Combining Known Techniques to Solve the Dominating Set Problem in Practice

Marlon Dittes

April 20, 2025

4164834

Bachelor Thesis

at

Algorithm Engineering Group Heidelberg
Heidelberg University

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

Co-Supervisors:
Adil Chhabra, Ernestine Großmann, Kenneth Langedal, Henrik Reinstädtler, Henning Woydt

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Dr. Christian Schulz, for his constant guidance and support throughout this thesis and my academic journey.

A special thanks to Ernestine Großmann and Kenneth Langedal for always providing constructive feedback and insightful suggestions. I also greatly appreciate the support from Adil Chhabra, Henrik Reinstädtler, Dr. Darren Strash and Henning Woydt, whose discussions and advice were immensely helpful.

I would like to express my heartfelt gratitude to my family and friends for their unwavering support, love, and encouragement throughout my life. Their constant belief in me and their presence have been invaluable in helping me complete this work.

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, 20.04.2025

[illegible]

Marlon Dittes

Abstract

This work addresses the Dominating Set and Hitting Set problem, which are both part of the PACE Challenge 2025, by leveraging existing solving techniques. Since the Dominating Set problem can be reduced to the Hitting Set problem, we explore the use of a dedicated Hitting Set solver, as well as ILP and MaxSAT approaches, to solve Dominating Set instances. To improve solver performance, we employ data reduction techniques aimed at decreasing the overall problem size prior to solving. Our results demonstrate that an open-source MaxSAT solver significantly outperforms commonly used open-source ILP solvers, while the dedicated hitting set solver fails to keep pace with either method. Furthermore, we find that the MaxSAT solver, when combined with reductions, can even compete with Gurobi, a commercial ILP solver.

Contents

CHAPTER 1

Introduction

In recent years, top-performing submissions in the PACE (Parameterized Algorithms and Computational Experiments) Challenge [1] have frequently employed powerful general-purpose solving techniques, most notably Integer Linear Programming (ILP) and Maximum Satisfiability (MaxSAT) solvers. For instance, the second-place solver in the 2024 exact track utilized an ILP-based approach, while the first-place submission in 2022 successfully applied a MaxSAT solver. These results underscore the practical effectiveness of such solvers when applied to hard combinatorial problems. This motivates the investigation of similar techniques for this year's challenge, which centers around the well-known DOMINATING SET problem.

This thesis explores the use of ILP and MaxSAT solvers for solving instances of the DOMINATING SET problem, alongside a dedicated solver designed for the related HITTING SET problem. To enhance solver performance, we additionally apply various reduction rules as a pre-processing step to simplify the instances before solving. Our initial idea was to employ a portfolio approach that leverages the strengths of multiple solvers. This strategy involves running different solvers with varying configurations and allocating them separate portions of the overall time budget. By distributing time strategically, one aims to leverage the individual strengths of each solver and increase the likelihood of solving a broader range of instances effectively.

1.1 Motivation

The PACE Challenge is an annual competition that focuses on difficult graph problems, often from the field of parameterized complexity. Participants are tasked with solving real-world-like benchmark instances within strict time and resource constraints. One of the defining features of the PACE Challenge is its emphasis on practical algorithms that work well in practice and not just in theory. The fact that only open-source solvers are permitted makes it a valuable benchmark for assessing tools that are freely available to the research community.

This year's focus on the DOMINATING SET problem adds another layer of interest. It is a classical NP-hard problem [14] with a wide range of applications, and its structure is well-suited for applying reductions, heuristics, and transformations into related formulations such as HITTING SET or SET COVER. Notably, both HITTING SET and SET COVER are among the 21 problems originally shown to be NP-complete by Karp in his seminal 1972 paper [23], highlighting their foundational role in computational complexity. Moreover, given the diversity of instance types and solver behaviors, exploring a portfolio of solving strategies can be especially beneficial. Some solvers may perform well on certain families of instances, while others succeed where the first ones fail. Solving exact instances within strict time constraints emphasizes the potential of combining general-purpose solvers with effective pre-processing techniques. This approach proves to be not only viable but also highly competitive. This thesis aims to evaluate how effective such methods can be in addressing this year's PACE instances.

The DOMINATING SET problem is a fundamental problem in graph theory and computing science with extensive applications across a variety of fields. Formally, given a graph $G = (V, E)$ where V represents the set of vertices and E the set of edges, a *dominating set* $D \subseteq V$ is a subset of vertices such that every vertex in V is either in D or adjacent to at least one vertex in D. The goal of the DOMINATING SET problem is to find a dominating set of minimum size, usually referred to as a *minimum dominating set*.

The importance of the DOMINATING SET problem comes from its usefulness in many real-world applications. In network design [6], for example, dominating sets can help identify the best locations for servers or routers, ensuring that every device in the network is either a server or connected to one. This ensures effective communication and resource accessibility while minimizing infrastructure costs. Similarly, in the field of wireless sensor networks [37], a minimal dominating set represents an optimal configuration of sensor nodes that can monitor the entire network, thus conserving power by reducing the number of active sensors.

Additionally, with the growing popularity of social networks, the DOMINATING SET problem has found important applications in social network analysis [24]. It is used to identify key influencers within a network by finding the smallest set of individuals that can directly reach or influence every other user. This approach enables effective strategies for information dissemination, product promotion, and public outreach.

1.2 Our Contribution

In this work, we explore the application of established solving techniques, namely ILP (Integer Linear Programming) and MaxSAT solvers, to tackle the DOMINATING SET and HITTING SET problem as part of the PACE Challenge 2025. Our main contribution is evaluating these methods, along with a dedicated hitting set solver, on real-world benchmark instances, with a focus on improving solver performance through pre-processing reductions.

We introduce a systematic comparison between general-purpose solvers (ILP and MaxSAT) and a specialized solver, intending to demonstrate the benefits of combining these techniques. Additionally, we apply a range of reduction rules aimed at simplifying problem instances before they are passed to solvers, thus improving both the efficiency and effectiveness of the solving process.

Another contribution of this work is the investigation of a portfolio approach, where different solver types and configurations are evaluated with time allocation strategies. This approach leverages their complementary strengths to solve a larger number of instances. This work provides insights into the potential of combining solvers with effective pre-processing to improve performance, especially under the strict time constraints of the PACE Challenge.

Through our experiments and analysis, we aim to better understand how different solving techniques and pre-processing methods can be used to tackle complex combinatorial problems like the DOMINATING SET and HITTING SET problems in a competitive setting.

1.3 Structure

The remainder of this thesis is organized as follows. Section 2 covers the necessary fundamentals, including key definitions and problem descriptions related to the DOMINATING SET and HITTING SET problems, as well as the optimization techniques used in this work. In Section 3.3, we discuss previous research on these problems. Section 4 outlines the approach taken in this thesis, detailing the reduction rules applied as pre-processing steps and describing the solving techniques based on ILP and MaxSAT. Section 5 presents the experimental evaluation, including the datasets used and the results obtained for both the reduction techniques and the different solvers. Finally, Section 6 provides a discussion of the findings, including the conclusions drawn from the experiments and suggestions for future work.

CHAPTER 2

Fundamentals

In this part of the thesis, we introduce the key concepts and terminology that will be used throughout the remainder of the work. Our focus lies on three closely related graph problems, with the *hitting set* formulation serving as the main perspective throughout our discussion.

2.1 General Definitions

Undirected Graph. An *undirected graph* is a graph $G = (V, E)$, where V is a set of vertices and E is a set of edges with $E \subseteq \binom{V}{2}$. Two vertices $u, v \in V$ are adjacent if there is an edge between them, we write $\{u, v\} \in E$. We denote by $|V|$ the number of vertices and likewise by $|E|$ the number of edges. For a vertex $v \in V$, the *neighborhood* of v, denoted by $N(v)$, is the set of all vertices that are adjacent to v, i.e. $N(v) = \{u \in V | \{u, v\} \in E\}$. The *closed neighborhood* of a vertex v, denoted by $N[v]$, is defined as the neighborhood of v together with v itself, i.e. $N[v] = N(v) \cup v$.

Hypergraph. A *hypergraph* $H = (V, \mathcal{F})$ is a collection of hyperedges $\mathcal{F}$ over a vertex set V, meaning every $F \in \mathcal{F}$ is a subset of V. If a vertex v is part of a hyperedge F, we write $v \in F$. Similarly to the graph definition, we use $|V|$ for the number of vertices, $|\mathcal{F}|$ for the number of hyperedges, and $|F|$ for the number of vertices in a hyperedge $F \in \mathcal{F}$. Given a set of vertices $S \subseteq V$, we write $H' = H - S$ to denote the induced hypergraph obtained by removing the set of vertices S. Formally, $H' = (V \setminus S, \{F \setminus S \mid F \in \mathcal{F},\ F \setminus S \neq \emptyset\})$. This means that we remove the vertices in S from the vertex set and from each hyperedge in $\mathcal{F}$, and we discard any resulting empty hyperedges to form the new hypergraph H'. Similarly, for a set of hyperedges $L \subseteq \mathcal{F}$, we write $H' = H - L$ to denote the hypergraph obtained by removing all hyperedges in L from H. Formally, $H' = (V, \mathcal{F} \setminus L)$.

