterative algorithm loses a lot of time by solving the entire LP at each iteration. Utilizing the old LP could therefore lead to improvements. Incorporating BSA for the clique cover relaxation instead of using a generic LP solver should also lead to at least faster intermediate results, as the algorithm outputs each improvement while solving the relaxation.

APPENDIX A

Appendix

Implementation Details

In the function `ComputeUniqueMaximalCliques` (2), there is the condition "w is adjacent to all nodes in C". This is verified by checking if C is a subset of $N(w)$. Since the clique representation as a set is only for the pseudo code, we sort C_{new} and $N(w)$. Afterwards, it is only necessary to iterate once over both arrays. For the same reason, the condition should also contain that $w \neq u$ to avoid adding u twice. In addition, there is a check if $\deg(w) \geq |C|$ before checking the adjacency. If this is not the case, w cannot be part of C and is discarded.

In the same function, the node u must fulfill the requirement that it does not share a common clique with v. This is figured out with the help of a clique mapping. Each node has an associated list of clique IDs it belongs to. If the two lists of u and v contain an identical clique ID, u is discarded. Also, all nodes with $deg(u) \leq 1$ are discarded, because edges will be added afterwards either way.

In the function `ExtendCliqueCover` (4), there is the case check for components with $|V| = 1$ missing. This is due to the fact that the implementation of the ECCR and I-ECCR algorithms returns the node weight of $v \in V$ as the weight of the upper bound for components that only contain a single node. Furthermore, the $lp.solutionImpactGuaranteed(c)$ and $lp.solutionImpactGuaranteed(c)$ methods always return true if the lp was never solved before. In addition, we only round down the upper bounds of the connected components if they are more than 0.0001 smaller than their rounded-up number. Otherwise, we round up the bound. This is reasoned by rounding errors produced by Gurobi. Those were verified by upper bounds that were exactly one weight smaller than the CHILS lower bound, if we run our program without the check.

Further Results

Table A.1: Information about the dataset used for the evaluation.

Instance	Group	Vertices	Edges	Scaling Factor
MR-D-01	AVR-SMALL	14 058	44 181	0,919 475 823 012 533 2
MW-D-01	AVR-SMALL	3 988	13 556	1,0
MW-W-01	AVR-SMALL	3 079	22 664	0,954 263 229 536 075 6
MT-D-01	AVR-SMALL	979	3 125	1,0
MT-W-01	AVR-SMALL	1 006	2 411	1,0
MW-W-10	AVR-MEDIUM	18 023	1 451 813	0,161 723 566 016 500 2
MW-W-05	AVR-MEDIUM	10 790	485 261	0,270 524 048 418 215 9
MW-D-FN	AVR-MEDIUM	47 504	4 017 196	0,284 721 744 226 440 3
MW-D-40	AVR-MEDIUM	33 563	1 879 303	0,401 095 818 612 834 3
MW-D-20	AVR-MEDIUM	20 054	606 318	0,666 688 273 998 955 1
MT-W-200	AVR-MEDIUM	12 320	515 871	0,403 325 031 109 666 1
MT-W-FN	AVR-MEDIUM	12 320	553 895	0,404 242 185 658 572 2
MR-D-03	AVR-MEDIUM	21 499	130 508	0,613 418 014 039 846 7
MT-D-FN	AVR-MEDIUM	10 880	604 041	0,572 860 550 121 975 2
MT-D-200	AVR-MEDIUM	10 880	505 359	0,541 468 192 878 346 1
MR-W-FN	AVR-MEDIUM	15 639	126 800	0,200 536 230 867 585 1
MR-D-FN	AVR-MEDIUM	30 467	296 369	0,441 754 362 705 736
MR-D-05	AVR-MEDIUM	27 621	236 044	0,484 983 732 770 262 6
MW-W-FN	AVR-MEDIUM	22 316	2 275 623	0,130 398 640 621 995 7
MSCD_000	MSCD	5 981	138 706	1 068 748,485 638 091
MSCD_001	MSCD	8 027	197 799	809 764,790 096 290 3
MSCD_002	MSCD	6 197	131 182	1 032 697,381 888 846 4
MSCD_003	MSCD	5 405	80 096	1 258 754,319 626 802 6
MSCD_004	MSCD	5 095	66 644	1 351 273,929 247 311
MSCD_005	MSCD	7 212	162 380	934 482,849 093 364 8
MSCD_006	MSCD	6 692	155 067	1 038 538,915 383 616
MSCD_007	MSCD	5 419	80 429	1 259 275,794 229 393
MSCD_008	MSCD	6 138	140 689	1 047 379,507 907 495 8
MSCD_009	MSCD	5 751	115 834	1 110 549,764 898 437
MSCD_010	MSCD	7 289	176 200	892 442,324 641 102 7
MSCD_011	MSCD	5 657	79 901	1 213 980,980 669 797 2
MSCD_012	MSCD	5 204	95 811	1 225 393,270 820 326 2
MSCD_013	MSCD	7 499	183 007	860 248,362 359 160 7
MSCD_014	MSCD	6 351	155 532	1 004 383,604 970 199 6
MSCD_015	MSCD	5 865	148 923	1 189 398,367 772 602 2
MSCD_016	MSCD	6 219	94 047	1 108 346,648 796 672 2
MSCD_017	MSCD	6 442	142 419	1 001 522,302 291 315 4
MSCD_018	MSCD	7 681	189 284	844 018,649 323 063 1
MSCD_019	MSCD	6 141	129 861	1 047 825,061 780 356 5
MSCD_020	MSCD	7 827	184 081	844 477,257 926 372 6

Table A.2: Best solution of our algorithms KCCR, ECCR, I-ECCR, and the original KaMIS upper bound (KCC). The (I)-ECCR configurations are described in Section 5.3.3.

Instance	KCC	KCCR	Best ECCR	Best I-ECCR
MR-D-01	1 863 354 499	1 754 754 382	**1 558 702 057**	**1 558 702 057**
MR-D-03	1 399 168 695	1 328 329 087	1 082 163 610	**1 082 078 096**
MR-D-05	1 186 713 710	1 134 033 844	871 634 378	**871 566 281**
MR-D-FN	1 109 245 970	1 062 773 797	799 732 117	**799 664 265**
MR-W-FN	1 359 332 691	1 312 046 268	**1 080 255 273**	**1 080 255 273**
MT-D-01	290 754 796	276 633 077	**238 166 485**	**238 166 485**
MT-D-200	326 647 135	323 374 852	**155 491 414**	**155 491 414**
MT-D-FN	308 369 298	305 011 029	**166 634 431**	**166 634 431**
MT-W-01	374 175 909	361 976 773	**312 121 568**	**312 121 568**
MT-W-200	267 867 951	265 269 369	**154 899 257**	**154 899 257**
MT-W-FN	245 852 556	243 956 023	**158 005 943**	**158 005 943**
MW-D-01	572 634 406	541 590 471	**477 467 196**	**477 467 196**
MW-D-20	593 451 405	579 229 940	**354 582 923**	**354 582 923**
MW-D-40	397 696 097	390 552 652	218 064 368	**218 045 481**
MW-D-FN	307 610 229	302 412 610	156 598 159	**156 594 579**
MW-W-01	1 520 639 965	1 460 570 893	**1 212 205 854**	**1 212 205 854**
MW-W-05	685 920 157	671 992 200	361 983 579	**361 970 268**
MW-W-10	511 884 047	504 737 963	**222 271 930**	**222 271 930**
MW-W-FN	452 822 276	447 270 740	**181 436 575**	**181 436 575**
MSCD_000	733 511 697	712 526 475	**586 281 125**	**586 281 125**
MSCD_001	507 549 567	502 359 258	**419 620 169**	**419 620 169**
MSCD_002	662 812 674	651 671 964	548 906 023	**548 902 864**
MSCD_003	764 555 528	749 650 503	**628 901 353**	**628 901 353**
MSCD_004	730 085 676	718 476 642	605 549 959	**605 360 111**
MSCD_005	582 810 239	576 473 437	**484 865 497**	**484 865 497**
MSCD_006	436 191 231	430 050 231	**347 703 648**	**347 703 648**
MSCD_007	751 743 946	737 194 040	621 924 548	**621 357 294**
MSCD_008	657 052 392	649 204 576	**553 635 953**	**553 635 953**
MSCD_009	770 303 501	754 816 853	617 904 321	**617 628 507**
MSCD_010	566 164 897	559 170 497	464 621 556	**463 796 312**
MSCD_011	607 990 485	600 737 327	**488 620 373**	**488 620 373**
MSCD_012	782 897 268	766 116 540	**647 363 476**	**647 363 476**
MSCD_013	564 400 939	556 650 339	455 699 149	**455 420 989**
MSCD_014	698 045 278	685 073 796	563 468 466	**563 353 713**
MSCD_015	428 350 477	421 896 204	**335 537 677**	**335 537 677**
MSCD_016	649 352 601	638 048 350	528 665 091	**528 348 364**
MSCD_017	628 428 927	621 863 390	527 862 717	**527 367 116**
MSCD_018	518 652 004	511 660 252	438 673 826	**438 350 078**
MSCD_019	647 100 082	639 122 660	546 687 116	**546 674 987**
MSCD_020	536 625 031	529 557 465	**435 281 292**	**435 281 292**

Table A.3: Runtimes of KCC, KCCR, ECCR with $CPV = 1$ and $CPV = 50$ in seconds. For the ECCR configurations, the average running time over the seeds 1 to 5 is used. For each algorithm, the four largest times are highlighted.