Dominating Set. A *dominating set* for the graph G is a subset $D \subseteq V$ such that every vertex $v \in V$ is either in D or is adjacent to at least one vertex in D. Formally, $D \subseteq V$ is a dominating set of G if and only if $\forall v \in V$ either $v \in D$ or there is a vertex $u \in D$ with $\{u, v\} \in E$. We call a dominating set *minimal* if it contains no other dominating set as a proper subset. A *minimum* dominating set is a dominating set with the smallest possible cardinality $|D|$. The DOMINATING SET problem asks for a minimum dominating set of any given graph.

Hitting Set. If a vertex $v \in V$ is included in a hyperedge $F \in \mathcal{F}$, we say that v *hits* the hyperedge F. Let $F(v)$ be the set of hyperedges that v hits, i.e. $F(v) = \{E \in \mathcal{F} | v \in E\}$. Accordingly, a subset of vertices $S \subseteq V$ is called a *hitting set* of a hypergraph $H = (V, \mathcal{F})$ if every hyperedge in $\mathcal{F}$ contains at least one vertex from S. Formally, $S \subseteq V$ is a hitting set of H if and only if for all hyperedges $F \in \mathcal{F}$, it holds that $H \cap F \neq \emptyset$. As for the dominating set, we say a hitting set is *minimal* if it contains no other hitting set as a proper subset. A *minimum* hitting set is a hitting set with the smallest possible cardinality $|S|$. Similarily, the HITTING SET problem asks for a minimum hitting set of any given hypergraph.

Set Cover. If an edge $F \in \mathcal{F}$ contains a vertex $v \in V$, we say F *covers* v. Based on this, we call a subset of hyperedges $S \subseteq \mathcal{F}$ a *set cover* if for all $v \in V$ there is at least one hyperedge $F \in S$ that covers v. Formally, $S \subseteq \mathcal{F}$ is a set cover if and only if for all vertices $v \in V$, there exists a hyperedge $F \in S$ such that $v \in F$. We say a set cover is *minimal* if it contains no other set cover as a proper subset and *minimum* if it is a set cover with the smallest possible cardinality $|S|$. As before, the SET COVER problem asks for a minimum set cover of a given hypergraph.

2.2 Optimization

An *Integer Linear Program* (ILP) consists of a set of variables, restricted to binary integer values, a linear objective function to be minimized or maximized, and a collection of linear constraints. Formally, an ILP can be written as: optimize $c^T x$ subject to $Ax \leq b$, where $x \in \mathbb{Z}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. Feasible solutions are those that satisfy all constraints. The objective function determines the best (optimal) one.

Maximum Satisfiability (MaxSAT) is an optimization extension of the BOOLEAN SATISFIABILITY problem (SAT). Given a boolean formula in conjunctive normal form (CNF), the MAXIMUM SATISFIABILITY problem asks for a truth assignment to the variables that satisfies the maximum number of clauses. Unlike SAT, where the goal is to determine whether all clauses can be satisfied, MaxSAT tolerates unsatisfied clauses and aims to maximize satisfaction. A natural generalization is the *Partial MaxSAT* problem, in which the formulation is divided into two sets of clauses: *hard* clauses, which must be satisfied, and *soft* clauses, which should be satisfied as much as possible. The objective is to find an assignment that satisfies all hard clauses and the maximum number of soft clauses.

In *Weighted Partial MaxSAT*, each soft clause is assigned a positive weight, representing its importance or cost. The goal then becomes to satisfy all hard clauses while maximizing the total weight of the satisfied soft clauses (or equivalently, minimizing the total weight of unsatisfied soft clauses).

CHAPTER 3

Related Work

This chapter is divided into three sections. First, we provide an overview of related work on the DOMINATING SET problem and closely related problems. Next, we review relevant research on Integer Linear Programming (ILP) approaches. Finally, we discuss related work in the context of MaxSAT solving.

3.1 Dominating Set

Throughout the history of the DOMINATING SET problem, numerous exact approaches have sought to improve upon the trivial running time of $O(2^{|V|})$, which is obtained by exhaustively enumerating all feasible solutions. Van Rooij and Bodlaender [36] reported a running time of $O(1.4969^n)$ using a measure-and-conquer technique, which was later refined by Iwata [21] through an extension known as the potential method. Both approaches follow the branch-and-reduce paradigm, relying on a combination of reduction and branching rules. Reduction rules transform the problem into an equivalent but smaller instance, while branching rules decompose the problem into multiple subinstances that are solved recursively. However, neither study included experimental evaluations, and both algorithms remain impractical for larger graph instances.

Due to the limitations of these exact approaches, various heuristic methods have been proposed to address the DOMINATING SET problem, particularly in the context of large graphs. Among these methods, metaheuristic techniques such as genetic algorithms [15], simulated annealing [16], and ant colony optimization [17] have proven effective in yielding approximate solutions with reasonable computational effort. Additionally, distributed approximation algorithms [25, 37] have also been explored, offering alternative approaches to tackle this problem. More recently, research has introduced an Iterated Greedy algorithm [8] that incorporates local improvement strategies, as well as an exact branch-and-bound method [22] that leverages improved lower bounds while presenting empirical results.

Bläsius et al. [5] present a fast branch-and-bound solver that claims to outperform modern ILP solvers, which are considered state-of-the-art for solving the minimum HITTING SET problem. Their work provides a foundation for future research by introducing an efficient algorithm that integrates lower bounds, upper bounds, and reduction rules. They conduct an experimental evaluation of different configurations of these components, highlighting the critical role of lower bounds in the algorithm's performance. Notably, the Costly Discard rule, which leverages both lower and upper bounds, plays a key role in improving the algorithm's efficiency.

3.2 ILP

Integer Linear Programming (ILP) is a widely studied and powerful method for solving combinatorial optimization problems, where the objective is to find an optimal solution subject to linear constraints, and decision variables are restricted to integer values. ILP has been applied to a vast range of problems, including scheduling [31], network design [35], resource allocation [29], and various other optimization problems.

The development of efficient ILP solvers has seen significant progress in recent decades. Modern commercial solvers such as Gurobi [13] and CPLEX [20] have become industry standards for solving large-scale ILP instances. One of the main challenges of ILP solvers is the exponential growth in solution space as the problem size increases. As a result, hybrid approaches that combine ILP with heuristics and metaheuristics [28, 32] have been explored to overcome the limitations of exact ILP solvers.

In recent PACE challenges, it has become evident that some of the top-performing algorithms leverage open-source ILP solvers. For instance, in the 2024 challenge, the second-place finisher [38] in the exact track utilized the HiGHS [33, 19] ILP solver, while the third-place team [10] employed SCIP [7]. Similarly, in the 2022 challenge, the second-place solution [11] also relied on SCIP, particularly to solve simplified instances after applying a reduction process. Given the consistent presence of these solvers in high-ranking solutions, it is crucial to experiment with them across a variety of problem instances and compare their performance against dedicated solvers.

3.3 MaxSAT

The MAXIMUM SATISFIABILITY (MaxSAT) problem, a generalization of the classical SATISFIABILITY (SAT) problem, has gained significant attention due to its applicability in various fields such as machine learning [18] and hardware design [9].

Similar to the PACE challenges, there is an annual MaxSAT competition [26] in which various open-source MaxSAT solvers from research teams worldwide compete to solve instances of the MaxSAT problem. One of the top-performing exact solvers in recent competitions is UWrMaxSat [27], followed closely by EvalMaxSAT [2], which was used after preprocessing in the PACE 2022 exact track winner's algorithm [30]. Additionally, the first-place winner of PACE 2023 [34] also utilized a SAT solver [4] as part of their overall algorithm.

Similar to ILP solvers, existing MaxSAT solvers are highly adaptable and can be applied to a broad spectrum of problems. Given their flexibility, it is beneficial to consider and compare ILP solvers and MaxSAT solvers when approaching a new challenge. By evaluating both methods, one can identify which approach is more efficient and suitable based on the specific characteristics of the problem.

CHAPTER 4

Approach

The general approach for solving Dominating Set instances is as follows: First, we transform the Dominating Set instance into a Hitting Set instance. We then reduce the instance size by applying multiple reduction rules specific to the Hitting Set problem. Finally, the reduced instance is translated into an ILP or MaxSAT formulation, which can be solved using existing solvers.

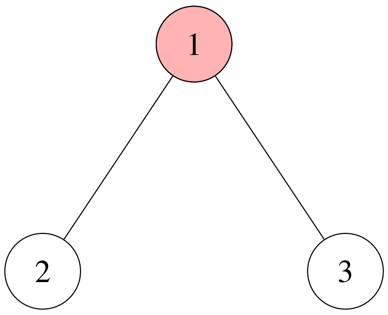
4.1 Transformation

Given a graph $G = (V, E)$ and its corresponding Dominating Set problem, we can transform it into a Hitting Set problem on a hypergraph. The goal of this transformation is to create a hypergraph $H = (V, \mathcal{F})$ such that a minimum dominating set in G corresponds to a minimum hitting set in H.

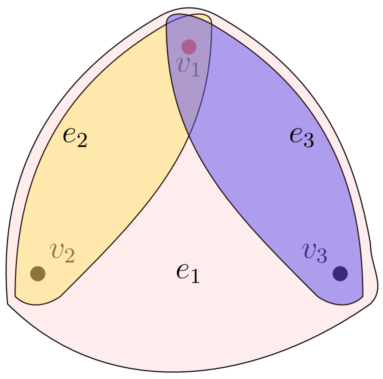
To begin the translation, the vertices of the hypergraph H correspond directly to the set of vertices V in the graph G. Next, for each vertex $v \in V$, we define a hyperedge in the hypergraph. This hyperedge, denoted as F_v, consists of the vertex v and all of its neighbors in the graph, that is, $F_v = \{v\} \cup N(v) = N[v]$. The collection of all hyperedges is thus given by $\mathcal{F} = \{F_v \mid v \in V\}$.

A hitting set $S \subseteq V$ in the hypergraph intersects with every hyperedge $F_v \in \mathcal{F}$, meaning $H \cap F_v \neq \emptyset$ for all $v \in V$. This condition directly mirrors the requirement for a dominating set in the graph, where each vertex v is either in the dominating set or is adjacent to a vertex that is in the set. Thus, solving the Hitting Set problem on the hypergraph H corresponds to finding a dominating set in the original graph G.

An example of this translation can be seen in Figure 4.1, where a specific Dominating Set instance is transformed into a Hitting Set instance on a hypergraph.



(a) Graph representation



(b) Hypergraph representation

Figure 4.1: (a) Undirected graph where vertex 1 has edges to vertices 2 and 3. The set $\{1\}$ is a dominating set, covering all other vertices, marked in red. (b) Corresponding hypergraph, where vertex 1 hits every hyperedge, making $\{v1\}$ a hitting set, marked in red.

4.2 Reductions

In this section, we introduce various data reduction rules for the HITTING SET problem. We use data reduction rules to simplify the hypergraph and make processing the HITTING SET problem easier. All rules are designed to reduce the hypergraph while keeping the optimality of the solution.

All rules were inspired by the work of van Rooij and Bodlaender [36] on the SET COVER problem. We will reformulate them for the HITTING SET problem. There will be an intuition and a small example for each rule. The reduction rules are introduced using a standardized scheme shown in Reduction 0.

Reduction 0 ([Reduction Name], [Figure])
Description of the pattern that can be reduced.

Reduced Hypergraph	*How to build the reduced hypergraph H'*
Offset	*How much can be added to the offset*
Reconstruction	*How to reconstruct the solution S for the original hypergraph given the solution S' on the reduced graph H'*

We start by providing the name and the corresponding figure for the reduction rule. Then, we define the pattern the rule can reduce. Lastly, we give details on how the actual reduction is performed. This information is composed of three parts: First is information on how to construct the reduced hypergraph H'. Secondly, the *offset* describes the difference between the size of a minimum hitting set on the reduced hypergraph $|S'|$ and the size of the minimum hitting set on the original hypergraph $|S|$. Third is the information

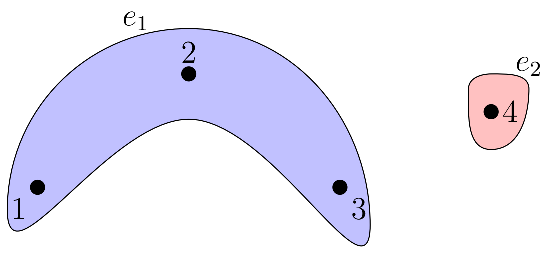


Figure 4.2: Example instance where we can apply the Isolated Vertex rule. Hyperedge e_2 can only be hit by vertex 4, hence we need to include vertex 4 in the hitting set.

on how to reconstruct the solution S for the original hypergraph from the solution S' on the reduced hypergraph.

Reduction 1 (Isolated Vertex, Figure 4.2)
Let $F \in \mathcal{F}$ with $F = \{v\}$. Then include v in the minimum hitting set.

Reduced Hypergraph	$H' = H - \{v\}$
Offset	$\lvert S\rvert = \lvert S'\rvert + 1$
Reconstruction	$S = S' \cup \{v\}$

If there is a hyperedge $F \in \mathcal{F}$ that only contains one vertex v, then we need to include v in the hitting set to be able to hit F.

Reduction 2 (Single Edge, Figure 4.3)
Let $F \in \mathcal{F}$ with $F = \{u, v\}$. Let $F(u) \subseteq F(v)$. Then include v in the minimum hitting set.

Reduced Hypergraph	$H' = H - \{u, v\}$
Offset	$\lvert S\rvert = \lvert S'\rvert + 1$
Reconstruction	$S = S' \cup \{v\}$

If there is a hyperedge $F \in \mathcal{F}$ that contains exactly two vertices u, v, then we need to include either v or u in the hitting set in order to hit the hyperedge F. However, if v hits all hyperedges that u hits, i.e. $F(u) \subseteq F(v)$, then including v is always better: Whenever we take u in the hitting set, we could equally well have taken v in order to possibly hit more hyperedges, since $|F(v)| \geq |F(u)|$.

Reduction 3 (Counting, Figure 4.4)
Let $v \in V$ with $|(\bigcup_{F \in F(v), |F|=2, u \in F} F(u)) \setminus F(v)| < |\{F \in F(v) || F| = 2\}|$. Then include v in the minimum hitting set.

Reduced Hypergraph	$H' = H - \{v\}$
Offset	$\lvert S\rvert = \lvert S'\rvert + 1$
Reconstruction	$S = S' \cup \{v\}$

For any vertex $v \in V$, let s be the number of vertices that share a hyperedge of size 2

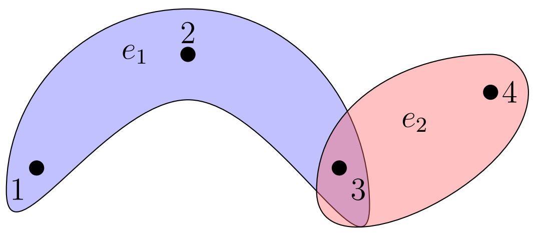


Figure 4.3: Example instance where we can apply the Single Edge rule. Edge e_2 only contains vertices 3 and 4, while vertex 3 hits every hyperedge that vertex 4 hits. Therefore, it is always better to include 3 in the hitting set.

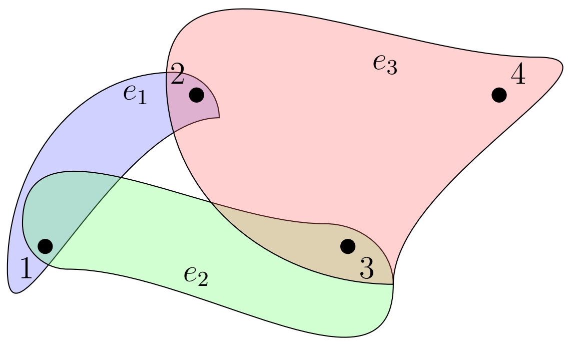


Figure 4.4: Example instance where we can apply the Counting rule. For vertex 1, $q = 1$ and $s = 2$, therefore $q < s$ holds. Thus, we can include vertex 1 in the hitting set.

with v, i.e. $F \in \mathcal{F}$ with $F = \{u, v\}$. Then, let q be the number of hyperedges which are getting hit by a vertex u that shares such a hyperedge $F = \{u, v\}$ with v but are not getting hit by v itself, i.e. $q = |(\bigcup_{F \in F(v), |F|=2, u \in F} F(u)) \setminus F(v)|$. In this situation, we hit $|F(v)|$ hyperedges using one vertex if we include v into the hitting set, and we hit $q + |F(v)|$ hyperedges using s vertices if we decide not to include v. Therefore, if $q < s$, including v is always at least as good as not including it since we then use $s - 1$ vertices less whilst hitting $q \leq s - 1$ less hyperedges. We are always able to cover these hyperedges with one additional vertex per hyperedge.

Reduction 4 (Vertex Domination, Figure 4.5)
Let $u, v \in V$ with $F(u) \subseteq F(v)$. Then remove u from the hypergraph.

Reduced Hypergraph	$H' = H - \{u\}$
Offset	$|S| = |S'|$
Reconstruction	$S = S'$

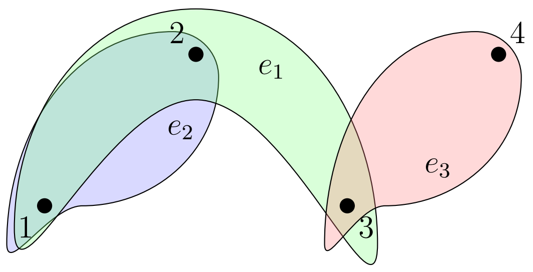


Figure 4.5: Example instance where we can apply both the (a) Vertex Domination rule and (b) Edge Domination rule. (a) Vertex 2 is dominating vertex 1 and vertex 3 is dominating vertex 4. Therefore, we can remove both vertex 1 and 4. (b) Hyperedge e_1 dominates hyperedge e_2. Hence, we can remove hyperedge e_1 from the hypergraph.

For any two vertices $u, v \in V$, we say v *dominates* u if v hits every hyperedge that u hits, i.e. $F(u) \subseteq F(v)$. In such event, if either vertex is needed in a minimum hitting set, it is always better to include v. This is because v hits all hyperedges that u hits and possibly more. Therefore, we can remove u.