Instance	KCC	KCCR	ECCR CPV=1	ECCR CPV=50
MR-D-01	0,286 323 0	0,538 965 0	0,869 221 2	1,067 632 0
MR-D-03	0,455 948 0	1,237 860 0	1,841 302 0	2,629 692 0
MR-D-05	0,362 249 0	1,708 640 0	2,382 998 0	3,446 954 0
MR-D-FN	0,474 459 0	2,263 050 0	2,787 606 0	4,214 558 0
MR-W-FN	0,077 614 9	1,185 920 0	7,378 086 0	26,549 040 0
MT-D-01	0,000 699 5	0,015 233 1	0,013 666 5	0,012 552 5
MT-D-200	0,029 859 1	1,351 240 0	0,321 707 8	0,327 121 8
MT-D-FN	0,069 735 6	1,595 760 0	0,347 736 0	0,347 552 8
MT-W-01	0,000 790 7	0,008 775 5	0,009 592 9	0,008 499 8
MT-W-200	0,053 538 4	1,105 660 0	0,308 095 8	0,339 908 0
MT-W-FN	0,150 544 0	1,172 410 0	0,418 156 4	0,455 050 4
MW-D-01	0,018 190 0	0,058 593 0	0,238 268 4	0,258 561 8
MW-D-20	0,137 706 0	2,660 200 0	1,540 884 0	1,537 358 0
MW-D-40	0,298 037 0	7,178 180 0	2,874 996 0	2,820 290 0
MW-D-FN	0,606 426 0	13,812 900 0	4,654 812 0	4,648 432 0
MW-W-01	0,004 095 6	0,115 487 0	0,582 822 8	1,499 690 0
MW-W-05	0,030 393 7	3,565 560 0	18,288 820 0	35,645 340 0
MW-W-10	0,091 445 6	9,673 920 0	71,219 680 0	82,058 200 0
MW-W-FN	0,125 752 0	8,649 500 0	110,870 000 0	110,849 800 0
MSCD_000	0,335 778 0	0,710 873 0	0,597 698 8	0,725 272 0
MSCD_001	0,321 021 0	0,802 435 0	0,708 624 4	0,807 622 6
MSCD_002	0,318 203 0	0,671 296 0	0,546 969 8	0,615 060 8
MSCD_003	0,090 837 1	0,295 169 0	0,208 238 4	0,235 022 0
MSCD_004	0,048 662 3	0,200 626 0	0,138 786 8	0,149 585 6
MSCD_005	0,130 957 0	0,705 864 0	0,535 462 0	0,630 667 8
MSCD_006	0,055 753 0	0,411 715 0	0,351 866 6	0,407 652 0
MSCD_007	0,081 670 5	0,284 822 0	0,198 560 0	0,235 863 2
MSCD_008	0,358 613 0	0,722 511 0	0,611 156 8	0,688 902 8
MSCD_009	0,300 845 0	0,601 112 0	0,502 960 2	0,574 786 4
MSCD_010	0,283 749 0	0,705 068 0	0,696 577 4	0,731 387 2
MSCD_011	0,048 320 3	0,251 836 0	0,147 087 2	0,167 005 0
MSCD_012	0,238 925 0	0,487 657 0	0,408 496 8	0,456 819 4
MSCD_013	0,430 977 0	0,800 775 0	0,616 246 6	0,783 167 6
MSCD_014	0,286 459 0	0,850 243 0	0,717 846 0	0,784 718 4
MSCD_015	0,025 827 7	0,415 094 0	0,336 619 4	0,496 344 0
MSCD_016	0,066 566 1	0,319 908 0	0,197 900 8	0,248 145 8
MSCD_017	0,355 246 0	0,713 856 0	0,588 205 6	0,701 048 6
MSCD_018	0,332 848 0	0,762 426 0	0,704 505 0	0,781 858 2
MSCD_019	0,313 980 0	0,658 732 0	0,515 520 0	0,603 898 4
MSCD_020	0,315 771 0	0,692 888 0	0,560 623 0	0,713 897 4

Table A.4: Number of cliques computed by the ECCR algorithm. Mean values over seeds 1 to 5.

Instance	CPV=1	CPV=2	CPV=3	CPV=4	CPV=10	CPV=20	CPV=50
MR-D-01	10 090,0	10 974,2	11 000,0	11 002,8	**11 003,2**	**11 003,2**	**11 003,2**
MR-D-03	13 212,0	13 749,0	13 775,4	13 785,4	**13 787,0**	**13 787,0**	**13 787,0**
MR-D-05	14 745,4	15 158,8	15 180,8	**15 188,2**	15 187,6	15 187,6	15 187,6
MR-D-FN	15 277,4	15 646,6	15 668,2	15 677,6	**15 682,8**	**15 682,8**	**15 682,8**
MR-W-FN	15 315,6	25 308,6	27 435,2	27 761,2	27 900,8	**27 902,4**	**27 902,4**
MT-D-01	562,4	581,8	**582,8**	**582,8**	**582,8**	**582,8**	**582,8**
MT-D-200	2 045,4	2 056,6	2 055,2	2 054,6	**2 057,6**	**2 057,6**	**2 057,6**
MT-D-FN	1 922,4	**1 931,2**	1 928,0	1 921,8	1 920,0	1 920,0	1 920,0
MT-W-01	549,2	**570,4**	**570,4**	**570,4**	**570,4**	**570,4**	**570,4**
MT-W-200	3 842,0	3 896,0	**3 899,4**	3 898,0	3 896,8	3 896,8	3 896,8
MT-W-FN	3 712,0	**3 763,6**	3 760,4	3 761,8	3 761,6	3 761,6	3 761,6
MW-D-01	2 955,4	3 227,6	3 243,0	3 245,2	**3 246,0**	**3 246,0**	**3 246,0**
MW-D-20	5 743,2	5 803,8	5 819,2	5 835,0	**5 836,8**	**5 836,8**	**5 836,8**
MW-D-40	6 410,6	6 414,2	6 460,6	6 471,4	6 482,4	**6 483,2**	**6 483,2**
MW-D-FN	6 826,8	6 836,4	6 884,4	6 897,6	6 918,0	**6 924,0**	**6 924,0**
MW-W-01	3 017,2	5 331,8	6 398,6	6 740,4	6 934,8	**6 940,4**	**6 940,4**
MW-W-05	10 578,8	15 592,6	16 730,0	16 966,8	17 409,2	17 767,6	**17 981,6**
MW-W-10	16 485,8	21 082,0	21 841,2	22 040,2	22 722,2	23 342,4	**23 884,2**
MW-W-FN	19 140,8	23 338,4	23 979,0	24 416,8	25 218,8	26 038,2	**26 887,8**
MSCD_000	4 943,2	6 715,0	7 041,0	7 101,8	7 217,2	**7 225,4**	7 224,0
MSCD_001	6 900,4	9 596,0	10 138,8	10 274,4	10 447,2	10 461,6	**10 462,2**
MSCD_002	4 974,6	6 494,8	6 777,6	6 861,0	6 938,6	**6 950,8**	6 950,4
MSCD_003	4 519,8	5 688,2	5 867,2	5 898,0	**5 940,2**	5 939,4	5 939,4
MSCD_004	4 149,6	5 055,4	5 183,4	5 213,6	5 247,8	**5 252,2**	**5 252,2**
MSCD_005	6 262,2	8 838,4	9 343,6	9 487,0	9 691,2	**9 702,2**	9 701,0
MSCD_006	6 078,2	8 558,8	9 040,2	9 175,4	9 324,2	**9 344,4**	9 344,0
MSCD_007	4 494,8	5 707,2	5 887,4	5 932,4	5 975,2	**5 975,4**	**5 975,4**
MSCD_008	5 098,8	6 974,8	7 303,0	7 390,4	7 511,0	**7 520,6**	7 519,4
MSCD_009	4 588,4	6 032,8	6 287,4	6 361,4	6 442,8	**6 448,2**	**6 448,2**
MSCD_010	6 263,4	8 626,0	9 087,6	9 196,8	9 353,8	**9 378,2**	9 377,4
MSCD_011	4 621,2	5 596,8	5 725,6	5 756,2	**5 788,8**	**5 788,8**	**5 788,8**
MSCD_012	4 107,0	5 281,2	5 464,0	5 513,8	5 567,4	5 573,0	**5 573,4**
MSCD_013	6 278,6	8 550,4	9 038,2	9 148,2	9 254,2	**9 277,0**	**9 277,0**
MSCD_014	5 274,6	7 152,8	7 473,8	7 587,2	7 695,2	**7 712,4**	7 711,4
MSCD_015	5 451,4	8 265,6	8 950,6	9 101,6	9 281,4	9 294,0	**9 296,0**
MSCD_016	5 289,8	6 707,8	6 909,2	6 958,8	7 025,0	7 030,6	**7 030,8**
MSCD_017	5 299,4	7 074,4	7 397,4	7 487,8	7 593,2	**7 600,8**	7 600,6
MSCD_018	6 506,4	8 855,6	9 321,2	9 454,8	9 584,2	9 603,2	**9 604,0**
MSCD_019	5 061,4	6 620,8	6 877,8	6 938,2	6 993,4	**7 001,4**	7 001,2
MSCD_020	6 796,6	9 557,0	10 097,2	10 255,2	10 413,6	10 441,6	**10 442,4**