At its core, the Single Edge rule is a special case of vertex domination that provides more information. If there is a hyperedge $F \in \mathcal{F}$ which only contains v and u, with u getting dominated by v, then we can immediately include v in the minimum hitting set. This is the case, because either u or v are needed to hit F and we just established that v is always better to include.

Reduction 5 (Edge Domination, Figure 4.5)
Let $F, E \in \mathcal{F}$ with $F \subseteq E$. Then remove E from the hypergraph.

Reduced Hypergraph	$H' = H - \{E\}$
Offset	$\lvert S\rvert = \lvert S'\rvert$
Reconstruction	$S = S'$

When considering two hyperedges $E, F \in \mathcal{F}$, we say E *dominates* F if E covers all vertices that F covers, i.e. $\forall v \in F : v \in E$. In this situation, if we chose a vertex v in order to hit F, v will also always hit E. Therefore, we do not care about E when looking for a minimum hitting set and are able to remove E.

4.3 Solving Process

In the following section, we present encodings of the HITTING SET problem for both Integer Linear Programming (ILP) and Weighted Partial MaxSAT (WPMS). We also outline the complete solving pipeline at the end.

4.3.1 Integer Linear Programming

Given a HITTING SET instance, we encode it into an Integer Linear Program as follows:

For each vertex v in the hypergraph $H = (V, \mathcal{F})$, we assign a decision variable x_v, where $x_v = 1$ (resp. $x_v = 0$) indicates that v is included (resp. not included) in a minimum hitting set. Each hyperedge $F \in \mathcal{F}$ results in one constraint in the ILP. In order to satisfy a constraint of a hyperedge F, the decision variable of at least one vertex that hits F must be set to 1. Overall, we want to then minimize the number of vertices included in the minimum hitting set, i.e. we want to minimize the sum over all decision variables. A minimum hitting set is then given by the set $S = \{v | x_v = 1\}$. For any given hypergraph $H = (V, \mathcal{F})$, we end up with the following formulation:

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} x_v \\ \text{subject to} \quad & \sum_{v \in F} x_v \geq 1 \quad \forall F \in \mathcal{F} \\ & x_v \in \{0, 1\} \quad \forall v \in V \end{aligned}$$

4.3.2 Weighted Partial Maximum Satisfiability

Given a HITTING SET instance on a hypergraph $H = (V, \mathcal{F})$, we encode it into a Weighted Partial Maximum Satisfiability instance as described in the following:

As for the ILP formulation, we assign a decision variable x_v for each vertex $v \in V$ that signals whether or not v is used in the solution. For each hyperedge $F \in \mathcal{F}$, we obtain one hard clause in order to make sure every hyperedge is hit by at least one vertex. Finally, for each vertex $v \in V$, generate a soft clause $\neg x_v$ to ensure as few variables are set to 1 as possible. Since we consider DOMINATING SET and HITTING SET instances on unweighted graphs, we set the weights of the soft clauses uniformly to 1.

4.3.3 Pipeline

Given a graph $G = (V, E)$, the goal is to find a minimum dominating set D. To solve this, we first transform the problem into an instance of the HITTING SET problem by constructing a corresponding hypergraph H based on the structure of G. Once the transformation is complete, we apply a set of five reduction rules in the same order as presented in Section 4.2, resulting in a reduced hypergraph H'. After simplification, we encode the reduced instance either as an Integer Linear Program (ILP) or a Weighted Partial MaxSAT (WPMS) formula. Solving this encoding yields a minimum hitting set S' for the reduced graph. We then apply the reconstruction steps of the reductions to obtain the minimum hitting set for the original graph H. This minimum hitting set S corresponds to a minimum dominating set D since we work with the same vertex set. Finally, we return D as the solution.

CHAPTER 5

Experimental Evaluation

All experiments were performed on a machine equipped with an 12th Gen Intel(R) Core(TM) i7-12700K @ 3.60 GHz. All code was written in `C++` and compiled with `g++` with the following parameters: `-O3 -std=c++17`. Due to the restrictions of the PACE Challenge, all algorithms must run strictly sequentially.

5.1 Dataset

All experiments were conducted on the full set of 100 exact public DOMINATING SET instances of the PACE Challenge 2025 [12]. We split the dataset into two sections, instances 1 to 50 and 51 to 100, since at the beginning only the first 50 instances were available. For the initial portfolio approach, we extracted multiple instance properties to analyze which algorithm performs well on which kind of graphs. The examined properties are shown in Table 5.1.

Symbol	**Description**
$\|\mathbf{V}\|$	Number of vertices in the graph
$\|\mathbf{E}\|$	Number of edges in the graph
$\boldsymbol{\rho}$	*Density* of the graph, computed as $\rho = \frac{2\|E\|}{\|V\|(\|V\|-1)}$ for undirected graphs
$\boldsymbol{\Delta}$	*Maximum degree*, defined as $\Delta = \max_{v\in V} \deg(v) = \max_{v\in V} \|N(v)\|$
Triangles	Number of triangles (fully connected subgraphs of three vertices)
d	*Average degree*, calculated as $d = \frac{2\|E\|}{\|V\|}$
$\boldsymbol{\sigma_d}$	*Standard deviation* of vertex degrees

Table 5.1: Summary of graph statistics

The properties for each instance can be seen in Table 5.2. First, we notice the following: There are many graphs sharing almost all properties, i.e. there are approx. 17 graphs that have exactly 8 340 vertices, 16 080 edges and 6 220 triangles. These graphs also have the same exact average degree, with the maximum degree varying between 30 and 36, and only slight changes in the standard deviation of the vertex degrees.

Other than that, approximately 80% of graphs have less than 10 000 vertices and less than 20 000 edges. Another almost 8% of graphs have more than 70 000 vertices and more than 700 000 edges. The standard deviation of most of these graphs is significantly elevated, usually more than 40 compared to most of the other graphs having a standard deviation of the vertex degree below 4. Additionally, about one third of the graphs have significantly more triangles than the rest, starting from circa 200 000 up to 3 198 008 triangles.

Most instances which neither fall in the group of sharing properties nor the instances with more than 70 000 triangles have a very low maximum degree, usually below 7. This is always paired with a lower amount of triangles as well, specifically under 2 000.

Lastly, we can spot three graphs that have a significantly higher average degree: exact_038.gr, exact_039.gr and exact_040.gr have average degree > 19, while every other graph has average degree < 6. Especially for exact_040.gr the standard deviation of the vertex degrees is also very high.

Table 5.2: Graph properties for graphs exact_001.gr to exact_100.gr.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	$\rho\times10^{-3}$	Δ	**Triangles**	d	σ_d
exact_001.gr	8 340	16 080	0.46	30	6 220	3.856	3.270
exact_002.gr	8 340	16 080	0.46	32	6 220	3.856	3.270
exact_003.gr	8 340	16 080	0.46	30	6 220	3.856	3.270
exact_004.gr	8 340	16 080	0.46	30	6 220	3.856	3.281
exact_005.gr	8 340	16 080	0.46	34	6 220	3.856	3.285
exact_006.gr	8 340	16 080	0.46	34	6 220	3.856	3.280
exact_007.gr	8 340	16 080	0.46	36	6 220	3.856	3.277
exact_008.gr	8 340	16 080	0.46	32	6 220	3.856	3.299
exact_009.gr	8 340	16 080	0.46	32	6 220	3.856	3.262
exact_010.gr	8 340	16 080	0.46	32	6 220	3.856	3.279
exact_011.gr	6 255	12 060	0.62	30	4 665	3.856	3.297
exact_012.gr	6 255	12 060	0.62	42	4 665	3.856	3.297
exact_013.gr	8 340	16 080	0.46	30	6 220	3.856	3.260
exact_014.gr	6 255	12 060	0.62	32	4 665	3.856	3.270
exact_015.gr	6 255	12 060	0.62	36	4 665	3.856	3.272
exact_016.gr	6 255	12 060	0.62	32	4 665	3.856	3.260
exact_017.gr	1 518	2 172	1.89	5	76	2.861	0.660
exact_018.gr	1 716	2 385	1.62	5	60	2.779	0.626
exact_019.gr	1 885	2 648	1.49	5	82	2.809	0.646
exact_020.gr	4 312	5 817	0.63	5	209	2.698	0.709

Continued on next page

Table 5.2 – continued from previous page

Graph	$\|V\|$	$\|E\|$	$\rho\times10^{-3}$	Δ	**Triangles**	d	σ_d
exact_021.gr	3 910	5 243	0.69	7	206	2.681	0.672
exact_022.gr	3 053	4 094	0.88	5	136	2.681	0.723
exact_023.gr	4 464	6 009	0.60	6	259	2.692	0.698
exact_024.gr	8 340	16 080	0.46	32	6 220	3.856	3.297
exact_025.gr	5 837	7 728	0.45	6	285	2.647	0.775
exact_026.gr	4 396	5 899	0.61	5	381	2.683	0.761
exact_027.gr	8 526	11 287	0.31	6	724	2.647	0.739
exact_028.gr	16 035	20 735	0.16	7	1 081	2.586	0.736
exact_029.gr	13 508	17 313	0.19	5	841	2.563	0.728
exact_030.gr	19 295	25 091	0.13	7	1 124	2.600	0.725
exact_031.gr	7 241	9 523	0.36	7	497	2.630	0.708
exact_032.gr	19 555	25 830	0.14	6	1 485	2.641	0.757
exact_033.gr	19 462	28 019	0.15	6	579	2.879	0.824
exact_034.gr	19 555	26 143	0.14	7	1 202	2.673	0.783
exact_035.gr	8 340	16 080	0.46	36	6 220	3.856	3.291
exact_036.gr	19 638	25 873	0.13	6	1 151	2.634	0.731
exact_037.gr	19 925	26 909	0.14	6	1 250	2.701	0.789
exact_038.gr	3 570	44 481	6.98	109	272 829	24.919	20.673
exact_039.gr	74 491	728 776	0.26	155	4 118 783	19.566	22.986
exact_040.gr	98 361	4 766 139	0.99	1 492	79 795 288	96.911	161.889
exact_041.gr	4 342	5 682	0.60	6	217	2.617	0.675
exact_042.gr	13 488	17 535	0.19	6	1 081	2.600	0.743
exact_043.gr	4 105	5 515	0.65	6	413	2.686	0.767
exact_044.gr	16 479	21 315	0.16	7	1 157	2.586	0.747
exact_045.gr	8 074	10 594	0.33	7	555	2.624	0.735
exact_046.gr	8 340	16 080	0.46	30	6 220	3.856	3.277
exact_047.gr	8 340	16 080	0.46	34	6 220	3.856	3.280
exact_048.gr	8 340	16 080	0.46	34	6 220	3.856	3.300
exact_049.gr	8 340	16 080	0.46	34	6 220	3.856	3.267
exact_050.gr	8 340	16 080	0.46	30	6 220	3.856	3.261
exact_051.gr	3 082	4 574	0.96	5	97	2.968	0.809
exact_052.gr	1 594	2 385	1.88	4	51	2.992	0.757
exact_053.gr	8 980	26 400	0.65	296	267 482	5.879	27.560
exact_054.gr	11 339	33 417	0.52	354	354 561	5.894	29.419
exact_055.gr	27 108	60 485	0.16	354	363 107	4.462	20.277
exact_056.gr	5 376	7 738	0.54	6	245	2.878	0.751
exact_057.gr	519 547	1 195 601	0.01	1 062	490 700	4.602	44.641
exact_058.gr	2 671	3 964	1.11	5	46	2.968	0.827

Continued on next page

Table 5.2 – continued from previous page

Graph	|V|	|E|	$\rho\times10^{-3}$	Δ	**Triangles**	d	σ_d
exact_059.gr	4 090	5 724	0.68	5	134	2.799	0.781
exact_060.gr	14 282	41 946	0.41	336	232 826	5.873	27.529
exact_061.gr	13 066	38 598	0.45	346	416 574	5.908	31.688
exact_062.gr	50 977	106 515	0.08	250	202 975	4.178	17.847
exact_063.gr	675 890	1 434 319	0.01	991	640 800	4.244	43.585
exact_064.gr	20 897	46 378	0.21	284	210 257	4.438	17.821
exact_065.gr	5 874	7 861	0.46	6	458	2.676	0.774
exact_066.gr	519 477	1 191 753	0.01	1 056	490 700	4.588	44.482
exact_067.gr	3 477	4 958	0.82	5	96	2.851	0.831
exact_068.gr	2 692	3 924	1.08	5	78	2.915	0.805
exact_069.gr	4 271	6 428	0.70	5	111	3.010	0.792
exact_070.gr	5 516	7 578	0.50	6	468	2.747	0.771
exact_071.gr	9 464	13 601	0.30	6	323	2.874	0.797
exact_072.gr	675 952	1 438 362	0.01	990	640 800	4.255	43.712
exact_073.gr	6 554	19 242	0.90	250	164 463	5.871	26.784
exact_074.gr	130 026	266 360	0.03	296	381 193	4.097	22.074
exact_075.gr	519 627	1 200 333	0.01	1 072	490 700	4.619	44.837
exact_076.gr	5 581	7 957	0.51	6	197	2.851	0.775
exact_077.gr	3 861	5 459	0.73	7	264	2.827	0.857
exact_078.gr	8 511	24 993	0.69	276	197 238	5.873	27.456
exact_079.gr	8 145	23 895	0.72	284	180 168	5.867	26.373
exact_080.gr	4 192	5 749	0.65	6	205	2.742	0.844
exact_081.gr	4 534	7 022	0.68	5	88	3.097	0.800
exact_082.gr	2 764	3 990	1.04	5	142	2.887	0.765
exact_083.gr	6 453	9 054	0.43	6	246	2.806	0.787
exact_084.gr	675 940	1 436 889	0.01	988	640 800	4.251	43.665
exact_085.gr	9 568	28 104	0.61	302	241 393	5.874	27.441
exact_086.gr	47 450	141 150	0.13	762	3 198 008	5.949	43.441
exact_087.gr	519 378	1 186 551	0.01	1 048	490 700	4.569	44.269
exact_088.gr	519 356	1 185 344	0.01	1 045	490 700	4.564	44.220
exact_089.gr	14 458	42 474	0.41	328	264 386	5.875	27.432
exact_090.gr	13 846	30 424	0.32	146	35 162	4.394	13.222
exact_091.gr	3 760	5 356	0.76	6	179	2.848	0.818
exact_092.gr	4 416	6 812	0.70	5	107	3.085	0.797
exact_093.gr	519 375	1 186 379	0.01	1 044	490 700	4.568	44.261
exact_094.gr	4 032	5 592	0.69	6	216	2.773	0.864
exact_095.gr	17 968	53 004	0.33	410	520 219	5.899	31.526
exact_096.gr	17 188	50 664	0.34	400	405 065	5.895	30.527

Continued on next page

Table 5.2 – continued from previous page

Graph	|V|	|E|	$\rho\times10^{-3}$	Δ	Triangles	d	σ_d
exact_097.gr	519 554	1 196 236	0.01	1 065	490 700	4.604	44.668
exact_098.gr	21 619	48 915	0.21	276	205 591	4.525	18.908
exact_099.gr	7 511	22 053	0.78	284	201 860	5.872	26.626
exact_100.gr	49 578	106 975	0.09	284	218 613	4.315	18.624

5.2 Results

In the next two sections, we will first examine the experimental results related to the reduction rules. This will include an analysis of how frequently reductions occurred and the computational time required to perform them. Following that, we will shift our focus to the solver performances, where we will explore the number of instances each solver successfully solved, along with the time it took to reach a solution.

5.2.1 Reductions

The reductions described in Section 4.2 were applied to each of the 100 exact instances. We additionally provide the time it took to complete the whole reduction process in seconds. The results can be seen in Table 5.3. The reduction rules were applied to the graphs in the following order:

1. Isolated Vertex rule
2. Single Edge rule
3. Counting rule
4. Edge Domination rule
5. Vertex Domination rule

The order in which the reductions were applied was chosen to prioritize efficiency, i.e. computationally inexpensive reductions, such as the removal of isolated vertices and single-edge components, were applied first. More computationally intensive reductions, like edge and vertex domination, were applied at the end. This ordering helps to streamline the overall reduction process by simplifying the graph before applying more complex operations.

In Table 5.3, it is immediately noticeable that none of the 100 instances contain any isolated vertices that can be reduced. Similarly, after removing the single-edge cases, the Counting rule does not identify any additional vertices for reduction. However, this does not imply that these two rules are redundant, as this observation may not hold for instances outside the PACE Challenge 2025 DOMINATING SET instances.

Overall, even though some graphs do not have single-edge cases, others greatly benefit from the rule. For graph exact_039.gr, we add 3554 vertices to the dominating set, thus reducing the instance by $7108 = 2 \times 3554$ vertices before more computationally intensive reductions kick in.

Additionally, it can be seen that both Edge and Vertex Domination rules are applicable for every single graph. For ca. 14% (resp. ca. 19%) of graphs we find below 100 edge (resp. vertex) domination cases. On the other hand, we find more than 4000 edge and vertex domination occurrences on about 54% of all instances.

For around 90% of the instances, we can finish the reduction process in under 60 seconds. For all instances where this is not the case, we observe a very large amount of domination occurrences, up to 1 281 600 edge domination and 640 800 vertex domination cases. It can be seen that exact_040.gr is an outlier: It takes over 4 000 seconds to complete the reduction process, way exceeding the time limit of 1 800 seconds defined by the PACE Challenge 2025 organizers. Looking at Table 5.2, we notice that exact_040.gr has by far the most amount of edges, about 4 times the edges of exact_063.gr, the graph with the second most edges. This is likely the cause for the outlier behavior. Every other instance can be reduced in under 900 seconds, thus it remains more than half of the initial time available for a solver to find a solution.

Lastly, it can be seen that, for the group of instances sharing the exact same amount of vertices and edges introduced in Section 5.1, they also share a very similar amount of edge and vertex domination cases, ca. 10 720 and 5 360 respectively. Additionally, they also have no single-edge cases. This further signifies their close relation observed in Section 5.1.

Table 5.3: Displays the number of reductions found for the graphs from exact_001.gr to exact_100.gr, along with the time taken to compute all reductions.