Table A.5: Mean **upper bounds** of the **ECCR** algorithm for different CPV. Seeds 1 to 5 were used.

Instance	CPV=1	CPV=2	CPV=3	CPV=4	CPV=5	CPV=7	CPV=9	CPV=10	CPV=11	CPV=13	CPV=15	CPV=20	CPV=50
MR-D-01	1 566 388 534,6	1 558 860 287,8	1 558 806 847,6	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**
MR-D-03	1 085 813 047,2	1 082 285 814,6	1 082 270 334,4	1 082 282 451,8	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**
MR-D-05	873 893 514,0	**871 762 907,0**	871 799 953,0	871 802 475,6	871 803 566,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0
MR-D-FN	801 520 209,8	799 949 372,2	799 844 354,0	**799 794 009,2**	799 812 391,4	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2
MR-W-FN	1 109 720 358,2	1 080 270 832,8	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**
MT-D-01	238 399 713,0	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**
MT-D-200	155 564 473,4	155 564 473,4	155 564 473,4	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**
MT-D-FN	166 717 087,4	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**
MT-W-01	312 466 895,2	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**
MT-W-200	**154 902 734,2**	154 906 211,4	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6
MT-W-FN	158 361 216,4	158 036 377,4	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**
MW-D-01	480 497 958,4	477 610 930,0	477 607 617,0	**477 607 580,6**	**477 607 580,6**	**477 607 580,6**	**477 607 580,6**	**477 607 580,6**	**477 607 580,6**	**477 607 580,6**	**477 607 580,6**	**477 607 580,6**	**477 607 580,6**
MW-D-20	355 354 452,6	354 952 590,6	354 935 022,6	354 904 828,0	**354 903 605,6**	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8
MW-D-40	218 423 825,6	**218 207 983,8**	218 208 632,6	218 220 550,0	218 220 504,4	218 220 359,0	218 220 547,4	218 220 547,4	218 220 547,4	218 220 547,4	218 220 547,4	218 220 547,4	218 220 547,4
MW-D-FN	156 785 582,2	156 662 275,8	156 676 551,2	156 674 655,6	156 677 005,0	156 660 487,0	156 660 487,0	156 660 487,0	**156 660 485,8**	**156 660 485,8**	**156 660 485,8**	**156 660 485,8**	**156 660 485,8**
MW-W-01	1 266 534 426,2	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**
MW-W-05	383 090 024,8	363 769 948,6	362 567 780,0	362 442 188,8	**362 433 853,6**	362 496 109,4	362 477 068,8	362 488 635,8	362 471 716,2	362 446 533,6	362 458 435,6	362 474 231,4	362 477 620,8
MW-W-10	229 049 617,2	222 956 635,0	222 850 908,8	222 773 394,8	222 661 079,0	222 657 034,2	222 638 993,4	222 648 945,8	222 647 349,6	222 640 414,0	222 647 361,2	**222 635 323,2**	222 650 476,6
MW-W-FN	185 011 878,6	181 882 811,2	181 707 057,6	181 667 069,2	181 645 012,8	**181 622 456,6**	181 682 644,0	181 673 493,0	181 697 373,6	181 699 492,4	181 688 568,6	181 698 319,8	181 697 367,0
MSCD_000	592 997 299,4	589 001 160,8	**587 853 332,4**	588 455 993,6	588 351 563,0	588 330 556,2	588 249 733,6	588 183 327,0	588 265 144,2	588 376 064,2	588 287 398,2	588 352 985,0	588 355 234,0
MSCD_001	427 331 998,6	422 179 605,0	422 026 786,8	421 675 463,4	**421 334 606,6**	422 011 283,6	421 863 735,4	421 820 965,8	421 760 641,4	421 810 709,4	421 699 415,2	421 630 350,6	421 630 350,6
MSCD_002	553 976 423,4	550 527 102,4	**550 157 846,6**	550 370 556,8	550 425 305,4	550 824 444,0	551 076 062,0	551 146 981,6	551 206 628,4	551 158 152,6	551 090 495,0	551 118 360,0	551 116 534,6
MSCD_003	639 312 458,0	631 328 124,0	630 796 335,8	**629 907 643,6**	630 086 844,6	630 053 739,0	630 314 506,0	630 462 747,8	630 377 980,6	630 443 439,0	630 443 439,0	630 443 439,0	630 443 439,0
MSCD_004	612 286 999,0	606 774 107,4	606 766 727,8	606 803 785,6	606 407 457,4	**606 385 921,2**	606 478 198,2	606 533 371,6	606 520 019,0	606 464 326,8	606 464 326,8	606 464 326,8	606 464 326,8
MSCD_005	494 826 894,6	487 383 682,4	486 325 156,0	486 074 855,2	**485 846 415,4**	486 615 741,8	486 772 701,6	486 806 377,6	486 772 451,0	486 889 166,0	486 817 492,2	486 843 857,0	486 843 857,0
MSCD_006	359 310 240,4	351 374 635,2	350 777 364,4	349 886 924,0	349 790 491,0	349 628 810,6	349 575 684,2	349 432 728,8	349 482 266,8	349 326 802,8	349 303 173,6	**349 248 838,6**	**349 248 838,6**
MSCD_007	628 503 352,6	623 558 228,2	**622 715 764,8**	623 252 577,8	623 647 944,8	623 859 510,0	623 791 190,0	623 898 784,4	623 888 489,6	623 931 865,0	623 865 445,0	623 865 135,0	623 865 135,0
MSCD_008	561 402 820,4	556 935 647,2	556 503 186,2	556 297 550,0	**555 963 910,0**	556 292 634,2	556 204 109,8	556 089 635,8	556 013 533,6	556 230 178,6	556 130 058,0	555 974 196,4	555 988 882,6
MSCD_009	622 647 295,6	619 392 864,0	619 319 745,4	619 613 974,8	**619 284 455,4**	619 630 094,4	619 464 001,6	619 478 368,4	619 402 806,6	619 567 723,4	619 612 019,8	619 668 334,8	619 668 334,8
MSCD_010	471 073 154,8	466 315 944,4	**465 388 821,6**	465 654 538,4	466 230 573,2	465 986 195,8	465 983 154,6	465 974 086,0	465 880 111,6	465 714 829,6	465 785 260,2	465 840 670,6	465 840 670,6
MSCD_011	495 103 395,2	490 585 951,0	489 874 803,4	490 117 097,8	489 955 744,4	**489 759 541,0**	489 792 863,4	489 792 863,4	489 792 863,4	489 792 863,4	489 792 863,4	489 792 863,4	489 792 863,4
MSCD_012	653 493 599,6	649 416 639,4	649 412 415,4	**649 144 224,0**	649 368 974,4	649 208 857,6	649 204 053,4	649 205 164,2	649 208 960,2	649 368 513,8	649 393 902,0	649 395 656,0	649 395 656,0
MSCD_013	461 841 712,4	457 605 479,4	457 670 715,2	457 638 169,4	457 393 447,8	**457 141 866,0**	457 236 548,6	457 221 898,6	457 369 300,6	457 267 298,6	457 396 711,4	457 494 199,2	457 494 199,2
MSCD_014	568 095 981,8	565 008 877,6	565 131 956,8	565 151 196,0	564 500 141,6	**564 303 512,2**	564 870 400,2	564 849 896,0	564 904 945,4	564 844 866,2	564 822 509,2	564 853 869,8	564 853 869,8
MSCD_015	351 380 045,2	338 763 453,4	337 562 700,6	337 840 593,8	337 856 949,2	337 543 020,4	**337 316 695,8**	337 425 171,0	337 353 776,2	337 499 587,4	337 500 162,2	337 515 435,6	337 507 642,2
MSCD_016	536 393 320,2	531 161 917,4	530 270 929,8	530 548 191,6	**530 261 913,4**	530 696 969,4	530 728 456,6	530 816 882,2	530 863 665,2	530 855 153,8	530 856 954,8	530 905 920,0	530 905 920,0
MSCD_017	534 206 304,2	**529 475 477,0**	529 560 988,2	529 984 089,2	530 024 130,8	530 134 455,6	530 116 852,0	530 205 103,2	530 065 296,4	529 765 625,2	529 810 305,2	529 820 018,4	529 820 018,4
MSCD_018	445 533 570,0	440 644 142,4	**439 926 558,8**	440 185 126,2	440 441 999,2	440 643 958,6	440 374 350,2	440 281 108,0	440 175 019,2	440 320 769,8	440 402 151,2	440 423 151,0	440 423 151,0
MSCD_019	553 170 422,0	549 281 846,2	548 920 557,2	549 248 998,6	549 067 012,6	**548 478 528,6**	548 658 419,0	548 696 093,8	548 791 319,4	548 942 907,6	549 151 218,4	549 151 218,4	549 151 218,4
MSCD_020	445 523 875,0	438 210 998,8	437 774 606,0	437 479 399,0	437 416 688,4	437 331 707,4	437 496 190,2	437 536 317,0	437 417 403,4	437 288 937,0	437 286 404,0	**437 240 651,0**	437 240 729,0