Graph	Isol. V.	Single E.	Count.	Edge Dom.	Vert. Dom.	Time(s)
exact_001.gr	0	0	0	10 720	5 360	3.63
exact_002.gr	0	0	0	10 728	5 364	3.62
exact_003.gr	0	0	0	10 720	5 360	3.64
exact_004.gr	0	0	0	10 720	5 360	3.70
exact_005.gr	0	0	0	10 726	5 363	3.63
exact_006.gr	0	0	0	10 720	5 360	3.61
exact_007.gr	0	0	0	10 720	5 360	3.64
exact_008.gr	0	0	0	10 722	5 361	3.63
exact_009.gr	0	0	0	10 722	5 361	7.22
exact_010.gr	0	0	0	10 720	5 360	3.61
exact_011.gr	0	0	0	8 044	4 022	2.04
exact_012.gr	0	0	0	8 040	4 020	2.06
exact_013.gr	0	0	0	10 722	5 361	7.22

Continued on next page

Table 5.3 – continued from previous page

Graph	Isol. V.	Single E.	Count.	Edge Dom.	Vert. Dom.	Time(s)
exact_014.gr	0	0	0	8 044	4 022	2.05
exact_015.gr	0	0	0	8 040	4 020	2.04
exact_016.gr	0	0	0	8 042	4 021	5.67
exact_017.gr	0	42	0	39	22	0.19
exact_018.gr	0	33	0	41	22	0.24
exact_019.gr	0	51	0	26	17	0.29
exact_020.gr	0	204	0	120	73	1.30
exact_021.gr	0	141	0	109	64	1.12
exact_022.gr	0	144	0	92	51	0.66
exact_023.gr	0	190	0	137	83	1.43
exact_024.gr	0	0	0	10 722	5 361	3.64
exact_025.gr	0	353	0	170	101	2.19
exact_026.gr	0	226	0	203	117	1.27
exact_027.gr	0	477	0	453	267	4.87
exact_028.gr	0	1 016	0	675	389	19.82
exact_029.gr	0	896	0	560	310	11.45
exact_030.gr	0	1 100	0	770	443	24.13
exact_031.gr	0	371	0	314	178	3.57
exact_032.gr	0	1 199	0	887	514	28.18
exact_033.gr	0	773	0	298	166	30.39
exact_034.gr	0	1 134	0	767	433	28.30
exact_035.gr	0	0	0	10 722	5 361	3.65
exact_036.gr	0	1 027	0	708	415	25.90
exact_037.gr	0	1 166	0	758	434	25.75
exact_038.gr	0	78	0	1 319	741	3.63
exact_039.gr	0	3 554	0	4 457	18 296	676.96
exact_040.gr	0	2 589	0	2 424	20 994	4 553.66
exact_041.gr	0	193	0	160	93	1.32
exact_042.gr	0	901	0	677	398	11.42
exact_043.gr	0	245	0	241	142	1.12
exact_044.gr	0	1 098	0	668	388	16.92
exact_045.gr	0	457	0	345	197	4.27
exact_046.gr	0	0	0	10 720	5 360	7.22
exact_047.gr	0	0	0	10 722	5 361	3.62
exact_048.gr	0	0	0	10 722	5 361	3.64
exact_049.gr	0	0	0	10 720	5 360	3.67
exact_050.gr	0	0	0	10 720	5 360	7.20
exact_051.gr	0	87	0	50	29	0.57

Continued on next page

Table 5.3 – continued from previous page

Graph	Isol. V.	Single E.	Count.	Edge Dom.	Vert. Dom.	Time(s)
exact_052.gr	0	41	0	13	8	0.16
exact_053.gr	0	0	0	17 600	8 800	0.97
exact_054.gr	0	0	0	22 278	11 139	1.46
exact_055.gr	0	443	0	38 970	19 513	22.05
exact_056.gr	0	169	0	97	56	1.67
exact_057.gr	0	0	0	981 400	490 700	639.14
exact_058.gr	0	88	0	11	7	0.43
exact_059.gr	0	183	0	56	38	0.92
exact_060.gr	0	0	0	27 964	13 982	1.97
exact_061.gr	0	0	0	25 732	12 866	1.79
exact_062.gr	0	182	0	87 953	43 982	62.77
exact_063.gr	0	0	0	1 281 600	640 800	851.76
exact_064.gr	0	191	0	31 277	15 643	11.73
exact_065.gr	0	372	0	231	148	1.71
exact_066.gr	0	0	0	981 400	490 700	630.29
exact_067.gr	0	170	0	57	37	0.65
exact_068.gr	0	91	0	36	23	0.44
exact_069.gr	0	96	0	58	32	1.14
exact_070.gr	0	273	0	269	164	1.66
exact_071.gr	0	358	0	167	96	5.03
exact_072.gr	0	0	0	1 281 600	640 800	857.71
exact_073.gr	0	0	0	12 828	6 414	0.53
exact_074.gr	0	215	0	244 855	122 432	347.72
exact_075.gr	0	0	0	981 400	490 700	645.22
exact_076.gr	0	223	0	73	44	1.74
exact_077.gr	0	184	0	103	63	0.83
exact_078.gr	0	0	0	16 662	8 331	0.81
exact_079.gr	0	0	0	15 930	7 965	0.76
exact_080.gr	0	235	0	76	52	0.90
exact_081.gr	0	86	0	37	23	1.32
exact_082.gr	0	89	0	77	44	0.44
exact_083.gr	0	280	0	121	78	5.85
exact_084.gr	0	0	0	1 281 600	640 800	857.98
exact_085.gr	0	0	0	18 736	9 368	1.03
exact_086.gr	0	0	0	94 100	47 050	25.16
exact_087.gr	0	0	0	981 400	490 700	626.40
exact_088.gr	0	0	0	981 400	490 700	730.10
exact_089.gr	0	0	0	28 316	14 158	1.95

Continued on next page

Table 5.3 – continued from previous page

Graph	Isol. V.	Single E.	Count.	Edge Dom.	Vert. Dom.	Time(s)
exact_090.gr	0	58	0	21 259	10 652	5.48
exact_091.gr	0	166	0	76	48	0.79
exact_092.gr	0	91	0	55	34	1.24
exact_093.gr	0	0	0	981 400	490 700	733.67
exact_094.gr	0	228	0	74	46	0.84
exact_095.gr	0	0	0	35 336	17 668	3.05
exact_096.gr	0	0	0	33 776	16 888	2.72
exact_097.gr	0	0	0	981 400	490 700	637.38
exact_098.gr	0	207	0	33 216	16 613	15.96
exact_099.gr	0	0	0	14 702	7 351	0.71
exact_100.gr	0	38	0	91 036	45 521	50.12

5.2.2 Solvers

In this section, we first compare the completion times of the following five solvers on the first 50 instances:

- FINDMINHS [5]: A dedicated Branch-and-Bound hitting set solver by Bläsius et al from the year 2023. We used the example settings provided on the GitHub [1].
- SCIP Optimization Suite [7]: An open source ILP solver often used in previous PACE challenges [11, 10].
- HiGHS [33, 19]: Another open source ILP solver which was popular in previous PACE competitions [38].
- Gurobi [13]: A commercial ILP solver. The setting `Threads=1` was used in order to force sequential execution.
- UWrMaxSat [27]: An open source MaxSAT solver. One of the best performing solvers on the unweighted instances of the MaxSAT Evaluation 2024 [3]. The standard settings provided in the README.md file of the solver were used [2].

These first 50 instances are useful for our analysis because they allow us to quickly determine which solvers are competitive. By evaluating all the solvers on these instances, we can avoid wasting computational resources on those that are clearly underperforming. This enables us to focus on analyzing variants of the most promising solvers for the remaining graphs, ensuring that we prioritize approaches with the highest potential for success.

[1]https://github.com/Felerius/findminhs
[2]https://maxsat-evaluations.github.io/2024/descriptions.html

Each solver was given a time-limit of 1 800 seconds to obtain a minimum dominating set, as required by the PACE Challenge. We ran every solver, except FINDMINHS, twice: Once with the use of reductions and once without. Running FINDMINHS with reductions was not necessary, since the original authors were already using all of the reductions applicable on these instances. By SOLVERNAME+R we refer to any given solver combined with the aforementioned reduction process.

The results without reductions are shown in Table 5.4. We immediately notice that the dedicated hitting set solver cannot compete with the other solvers: FINDMINHS times out for every of the first 50 graphs. It can also be seen that the open source ILP solvers, SCIP and HiGHS, cannot compete with the open source MaxSAT solver, UWrMaxSAT, on these instances. SCIP manages to solve 17, while HiGHS only manages to solve 9 out of 50 instances. Additionally, on all instances solved by an open source ILP, except exact_039.gr, UWrMaxSat finds a minimum dominating set significantly faster. All instances solved by an open source ILP, mostly taking more than 100 seconds, can be solved by UWrMaxSat in under 60 seconds.

We also notice that UWrMaxSat is very competitive with Gurobi, only solving one instance less on the first 50 instances. Except on 8 instances, UWrMaxSat finds a solution more quickly than Gurobi. On 12 instances we find a solution more than 30 seconds faster using UWrMaxSat instead of Gurobi. However, there are 7 instances, where Gurobi finds a solution 100 seconds sooner, oftentimes even up to 400 seconds.

Table 5.4: Solver times in seconds for the graphs exact_001.gr to exact_050.gr.