Table A.6: Mean **running times** of the **ECCR** algorithm for different CPV. Seeds 1 to 5 were used.

Instance	CPV=1	CPV=2	CPV=3	CPV=4	CPV=5	CPV=7	CPV=9	CPV=10	CPV=11	CPV=13	CPV=15	CPV=20	CPV=50
MR-D-01	**0,869 221**	1,109 930	1,062 806	1,094 800	1,067 124	1,049 799	1,071 674	1,070 476	1,068 388	1,062 452	1,054 854	1,077 316	1,067 632
MR-D-03	**1,841 302**	2,612 236	2,709 836	2,718 996	2,481 178	2,566 302	2,612 972	2,606 344	2,540 198	2,580 178	2,687 844	2,532 620	2,629 692
MR-D-05	**2,382 998**	3,595 324	3,571 598	3,695 206	3,149 832	3,590 654	3,526 662	3,583 186	3,546 864	3,511 298	3,479 208	3,503 124	3,446 954
MR-D-FN	**2,787 606**	4,287 392	4,198 506	4,178 988	3,839 950	4,125 614	4,230 088	4,245 522	4,309 576	4,206 390	4,413 540	4,154 284	4,214 558
MR-W-FN	**7,378 086**	32,360 060	28,145 960	26,140 260	22,679 060	25,137 080	22,803 880	26,812 840	24,188 120	25,154 440	26,983 940	24,813 780	26,549 040
MT-D-01	0,013 666	0,012 353	0,012 554	0,012 649	0,012 742	0,012 643	0,013 314	0,013 112	**0,011 703**	0,013 263	0,012 643	0,012 692	0,012 553
MT-D-200	0,321 708	**0,311 462**	0,322 552	0,318 225	0,312 899	0,311 736	0,315 565	0,316 653	0,322 464	0,312 278	0,315 520	0,317 476	0,327 122
MT-D-FN	0,347 736	0,345 747	0,350 602	0,349 647	0,347 463	0,344 880	0,346 710	0,352 409	0,344 959	0,348 241	**0,339 830**	0,346 978	0,347 553
MT-W-01	0,009 593	**0,007 678**	0,007 905	0,009 285	0,008 939	0,008 387	0,009 073	0,008 592	0,009 166	0,009 467	0,009 004	0,008 342	0,008 500
MT-W-200	**0,308 096**	0,332 500	0,333 305	0,339 403	0,330 659	0,335 749	0,339 286	0,342 357	0,340 315	0,338 461	0,338 004	0,333 391	0,339 908
MT-W-FN	**0,418 156**	0,431 994	0,444 868	0,454 441	0,445 834	0,444 750	0,448 369	0,454 095	0,439 539	0,440 017	0,437 336	0,445 692	0,455 050
MW-D-01	**0,238 268**	0,245 437	0,257 780	0,256 850	0,251 706	0,262 553	0,260 777	0,261 891	0,263 526	0,259 497	0,257 281	0,260 964	0,258 562
MW-D-20	1,540 884	1,521 568	**1,483 236**	1,518 374	1,523 260	1,552 970	1,586 726	1,542 234	1,496 948	1,540 192	1,560 636	1,537 706	1,537 358
MW-D-40	2,874 996	**2,792 920**	3,013 916	2,860 096	2,833 452	2,882 070	2,877 318	2,938 464	2,816 626	2,906 620	2,808 052	2,963 498	2,820 290
MW-D-FN	4,654 812	4,598 356	**4,566 886**	4,621 000	4,604 702	4,662 970	4,714 216	4,691 404	4,611 406	4,630 596	4,664 102	4,689 210	4,648 432
MW-W-01	**0,582 823**	2,345 016	1,680 782	1,477 946	1,585 370	1,514 378	1,440 104	1,439 542	1,495 132	1,497 010	1,589 254	1,553 876	1,499 690
MW-W-05	**18,288 820**	48,604 060	34,477 300	34,080 420	33,705 780	32,842 140	33,737 620	34,710 220	33,509 000	35,970 740	36,032 740	34,895 740	35,645 340
MW-W-10	71,219 680	**70,001 140**	70,591 580	74,432 480	71,407 300	74,430 260	73,361 600	76,486 780	78,817 460	74,707 060	74,095 920	80,402 160	82,058 200
MW-W-FN	110,870 000	**91,482 420**	94,218 980	91,952 920	93,554 320	93,453 940	101,225 380	98,183 240	94,956 500	100,791 120	96,774 880	105,040 800	110,849 800
MSCD_000	**0,597 699**	0,693 860	0,706 638	0,662 224	0,691 225	0,726 686	0,718 290	0,727 391	0,729 718	0,730 683	0,730 453	0,659 331	0,725 272
MSCD_001	**0,708 624**	0,752 578	0,775 077	0,786 752	0,792 883	0,815 261	0,778 732	0,800 447	0,798 031	0,821 583	0,823 939	0,806 315	0,807 623
MSCD_002	**0,546 970**	0,602 259	0,613 579	0,626 869	0,630 339	0,607 021	0,610 953	0,584 239	0,630 596	0,637 031	0,611 261	0,606 573	0,615 061
MSCD_003	**0,208 238**	0,226 219	0,226 829	0,230 688	0,230 121	0,230 027	0,238 256	0,233 424	0,232 249	0,228 563	0,231 243	0,230 802	0,235 022
MSCD_004	**0,138 787**	0,143 969	0,150 817	0,148 432	0,149 397	0,151 044	0,153 348	0,149 355	0,152 414	0,150 754	0,151 609	0,153 561	0,149 586
MSCD_005	**0,535 462**	0,608 451	0,642 986	0,684 784	0,650 903	0,685 532	0,646 291	0,680 639	0,636 742	0,616 855	0,700 223	0,678 291	0,630 668
MSCD_006	**0,351 867**	0,369 575	0,387 418	0,391 477	0,387 954	0,413 808	0,410 482	0,415 664	0,394 104	0,404 625	0,414 383	0,420 604	0,407 652
MSCD_007	**0,198 560**	0,220 604	0,235 818	0,236 115	0,233 506	0,234 570	0,241 667	0,239 568	0,242 654	0,235 459	0,237 110	0,234 616	0,235 863
MSCD_008	0,611 157	0,674 560	**0,591 548**	0,673 695	0,705 390	0,656 094	0,708 977	0,706 967	0,706 187	0,683 497	0,704 501	0,712 908	0,688 903
MSCD_009	**0,502 960**	0,548 003	0,565 115	0,567 790	0,568 499	0,571 551	0,574 774	0,571 710	0,574 041	0,577 848	0,572 626	0,575 067	0,574 786
MSCD_010	0,696 577	**0,694 074**	0,723 366	0,755 772	0,704 412	0,783 889	0,740 289	0,727 172	0,724 455	0,732 681	0,750 848	0,755 845	0,731 387
MSCD_011	**0,147 087**	0,160 272	0,167 047	0,163 095	0,167 388	0,168 759	0,167 375	0,166 074	0,165 984	0,165 890	0,171 390	0,167 006	0,167 005
MSCD_012	**0,408 497**	0,431 505	0,445 746	0,449 006	0,454 402	0,461 245	0,458 025	0,454 447	0,452 003	0,458 239	0,459 466	0,456 364	0,456 819
MSCD_013	**0,616 247**	0,671 907	0,749 125	0,710 513	0,756 155	0,794 906	0,769 543	0,795 368	0,749 113	0,836 277	0,844 641	0,813 249	0,783 168
MSCD_014	0,717 846	**0,674 551**	0,769 224	0,745 629	0,719 430	0,786 142	0,800 837	0,737 671	0,815 634	0,736 754	0,800 865	0,740 748	0,784 718
MSCD_015	**0,336 619**	0,429 618	0,458 770	0,473 586	0,465 581	0,494 113	0,482 090	0,498 726	0,479 306	0,481 488	0,504 040	0,506 025	0,496 344
MSCD_016	**0,197 901**	0,234 500	0,239 897	0,243 872	0,242 833	0,241 874	0,244 226	0,241 864	0,245 610	0,244 421	0,247 007	0,246 740	0,248 146
MSCD_017	**0,588 206**	0,595 942	0,687 228	0,640 089	0,622 237	0,647 460	0,676 931	0,652 681	0,708 049	0,674 948	0,598 695	0,689 061	0,701 049
MSCD_018	**0,704 505**	0,719 402	0,770 158	0,764 404	0,716 130	0,791 985	0,826 792	0,783 668	0,725 856	0,827 736	0,825 655	0,809 311	0,781 858
MSCD_019	**0,515 520**	0,580 557	0,568 334	0,600 039	0,604 964	0,604 041	0,603 843	0,605 365	0,605 924	0,583 872	0,607 252	0,606 243	0,603 898
MSCD_020	**0,560 623**	0,684 141	0,667 607	0,684 865	0,689 086	0,726 193	0,706 963	0,686 902	0,673 219	0,707 613	0,692 647	0,686 421	0,713 897

Table A.7: Mean **upper bounds** of the **I-ECCR** algorithm for different CPV. Seeds 1 to 5 were used.

Instance	CPV=1	CPV=2	CPV=3	CPV=4	CPV=5	CPV=7	CPV=9	CPV=10	CPV=11	CPV=13	CPV=15	CPV=20	CPV=50
MR-D-01	1 558 798 900,2	1 558 809 717,2	1 558 806 847,6	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**	**1 558 797 450,8**
MR-D-03	1 082 294 064,6	1 082 262 903,6	1 082 267 663,0	1 082 282 451,8	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**	**1 082 260 072,4**
MR-D-05	**871 699 591,4**	871 750 588,4	871 799 953,0	871 802 475,6	871 803 566,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0	871 803 316,0
MR-D-FN	**799 774 195,6**	799 823 764,8	799 802 550,4	799 794 009,2	799 812 391,4	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2	799 812 622,2
MR-W-FN	1 080 273 343,4	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**	**1 080 255 273,0**
MT-D-01	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**	**238 166 485,0**
MT-D-200	155 564 473,4	155 564 473,4	155 564 473,4	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**	**155 544 590,8**
MT-D-FN	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**	**166 634 431,0**
MT-W-01	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**	**312 121 568,0**
MT-W-200	**154 902 734,2**	154 906 211,4	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6	154 909 688,6
MT-W-FN	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**	**158 005 943,0**
MW-D-01	477 603 311,0	**477 601 449,8**	477 607 617,0	477 607 580,6	477 607 580,6	477 607 580,6	477 607 580,6	477 607 580,6	477 607 580,6	477 607 580,6	477 607 580,6	477 607 580,6	477 607 580,6
MW-D-20	354 923 948,2	354 952 580,4	354 935 022,6	354 904 828,0	**354 903 605,6**	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8	354 903 959,8
MW-D-40	218 234 275,8	**218 207 983,8**	218 208 632,6	218 220 550,0	218 220 504,4	218 220 359,0	218 220 547,4	218 220 547,4	218 220 547,4	218 220 547,4	218 220 547,4	218 220 547,4	218 220 547,4
MW-D-FN	**156 657 603,2**	156 661 069,2	156 676 551,2	156 674 655,6	156 677 005,0	156 660 487,0	156 660 487,0	156 660 487,0	156 660 485,8	156 660 485,8	156 660 485,8	156 660 485,8	156 660 485,8
MW-W-01	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**	**1 212 205 854,0**
MW-W-05	362 503 332,0	362 593 572,2	362 454 167,4	**362 387 880,4**	362 418 346,0	362 496 109,0	362 477 068,8	362 488 635,8	362 471 716,2	362 446 533,6	362 458 435,6	362 474 231,4	362 477 620,8
MW-W-10	222 662 982,4	222 713 641,2	222 800 476,8	222 758 102,2	222 660 620,2	222 657 034,2	222 638 993,4	222 648 945,8	222 647 349,6	222 640 414,0	222 647 361,2	**222 635 323,2**	222 650 476,6
MW-W-FN	181 752 048,4	181 663 603,6	181 689 009,2	181 663 106,4	181 643 376,8	**181 622 451,2**	181 682 643,8	181 673 493,0	181 697 373,6	181 699 492,4	181 688 568,6	181 698 319,8	181 697 367,0
MSCD_000	588 617 879,6	588 317 900,8	**587 852 335,4**	588 455 821,0	588 351 563,0	588 330 556,2	588 249 733,6	588 183 327,0	588 265 144,2	588 376 064,2	588 287 398,2	588 352 985,0	588 355 234,0
MSCD_001	421 838 694,0	421 617 022,0	421 875 544,8	421 662 765,8	**421 314 389,0**	421 987 191,8	421 863 735,4	421 820 965,8	421 760 641,4	421 810 709,4	421 699 415,2	421 630 350,6	421 630 350,6
MSCD_002	550 614 515,4	550 293 631,2	**550 150 389,2**	550 365 753,6	550 424 267,8	550 824 444,0	551 076 062,0	551 146 981,6	551 206 628,4	551 158 152,6	551 090 495,0	551 118 360,0	551 116 534,6
MSCD_003	630 505 333,2	631 037 970,0	630 783 923,0	**629 907 092,2**	630 085 867,4	630 053 739,0	630 314 506,0	630 462 747,8	630 377 980,6	630 443 439,0	630 443 439,0	630 443 439,0	630 443 439,0
MSCD_004	606 954 538,2	606 501 384,6	606 724 711,4	606 788 436,4	606 407 457,4	**606 385 921,2**	606 478 198,2	606 533 371,6	606 520 019,0	606 464 326,8	606 464 326,8	606 464 326,8	606 464 326,8
MSCD_005	486 609 888,6	486 501 555,4	486 220 746,0	486 073 813,6	**485 846 236,2**	486 615 741,8	486 772 701,6	486 806 377,6	486 772 451,0	486 889 166,0	486 817 492,2	486 843 857,0	486 843 857,0
MSCD_006	350 910 704,6	350 714 886,6	350 517 518,0	349 857 491,8	349 790 390,2	349 628 810,6	349 575 684,2	349 432 728,8	349 482 266,8	349 326 802,8	349 303 173,6	**349 248 838,6**	**349 248 838,6**
MSCD_007	622 878 280,0	623 118 511,0	**622 710 635,0**	623 252 577,8	623 647 944,8	623 859 510,0	623 791 190,0	623 898 784,4	623 888 489,6	623 931 865,0	623 865 445,0	623 865 135,0	623 865 135,0
MSCD_008	556 482 903,8	556 126 117,8	556 488 784,6	556 297 550,0	**555 963 910,0**	556 292 634,2	556 204 109,8	556 089 635,8	556 013 533,6	556 230 178,6	556 130 058,0	555 974 196,4	555 988 882,6
MSCD_009	619 264 981,0	**619 179 863,8**	619 308 269,8	619 595 025,4	619 264 147,8	619 630 094,4	619 464 001,6	619 478 368,4	619 402 806,6	619 567 723,4	619 612 019,8	619 668 334,8	619 668 334,8
MSCD_010	465 518 519,6	465 829 787,0	**465 363 705,6**	465 650 026,8	466 230 572,0	465 986 195,8	465 983 154,6	465 974 086,0	465 880 111,6	465 714 829,6	465 785 260,2	465 840 670,6	465 840 670,6
MSCD_011	490 108 715,6	490 352 078,0	489 874 803,4	490 105 340,0	489 955 744,4	**489 759 541,0**	489 792 863,4	489 792 863,4	489 792 863,4	489 792 863,4	489 792 863,4	489 792 863,4	489 792 863,4
MSCD_012	650 436 661,4	**649 040 111,8**	649 276 062,6	649 143 884,2	649 368 505,6	649 208 857,6	649 204 053,4	649 205 164,2	649 208 960,2	649 368 513,8	649 393 902,0	649 395 656,0	649 395 656,0
MSCD_013	457 265 574,2	**457 104 531,4**	457 599 252,0	457 638 169,4	457 393 444,8	457 141 828,8	457 236 548,6	457 221 898,6	457 369 300,6	457 267 298,6	457 396 711,4	457 494 199,2	457 494 199,2
MSCD_014	564 661 762,6	564 625 524,6	565 097 982,8	565 150 288,4	564 500 139,2	**564 303 512,2**	564 870 400,2	564 849 896,0	564 904 945,4	564 844 866,2	564 822 509,2	564 853 869,8	564 853 869,8
MSCD_015	338 715 963,8	337 738 611,6	**337 277 733,4**	337 831 166,8	337 855 301,2	337 543 020,4	337 316 695,8	337 425 171,0	337 353 776,2	337 499 587,4	337 500 162,2	337 515 435,6	337 507 642,2
MSCD_016	**530 105 711,4**	530 613 921,8	530 256 304,2	530 547 971,2	530 261 693,0	530 696 969,4	530 728 456,6	530 816 882,2	530 863 665,2	530 855 153,8	530 856 954,8	530 905 920,0	530 905 920,0
MSCD_017	529 326 072,2	**529 015 269,4**	529 517 757,0	529 964 310,0	530 010 989,8	530 134 455,6	530 116 852,0	530 205 103,2	530 065 296,4	529 765 625,2	529 810 305,2	529 820 018,4	529 820 018,4
MSCD_018	440 238 623,2	440 185 672,2	**439 873 137,0**	440 174 295,8	440 439 690,6	440 643 958,6	440 374 350,2	440 281 108,0	440 175 019,2	440 320 769,8	440 402 151,2	440 423 151,0	440 423 151,0
MSCD_019	548 795 135,8	548 843 343,0	548 787 080,4	549 243 461,4	549 066 327,8	**548 478 528,6**	548 658 419,0	548 696 093,8	548 791 319,4	548 942 907,6	549 151 218,4	549 151 218,4	549 151 218,4
MSCD_020	438 142 321,0	437 668 400,0	437 680 931,8	437 473 613,4	437 416 688,4	437 331 707,4	437 496 190,2	437 536 317,0	437 417 403,4	437 288 937,0	437 286 404,0	**437 240 651,0**	437 240 729,0