Graph	findminhs	SCIP	HiGHS	Gurobi	UWrMaxSat
exact_001.gr	-	-	-	13.15	0.43
exact_002.gr	-	-	-	11.79	0.14
exact_003.gr	-	-	-	54.18	0.04
exact_004.gr	-	-	-	14.82	0.39
exact_005.gr	-	-	-	94.94	0.06
exact_006.gr	-	-	-	8.11	0.54
exact_007.gr	-	-	-	126.73	0.68
exact_008.gr	-	-	-	47.25	0.04
exact_009.gr	-	-	-	197.88	0.66
exact_010.gr	-	-	-	1.1	0.37
exact_011.gr	-	-	-	3.14	0.83
exact_012.gr	-	-	441.81	2.11	0.97
exact_013.gr	-	-	-	3.15	0.11
exact_014.gr	-	-	-	6.67	0.70
exact_015.gr	-	-	-	32.22	0.07
exact_016.gr	-	-	-	14.97	0.14

Continued on next page

Table 5.4 – continued from previous page

Graph	findminhs	SCIP	HiGHS	Gurobi	UWrMaxSat
exact_017.gr	-	363.21	-	156.83	602.78
exact_018.gr	-	242.04	-	193.91	641.80
exact_019.gr	-	296.12	-	39.79	657.55
exact_020.gr	-	-	-	26.2	4.02
exact_021.gr	-	1 139.69	-	92.52	210.13
exact_022.gr	-	134.79	1 663.7	54.42	1.26
exact_023.gr	-	141.17	1 491.72	16.7	1.40
exact_024.gr	-	-	-	33.52	0.89
exact_025.gr	-	-	-	135.29	-
exact_026.gr	-	368.61	-	183.62	725.72
exact_027.gr	-	1 410.82	-	66.65	1 719.35
exact_028.gr	-	-	-	8.94	5.21
exact_029.gr	-	376.62	-	10.03	3.51
exact_030.gr	-	839.21	-	19.33	13.20
exact_031.gr	-	-	-	87.22	104.52
exact_032.gr	-	-	-	37.73	11.93
exact_033.gr	-	-	-	-	-
exact_034.gr	-	-	-	-	-
exact_035.gr	-	-	-	10.65	0.44
exact_036.gr	-	-	-	120.57	46.83
exact_037.gr	-	-	-	-	-
exact_038.gr	-	14.86	104	3.78	455.19
exact_039.gr	-	2.74	2.5	5.02	6.19
exact_040.gr	-	12.9	14.85	9.36	33.29
exact_041.gr	-	67.9	345.64	4.38	0.11
exact_042.gr	-	243.64	-	6.83	4.86
exact_043.gr	-	17.1	123.11	1.27	3.16
exact_044.gr	-	-	-	13.03	7.63
exact_045.gr	-	61.35	1 239.73	3.41	1.99
exact_046.gr	-	-	-	15	0.98
exact_047.gr	-	-	-	220.22	0.08
exact_048.gr	-	-	-	121.05	0.35
exact_049.gr	-	-	-	266.23	0.82
exact_050.gr	-	-	-	8.47	0.43

Looking at Figure 5.1, the discussed results are verified. SCIP outperforms HiGHS, while Gurobi and UWrMaxSat significantly outperform both SCIP and HiGHS. It is also noticeable that UWrMaxSat solves around 37 instances in a very short time, outperforming Gurobi at the beginning. UWrMaxSat and Gurobi are equally as fast at solving the first 39 instances. Gurobi then solves all remaining instances, for which it is able to find a solution for within the time-limit, in under 260 seconds. UWrMaxSat slowly catches up, with a small bump at around 600 seconds.

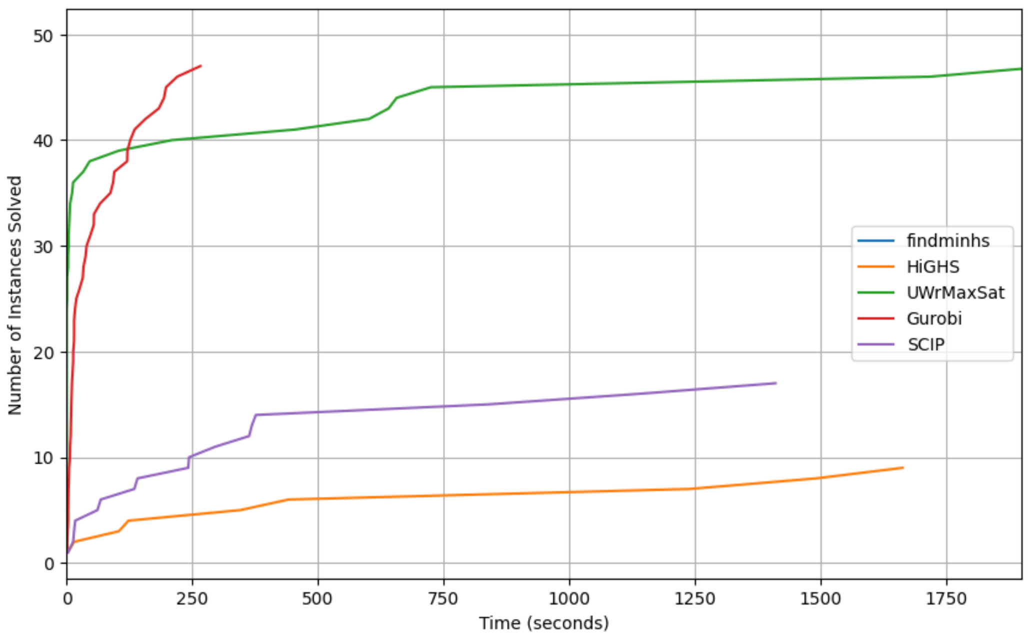


Figure 5.1: Plot of the solver performances on the graphs exact_001.gr to exact_050.gr. We plot the cumulative number of instances solved over time, with the x-axis representing the time in seconds.

In Table 5.5, we present the completion times for the first 50 instances, but this time for the different solvers after the reduction process has been applied. We do this in order to assess the impact of the reduction process on the different solvers and to determine which solvers benefit the most from the reduction. UWr+R refers to the UWrMaxSat solver together with all reductions, while UWr+Re (resp. UWr+Rv) refer to applying all reductions except edge (resp. vertex) domination before using the UWrMaxSat solver. This was done in order to further analyze how the different reductions influence the overall solving time.

We first recognize that no solver is able to solve instance 40 when using reduction rules. This is due to the reduction process taking longer than the time-limit of 1 800 seconds, as mentioned in Section 5.2.1.

We also notice that SCIP+R only manages to solve 14, while SCIP was able to solve 17 instances. With the use of reductions, SCIP+R solves instance 16 now, but is unable to solve instances 27, 30, 39, and 40 anymore. Especially for instance 40, but possibly also for instance 39, this is caused by the significant amount of time used in order to apply the reduction rules. We find that the amount of time needed to find a solution via SCIP+R is sometimes very different when compared to SCIP. For instances 17, 18, and 19, SCIP+R finds a solution about 100 seconds sooner. For instance 21, there is a difference of about 500 seconds. However, on instance 29, SCIP is ca. 300 seconds faster than SCIP+R. When looking at Table 5.3, we notice that for the instances were SCIP+R performs better, we have a lower amount of reductions found, about 40 per applicable reduction. SCIP seems to perform better than SCIP+R when a larger amount of applicable reductions is found (approx. 400).

Like SCIP+R, HiGHS+R also solves one instance less than its non-reducing counterpart HiGHS. While HiGHS+R is able to find a solution for 14 and 16, it is unable to find a solution for instances 22, 40, and 45. When comparing the solver times of instances that both HiGHS and HiGHS+R find a solution on, we do not see anything noticeable. Even with reductions, SCIP+R still significantly outperforms HiGHS+R.

When looking at the results for Gurobi+R, it is noticeable that Gurobi does not really benefit from the reduction process for these graphs. Gurobi+R solves the same amount of instances as Gurobi on instances 1 to 50. However, there are circa 10 instances that Gurobi+R solves more than 60 seconds slower than plain Gurobi. A significant improvement is only noticeable for instances 47 and 49, being about 200 seconds faster each.

Moreover, we notice that UWr+R is able to solve all instances between 1 and 50 which plain UWrMaxSat found a solution on except instance 40, which was already discussed. UWr+R additionally finds a solution for exact_025.gr, which was the one instance that Gurobi was able to solve more than UWrMaxSat. Generally, we notice that UWr+R is at least 60 seconds faster on 6 instances, while in return being slower on 3 other instances compared to UWrMaxSat. One of these instances is exact_039.gr, for which we notice in Table5.3 that the slower time is due to the initial reduction finding time. We also recognize that all instances that were originally solved under 1 second by UWrMaxSat are now introduced with some overhead originating from the reduction computation time shown in Section 5.2.1.

Looking at the other two reduction variants of UWrMaxSat, UWr+Re and UWr+Rv, we notice some significant changes when comparing it to UWr+R. For instance, UWr+Rv finds solutions faster on instances 22, 26, and 39, whilst being slower on 32 and 41. For instances 22 and 26, we do not observe anything in Table 5.3 which could be the cause for the significant speedup. However, the same goes for the instances were UWr+Rv performs worse. UWr+Re on the other hand is approximately 200 seconds faster than UWr+R on 5 separate instances, while only being significantly worse on exact_022.gr. It is important to note, that both UWr+Re and UWr+Rv solve one more instance, namely exact_034.gr. In both cases, the solution was found in less than 160 seconds. Looking at the reduction stats of instance 34, we once again do not find anything that cause this behavior. The graph properties of instance 34 are also not suspicious.