Table A.8: Mean **running times** of the **I-ECCR** algorithm for different CPV. Seeds 1 to 5 were used.

Instance	CPV=1	CPV=2	CPV=3	CPV=4	CPV=5	CPV=7	CPV=9	CPV=10	CPV=11	CPV=13	CPV=15	CPV=20	CPV=50
MR-D-01	3,073 988	1,826 688	1,093 794	1,142 622	1,083 214	1,108 300	1,058 116	1,056 528	1,098 156	1,132 566	1,117 508	**1,031 724**	1,114 460
MR-D-03	6,036 602	3,497 254	3,497 704	2,701 036	2,670 850	2,599 862	2,632 660	2,663 908	2,584 258	2,573 030	2,596 562	2,576 656	**2,490 918**
MR-D-05	8,138 164	5,317 310	3,668 406	3,622 658	3,546 934	3,628 112	**3,471 774**	3,540 998	3,539 904	3,529 914	3,489 700	3,593 416	3,593 418
MR-D-FN	10,012 550	7,246 716	5,614 270	4,305 930	4,207 316	4,259 152	4,280 508	4,201 066	4,175 340	**4,126 094**	4,182 090	4,169 184	4,223 772
MR-W-FN	66,330 760	41,852 260	28,702 580	25,907 420	**22,194 280**	25,154 280	23,038 240	26,722 400	24,203 060	25,767 380	27,038 480	25,080 880	26,223 880
MT-D-01	0,016 343	0,012 953	**0,012 121**	0,013 622	0,013 852	0,013 756	0,013 546	0,013 086	0,013 702	0,013 015	0,012 901	0,013 579	0,013 911
MT-D-200	**0,309 727**	0,314 616	0,316 355	0,326 211	0,314 332	0,315 808	0,319 724	0,315 522	0,316 974	0,315 127	0,316 485	0,319 529	0,318 041
MT-D-FN	0,374 493	**0,342 770**	0,359 820	0,357 472	0,350 551	0,356 449	0,352 479	0,344 457	0,358 021	0,351 992	0,348 731	0,353 948	0,356 744
MT-W-01	0,011 493	0,009 286	0,009 909	0,008 803	0,009 863	0,009 282	0,009 367	**0,008 606**	0,009 629	0,009 224	0,009 509	0,010 307	0,009 866
MT-W-200	0,342 543	0,350 359	0,343 480	0,347 574	0,346 896	0,348 897	0,351 405	0,347 438	0,346 025	**0,335 696**	0,349 157	0,345 156	0,348 065
MT-W-FN	0,561 436	0,469 498	0,449 249	0,464 664	0,467 996	0,449 219	**0,440 742**	0,454 582	0,452 879	0,453 943	0,446 972	0,444 146	0,444 334
MW-D-01	0,691 240	0,322 593	**0,255 271**	0,257 733	0,256 977	0,261 826	0,259 582	0,261 100	0,262 582	0,262 748	0,256 289	0,260 353	0,261 742
MW-D-20	2,792 672	1,607 002	1,541 570	**1,516 252**	1,541 140	1,559 740	1,531 410	1,573 586	1,613 078	1,668 060	1,522 756	1,565 726	1,565 742
MW-D-40	4,910 064	2,870 642	2,915 648	2,839 080	2,841 962	**2,796 104**	2,860 078	2,796 734	2,872 516	2,881 990	2,886 404	2,843 482	2,814 676
MW-D-FN	7,211 338	**4,646 100**	4,680 364	4,663 630	4,650 906	4,753 556	4,676 964	4,748 958	4,764 958	4,692 658	4,692 962	4,739 470	4,724 510
MW-W-01	4,245 320	2,353 328	1,687 298	1,479 332	1,575 788	1,506 820	1,455 204	**1,436 672**	1,492 138	1,501 386	1,597 382	1,558 142	1,501 046
MW-W-05	125,174 200	80,915 600	63,970 180	63,361 260	59,618 860	39,767 820	34,174 000	34,291 880	**33,329 960**	35,875 920	35,208 300	35,201 620	35,403 780
MW-W-10	280,319 800	136,017 800	142,856 600	129,645 440	125,186 620	74,689 100	**73,725 020**	76,242 520	78,637 740	76,311 220	73,942 940	80,392 340	80,756 240
MW-W-FN	371,619 000	176,784 200	162,624 600	146,769 080	111,289 120	**94,424 220**	101,190 740	98,085 680	97,562 700	102,481 780	98,062 780	106,443 200	109,629 860
MSCD_000	1,163 226	0,896 548	0,700 746	0,788 142	0,703 348	0,723 839	0,717 679	0,697 321	**0,667 751**	0,702 475	0,673 189	0,727 535	0,683 601
MSCD_001	1,500 432	1,230 260	1,049 049	0,889 422	0,993 035	0,913 126	0,818 697	0,821 996	0,828 176	**0,804 934**	0,831 864	0,831 491	0,823 686
MSCD_002	0,995 178	0,806 686	0,740 990	0,708 319	0,694 701	0,636 731	0,632 331	0,610 921	0,633 381	**0,588 696**	0,636 585	0,611 266	0,632 245
MSCD_003	0,421 095	0,303 194	0,289 787	0,250 190	0,253 546	0,236 777	0,232 761	0,234 082	0,237 628	0,233 121	0,232 434	**0,232 158**	0,235 320
MSCD_004	0,302 699	0,215 412	0,178 175	0,173 041	0,151 345	**0,149 689**	0,154 319	0,153 669	0,157 602	0,153 383	0,151 621	0,153 570	0,153 006
MSCD_005	1,222 212	0,944 230	0,897 131	0,842 615	0,723 664	0,687 910	0,721 986	0,694 050	0,646 807	0,659 177	**0,622 518**	0,623 058	0,640 849
MSCD_006	0,973 084	0,578 895	0,571 781	0,522 762	0,435 722	0,422 091	0,418 970	0,421 860	0,407 813	0,419 370	0,415 306	0,417 347	**0,401 313**
MSCD_007	0,421 785	0,301 563	0,276 951	0,239 143	0,238 632	**0,233 841**	0,241 476	0,244 635	0,247 149	0,246 142	0,247 859	0,237 236	0,240 240
MSCD_008	1,100 372	0,819 794	0,867 761	0,683 886	0,687 981	0,710 055	**0,634 337**	0,707 557	0,661 546	0,638 894	0,682 599	0,666 106	0,686 915
MSCD_009	0,839 143	0,692 675	0,607 461	0,599 465	0,602 117	0,575 959	0,578 319	**0,570 358**	0,578 430	0,578 056	0,573 357	0,577 044	0,574 752
MSCD_010	1,341 284	1,026 929	0,950 206	0,962 072	0,760 872	0,808 623	**0,722 841**	0,730 836	0,777 616	0,780 966	0,739 610	0,754 874	0,777 000
MSCD_011	0,282 211	0,230 690	0,173 829	0,184 780	0,166 862	0,168 945	0,170 340	0,166 842	**0,166 281**	0,169 610	0,169 333	0,168 192	0,167 547
MSCD_012	0,650 663	0,582 065	0,532 600	0,477 854	0,479 352	0,464 336	0,459 804	0,457 584	**0,456 268**	0,459 364	0,460 090	0,463 172	0,458 744
MSCD_013	1,332 410	1,042 402	0,957 736	0,845 022	0,834 790	0,834 971	0,797 929	0,808 007	**0,729 799**	0,775 710	0,833 851	0,786 422	0,765 069
MSCD_014	1,370 146	0,966 212	0,996 207	0,768 501	0,815 778	0,805 455	0,779 853	**0,744 193**	0,830 656	0,811 203	0,813 256	0,815 796	0,814 966
MSCD_015	1,100 964	0,893 070	0,765 220	0,703 685	0,527 457	0,495 350	0,501 570	0,493 233	**0,483 483**	0,493 640	0,503 919	0,503 862	0,504 421
MSCD_016	0,456 453	0,330 318	0,281 459	0,266 023	0,261 608	0,244 155	**0,242 657**	0,244 837	0,244 853	0,247 320	0,245 331	0,245 449	0,245 572
MSCD_017	0,977 799	0,792 290	0,768 005	0,760 732	0,728 691	0,705 997	**0,580 773**	0,631 180	0,657 403	0,602 528	0,616 796	0,655 259	0,676 714
MSCD_018	1,250 838	1,075 721	1,001 705	0,801 476	0,819 954	0,777 080	0,801 910	0,764 930	0,778 376	**0,730 960**	0,787 267	0,810 570	0,761 155
MSCD_019	0,912 594	0,780 209	0,683 472	0,684 393	0,606 694	0,603 420	0,602 424	**0,582 994**	0,605 986	0,604 607	0,606 851	0,606 897	0,605 802
MSCD_020	1,399 494	0,956 868	0,989 357	0,884 688	0,683 524	0,729 350	0,687 862	0,687 404	0,726 671	0,733 414	0,722 720	**0,673 223**	0,720 050

Table A.9: The **upper bounds** for all algorithms. **Mean** values of seeds 1 to 5 are used for the ECCR and I-ECCR algorithm. Green CHILS values mark instances, where an optimal upper bound was found.

Instance	KCC	KCCR	Mean ECCR	Mean I-ECCR	BSA	CHILS
MR-D-01	1 863 354 499	1 754 754 382	**1 558 797 450,8**	**1 558 797 450,8**	1 558 837 933	1 555 272 030
MR-D-03	1 399 168 695	1 328 329 087	1 082 260 072,4	1 082 260 072,4	**1 081 890 341**	1 078 928 116
MR-D-05	1 186 713 710	1 134 033 844	871 803 316,0	871 803 316,0	**871 382 459**	868 086 806
MR-D-FN	1 109 245 970	1 062 773 797	799 812 622,2	799 812 622,2	**799 435 902**	795 413 712
MR-W-FN	1 359 332 691	1 312 046 268	**1 080 255 273,0**	**1 080 255 273,0**	1 080 349 170	**1 080 255 273**
MT-D-01	290 754 796	276 633 077	**238 166 485,0**	**238 166 485,0**	238 190 197	**238 166 485**
MT-D-200	326 647 135	323 374 852	155 544 590,8	155 544 590,8	**155 524 943**	155 448 042
MT-D-FN	308 369 298	305 011 029	**166 634 431,0**	**166 634 431,0**	**166 634 431**	166 626 056
MT-W-01	374 175 909	361 976 773	**312 121 568,0**	**312 121 568,0**	312 152 337	**312 121 568**
MT-W-200	267 867 951	265 269 369	**154 909 688,6**	**154 909 688,6**	154 916 643	154 897 702
MT-W-FN	245 852 556	243 956 023	**158 005 943,0**	**158 005 943,0**	158 021 534	**158 005 943**
MW-D-01	572 634 406	541 590 471	477 607 580,6	477 607 580,6	**477 566 115**	476 437 726
MW-D-20	593 451 405	579 229 940	354 903 959,8	354 903 959,8	**354 354 403**	350 959 680
MW-D-40	397 696 097	390 552 652	218 220 547,4	218 220 547,4	**217 955 374**	215 499 651
MW-D-FN	307 610 229	302 412 610	156 660 485,8	156 660 487,0	**156 561 937**	154 789 973
MW-W-01	1 520 639 965	1 460 570 893	**1 212 205 854,0**	**1 212 205 854,0**	1 212 304 599	**1 212 205 854**
MW-W-05	685 920 157	671 992 200	362 477 620,8	362 488 635,8	**361 028 951**	359 448 366
MW-W-10	511 884 047	504 737 963	222 650 476,6	222 648 945,8	**220 095 070**	216 901 878
MW-W-FN	452 822 276	447 270 740	181 697 367,0	181 673 493,0	**179 057 525**	175 920 756
MSCD_000	733 511 697	712 526 475	588 355 234,0	588 183 327,0	**579 998 912**	579 941 037
MSCD_001	507 549 567	502 359 258	421 630 350,6	421 820 965,8	**412 657 530**	412 621 002
MSCD_002	662 812 674	651 671 964	551 116 534,6	551 146 981,6	**542 695 962**	542 641 794
MSCD_003	764 555 528	749 650 503	630 443 439,0	630 462 747,8	**623 616 851**	623 246 490
MSCD_004	730 085 676	718 476 642	606 464 326,8	606 533 371,6	**600 825 661**	600 237 047
MSCD_005	582 810 239	576 473 437	486 843 857,0	486 806 377,6	**476 576 370**	476 528 726
MSCD_006	436 191 231	430 050 231	349 248 838,6	349 432 728,8	**341 180 519**	341 098 573
MSCD_007	751 743 946	737 194 040	623 865 135,0	623 898 784,4	**616 203 775**	615 643 728
MSCD_008	657 052 392	649 204 576	555 988 882,6	556 089 635,8	**547 408 921**	547 354 264
MSCD_009	770 303 501	754 816 853	619 668 334,8	619 478 368,4	**611 962 315**	611 905 129
MSCD_010	566 164 897	559 170 497	465 840 670,6	465 974 086,0	**456 057 433**	456 011 861
MSCD_011	607 990 485	600 737 327	489 792 863,4	489 792 863,4	**485 001 429**	484 605 284
MSCD_012	782 897 268	766 116 540	649 395 656,0	649 205 164,2	**642 575 105**	642 511 000
MSCD_013	564 400 939	556 650 339	457 494 199,2	457 221 898,6	**449 372 168**	448 876 895
MSCD_014	698 045 278	685 073 796	564 853 869,8	564 849 896,0	**553 686 064**	553 634 101
MSCD_015	428 350 477	421 896 204	337 507 642,2	337 425 171,0	**328 473 920**	327 468 394
MSCD_016	649 352 601	638 048 350	530 905 920,0	530 816 882,2	**525 007 532**	524 650 259
MSCD_017	628 428 927	621 863 390	529 820 018,4	530 205 103,2	**520 652 792**	520 600 877
MSCD_018	518 652 004	511 660 252	440 423 151,0	440 281 108,0	**429 705 769**	429 662 819
MSCD_019	647 100 082	639 122 660	549 151 218,4	548 696 093,8	**540 059 757**	540 004 600
MSCD_020	536 625 031	529 557 465	437 240 729,0	437 536 317,0	**428 304 211**	428 261 405

Utilized AI Tools

General Purpose AIs

We used *ChatGPT* and *Microsoft Copilot* with the following types of prompts:

Thesis Writing

- Find paper with the following content ...
- Give me more formal examples for this text ...

Code Generation

- Fix this LaTeX error ...
- Change this LaTeX style to ...
- Write LaTeX code that reads this CSV file and shows it with this styling ...
- Edit this Python code so that the resulting plot style is ...
- Generate Python code that merges these CSV files ...
- Apply this operation on the pandas dataframe ...

Translations and Spell Checking

We used *DeepL Translate* for translations between English and German. *DeepL Write* was only used in "Corrections only" mode for spell checking. Also, *Grammarly* was only used for spell checking.

Zusammenfassung

In dieser Arbeit präsentieren wir vier Algorithmen, die obere Schranken für das Problem der Maximal Gewichteten Unabhängigen Menge (MWIS) zu berechnen. Dieses NP-vollständige Problem besteht darin, für einen Graphen eine Menge an Knoten zu finden, so dass die Knoten nicht miteinander verbunden sind und die Summe der Knotengewichte größt möglich ist. Der von uns verwendete Ansatz ist die Clique-Abdeckungs-Relaxation (engl.: clique cover relaxation). Dafür formulieren wir das MWIS als lineares Optimierungsproblem (LP), welches die Cliques aus der Clique Abdeckung als Bedingung verwendet. Diese Clique-Abdeckungs-Relaxation wenden wir sowohl auf die von KaMIS berechnete Abdeckung wie auch auf unsere eigens entwickelte Clique-Abdeckung an. KaMIS ist ein Programm zum optimalen Lösen von MWIS Problemen. Für unsere eigene Abdeckung stellen wir zusätzlich einen Algorithmus vor, der die obere Schranke iterativ verbessern kann. In der Evaluation untersuchen wir verschiedene Konfigurationen für unsere Algorithmen in Bezug auf Lösungs- und Laufzeitqualität. Unser Vergleich mit den oberen Schranken von KaMIS zeigt, dass wir deutlich bessere Werte erhalten, aber äußerst lange Laufzeiten für einige Graphen haben. Zusätzlich zeigen wir, dass zwei unserer Algorithmen auf Graphen für Fahrzeug-Routenplanung mit einem hochmodernen Algorithmus, der jedoch eine vorberechnete Clique-Abdeckung benötigt, mithalten können und in der Lage sind die optimale Lösung für einige Instanzen zu finden.

Bibliography

[1] Here wego. URL https://wego.here.com. Accessed 2025-09-17.

[2] Faisal N. Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent advances in practical data reduction. *CoRR*, abs/2012.12594, 2020. URL https://arxiv.org/abs/2012.12594.

[3] Diogo Vieira Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. *J. Heuristics*, 18(4):525–547, 2012. doi: 10.1007/S10732-012-9196-4. URL https://doi.org/10.1007/s10732-012-9196-4.

[4] Lukas Barth, Benjamin Niedermann, Martin Nöllenburg, and Darren Strash. Temporal map labeling: A new unified framework with experiments. *CoRR*, abs/1609.06327, 2016. URL http://arxiv.org/abs/1609.06327.

[5] William Brendel and Sinisa Todorovic. Segmentation as maximum-weight independent set. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pages 307–315. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper/2010/hash/a0a080f42e6f13b3a2df133f073095dd-Abstract.html.

[6] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, 2002. doi: 10.1007/S101070100263. URL https://doi.org/10.1007/s101070100263.

[7] Yuanyuan Dong, Andrew V. Goldberg, Alexander Noe, Nikos Parotsidis, Mauricio G. C. Resende, and Quico Spaen. New instances for maximum weight independent set from a vehicle routing application. *CoRR*, abs/2105.12623, 2021. URL https://arxiv.org/abs/2105.12623.

[8] 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*, 85(1):91–112, 2025. doi: 10.1002/NET.22247. URL https://doi.org/10.1002/net.22247.

[9] Joe Dundas, T. Andrew Binkowski, Bhaskar DasGupta, and Jie Liang. Topology independent protein structural alignment. *BMC Bioinform.*, 8, 2007. doi: 10.1186/1471-2105-8-388. URL https://doi.org/10.1186/1471-2105-8-388.

[10] Free Software Foundation. GLPK - GNU Linear Programming Kit, 2025. URL https://www.gnu.org/software/glpk/.

[11] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0-7167-1044-7.

[12] Alexander Gellner, Sebastian Lamm, Christian Schulz, Darren Strash, and Bogdán Zaválnij. Boosting data reduction for the maximum weight independent set problem using increasing transformations. *CoRR*, abs/2008.05180, 2020. URL https://arxiv.org/abs/2008.05180.

[13] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Evaluation of labeling strategies for rotating maps. *ACM J. Exp. Algorithmics*, 21(1):1.4:1–1.4:21, 2016. doi: 10.1145/2851493. URL https://doi.org/10.1145/2851493.

[14] Ernestine Großmann, Kenneth Langedal, and Christian Schulz. A comprehensive survey of data reduction rules for the maximum weighted independent set problem. *CoRR*, abs/2412.09303, 2024. doi: 10.48550/ARXIV.2412.09303. URL https://doi.org/10.48550/arXiv.2412.09303.

[15] Ernestine Großmann, Kenneth Langedal, and Christian Schulz. Concurrent iterated local search for the maximum weight independent set problem. In Petra Mutzel and Nicola Prezza, editors, *23rd International Symposium on Experimental Algorithms, SEA 2025, July 22-24, 2025, Venice, Italy*, volume 338 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi: 10.4230/LIPICS.SEA.2025.22. URL https://doi.org/10.4230/LIPIcs.SEA.2025.22.

[16] Gurobi Optimization, LLC. Gurobi Optimizer Documentation, 2025. URL https://docs.gurobi.com/projects/optimizer/en/current/features/concurrent.html#secconcurrent.

[17] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2025. URL https://www.gurobi.com.

[18] Stefan Haller and Bogdan Savchynskyy. A bregman-sinkhorn algorithm for the maximum weight independent set problem. arXiv preprint arXiv:2408.02086, 2024, 2024. URL https://arxiv.org/abs/2408.02086.

[19] KarlsruheMIS. Kamis: Karlsruhe maximum independent sets. https://github.com/KarlsruheMIS/KaMIS, 2024. Version 3.0; Open-source project for computing maximum independent sets and vertex covers of large sparse graphs; MIT License.

[20] 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 Stephen G. Kobourov and Henning Meyerhenke, editors, *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*, pages 144–158. SIAM, 2019. doi: 10.1137/1.9781611975499.12. URL https://doi.org/10.1137/1.9781611975499.12.

[21] Jianfeng Liu, Sihong Shao, and Chaorui Zhang. Application of causal inference techniques to the maximum weight independent set problem. *arXiv preprint arXiv:2301.05510*, 2023. doi: 10.48550/arXiv.2301.05510. URL https://arxiv.org/abs/2301.05510.

[22] Bruno C. S. Nogueira, Rian G. S. Pinheiro, and Anand Subramanian. A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optim. Lett.*, 12(3):567–583, 2018. doi: 10.1007/S11590-017-1128-7. URL https://doi.org/10.1007/s11590-017-1128-7.

[23] Mangal Prakash. *Fully Unsupervised Image Denoising, Diversity Denoising and Image Segmentation with Limited Annotations*. PhD thesis, Dresden University of Technology, Germany, 2022. URL https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-787075.

[24] Luzhi Wang, Chu-Min Li, Junping Zhou, Bo Jin, and Minghao Yin. An exact algorithm for minimum weight vertex cover problem in large graphs. *CoRR*, abs/1903.05948, 2019. URL http://arxiv.org/abs/1903.05948.

[25] Peng Wang and Stephan Bohacek. On the practical complexity of solving the maximum weighted independent set problem for optimal scheduling in wireless networks. In Xudong Wang and Ness B. Shroff, editors, *Proceedings of the 4th Annual International Conference on Wireless Internet, WICON 2008, Maui, Hawaii, USA, November 17-19, 2008*, ACM International Conference Proceeding Series, page 15. ICST, 2008. doi: 10.4108/ICST.WICON2008.4862. URL https://doi.org/10.4108/ICST.WICON2008.4862.

[26] Jeffrey S. Warren and Illya V. Hicks. Combinatorial branch-and-bound for the maximum weight independent set problem. Technical report, Texas A&M University, Citeseer, August 7 2006. URL https://www.cmor-faculty.rice.edu/~ivhicks/jeff.rev.pdf.

[27] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *Eur. J. Oper. Res.*, 242(3):693–709, 2015. doi: 10.1016/J.EJOR.2014.09.064. URL https://doi.org/10.1016/j.ejor.2014.09.064.

[28] Xiaohua Xu, Shaojie Tang, and Peng-Jun Wan. Maximum weighted independent set of links under physical interference model. In Gopal Pandurangan, V. S. Anil Kumar, Gu Ming, Yunhao Liu, and Yingshu Li, editors, *Wireless Algorithms, Systems, and Applications, 5th International Conference, WASA 2010, Beijing, China, August 15-17, 2010. Proceedings*, volume 6221 of *Lecture Notes in Computer Science*, pages 68–74. Springer, 2010. doi: 10.1007/978-3-642-14654-1_8. URL https://doi.org/10.1007/978-3-642-14654-1_8.

[29] Yaqin Zhou, Xiang-Yang Li, Min Liu, XuFei Mao, Shaojie Tang, and Zhongcheng Li. Throughput optimizing localized link scheduling for multihop wireless networks under physical interference model. *IEEE Trans. Parallel Distributed Syst.*, 25(10): 2708–2720, 2014. doi: 10.1109/TPDS.2013.210. URL https://doi.org/10.1109/TPDS.2013.210.