Table 5.5: Solver times in seconds for the graphs exact_001.gr to exact_050.gr.

Graph	SCIP+R	HiGHS+R	Gurobi+R	UWr+R	UWr+Re	UWr+Rv
exact_001.gr	-	-	379.16	9.62	1.81	1.30
exact_002.gr	-	-	18.86	2.53	1.37	1.09
exact_003.gr	-	-	29.12	3.48	2.31	3.48
exact_004.gr	-	-	504.32	3.58	1.48	1.03
exact_005.gr	-	-	85.40	9.44	1.11	3.98
exact_006.gr	-	-	7.58	7.20	1.18	2.75
exact_007.gr	-	-	162.53	15.01	1.79	17.93
exact_008.gr	-	-	45.27	3.99	2.52	6.52
exact_009.gr	-	-	109.89	3.70	1.79	8.24
exact_010.gr	-	-	1.04	2.79	2.81	1.96
exact_011.gr	-	-	2.23	1.18	0.41	1.38
exact_012.gr	-	348.67	2.34	4.17	1.24	0.09
exact_013.gr	-	-	4.21	3.49	1.67	3.14
exact_014.gr	-	1 038.61	101.50	1.68	1.65	1.46
exact_015.gr	-	-	7.20	1.30	0.88	0.45
exact_016.gr	450.27	848.52	20.23	1.38	1.62	1.24
exact_017.gr	279.92	-	148.50	354.93	334.99	598.13
exact_018.gr	155.03	-	129.59	621.43	609.95	778.64
exact_019.gr	181.22	-	107.51	575.29	586.07	597.76
exact_020.gr	-	-	24.05	2.17	79.32	2.40
exact_021.gr	619.00	-	62.74	4.52	2.21	2.85
exact_022.gr	148.07	-	46.30	1.38	230.94	0.38
exact_023.gr	135.87	1 454.16	12.67	2.07	7.53	25.34
exact_024.gr	-	-	14.08	2.96	1.08	2.11
exact_025.gr	-	-	541.35	1 070.79	925.73	812.60
exact_026.gr	389.64	-	749.25	601.50	627.37	6.06
exact_027.gr	-	-	156.22	6.76	5.27	3.99
exact_028.gr	-	-	8.16	22.70	17.31	9.98
exact_029.gr	715.66	-	5.60	11.15	8.66	10.35
exact_030.gr	-	-	23.42	28.72	24.95	13.04
exact_031.gr	-	-	83.98	29.09	33.62	10.66
exact_032.gr	-	-	61.08	781.72	37.38	804.30
exact_033.gr	-	-	-	-	-	-
exact_034.gr	-	-	-	-	45.85	153.98
exact_035.gr	-	-	53.03	3.15	1.70	1.60
exact_036.gr	-	-	160.13	88.47	-	-
exact_037.gr	-	-	-	-	-	-

Continued on next page

Table 5.5 – continued from previous page

Graph	SCIP+R	HiGHS+R	Gurobi+R	UWr+R	UWr+Re	UWr+Rv
exact_038.gr	22.08	494.79	4.20	466.39	104.29	471.14
exact_039.gr	-	1.37	0.28	701.64	556.07	118.86
exact_040.gr	-	-	-	-	-	-
exact_041.gr	43.03	374.73	4.36	1.95	2.02	129.01
exact_042.gr	219.63	-	5.53	12.84	13.91	6.01
exact_043.gr	14.64	119.78	1.33	1.19	1.29	1.88
exact_044.gr	-	-	9.10	21.66	14.76	10.46
exact_045.gr	86.43	-	3.47	5.83	4.89	6.02
exact_046.gr	-	-	13.29	2.43	1.87	1.08
exact_047.gr	-	-	26.36	26.84	2.01	3.31
exact_048.gr	-	-	161.58	2.74	2.99	1.53
exact_049.gr	-	-	43.37	2.14	1.06	6.63
exact_050.gr	-	-	10.99	10.57	1.39	4.88

The plot of the solver performance with initial reductions in Figure 5.2 shows similar results to the previous plot. Once again, SCIP+R and HiGHS+R cannot compete with Gurobi+R and the UWrMaxSat variants. We once again see that the first about 37 instances are solved faster by any UWrMaxSat variant, until Gurobi+R overtakes at around the 40 instances mark. We then see Gurobi+R finishing all results by 750 seconds, which is significantly worse compared to the 260 seconds it took plain Gurobi. We can also notice that UWr+Re seems to mostly outperform the other variants. At the end, every UWrMaxSat variant ends up with the same amount of solved instances. Given the consistently better performance of the UWrMaxSat solver compared to all other open-source solvers, the portfolio approach initially considered was ultimately abandoned.

Table 5.6: Solver times in seconds for the graphs exact_051.gr to exact_100.gr.

Graph	Gurobi	Gurobi+R	UWrMaxSat	UWr+R
exact_051.gr	-	-	80.63	-
exact_052.gr	288.23	252.04	1 131.02	46.27
exact_053.gr	16.02	0.04	0.22	1.73
exact_054.gr	7.17	0.09	1.67	2.88
exact_055.gr	63.88	95.17	541.09	67.42
exact_056.gr	-	-	-	18.03
exact_057.gr	-	-	-	-
exact_058.gr	-	-	-	-
exact_059.gr	178.39	174.80	-	989.22

Continued on next page

Table 5.6 – continued from previous page

Graph	Gurobi	Gurobi+R	UWrMaxSat	UWr+R
exact_060.gr	137.17	2.68	1.50	2.03
exact_061.gr	18.63	0.16	1.20	3.64
exact_062.gr	336.32	588.10	-	-
exact_063.gr	-	-	-	-
exact_064.gr	191.28	167.66	4.54	907.22
exact_065.gr	57.54	45.72	839.58	527.80
exact_066.gr	-	-	-	-
exact_067.gr	-	-	5.01	-
exact_068.gr	418.30	401.32	791.80	807.69
exact_069.gr	-	-	-	-
exact_070.gr	178.39	155.14	1 308.34	261.15
exact_071.gr	-	-	-	-
exact_072.gr	-	-	-	-
exact_073.gr	1.82	0.07	0.95	1.33
exact_074.gr	63.67	139.84	11.09	364.10
exact_075.gr	-	-	-	-
exact_076.gr	-	-	-	-
exact_077.gr	201.82	388.91	30.84	108.60
exact_078.gr	10.42	0.17	0.80	1.72
exact_079.gr	10.68	0.07	1.53	1.96
exact_080.gr	218.29	269.28	379.37	1 353.55
exact_081.gr	-	-	-	-
exact_082.gr	30.61	37.17	6.82	6.61
exact_083.gr	-	-	1 289.85	-
exact_084.gr	246.27	-	-	-
exact_085.gr	8.99	0.05	0.57	1.70
exact_086.gr	154.86	0.40	14.05	27.23
exact_087.gr	146.81	-	-	-
exact_088.gr	-	-	-	-
exact_089.gr	66.38	0.32	1.29	3.08
exact_090.gr	262.25	280.64	30.23	65.10
exact_091.gr	190.65	160.15	1 078.28	65.63
exact_092.gr	-	-	-	-
exact_093.gr	-	1 050.45	-	-
exact_094.gr	75.00	99.80	713.36	8.37
exact_095.gr	116.57	0.22	2.21	4.26
exact_096.gr	42.57	0.92	1.93	3.64
exact_097.gr	-	-	-	-

Continued on next page

Table 5.6 – continued from previous page

Graph	Gurobi	Gurobi+R	UWrMaxSat	UWr+R
exact_098.gr	114.70	126.64	1.60	15.17
exact_099.gr	5.89	0.01	1.72	1.16
exact_100.gr	-	414.71	-	1 588.38

Because FINDMINHS, SCIP, and HiGHS exhibited poor performance on the first 50 graphs, our analysis of the remaining 50 graphs is limited to a direct comparison of Gurobi, Gurobi+R, UWrMaxSat, and UWr+R. Table 5.6 presents the results of this direct comparison.

When comparing these solvers, it is important to note the significantly longer pre-processing time needed on the graphs exact_050.gr to exact_100.gr. In Table 5.3, it can be seen that there are overall 11 graphs that require a pre-processing time of more than 60 seconds. This significantly affects both Gurobi+R and UWr+R in the following comparisons.

For Gurobi and Gurobi+R, we actually notice some changes in comparison to the first 50 graphs. Notably, we see some differences in which instances the two variants can solve: Overall, both approaches are able to solve 31 instances. However, while Gurobi+R is unable to solve instances 84 and 87, plain Gurobi cannot solve instances 93 and 100. When looking at Table 5.3, we notice that the reduction times for instance 84 and 87 are 857.98 and 626.40 seconds respectively. As noted above, this heavily impacts the ability of Gurobi+R finding a solution in the remaining time.

Moreover, Gurobi+R is comparably fast to Gurobi on these instances. While notably slower on graphs 62, 74, 77, and 80, Gurobi+R performs significantly better on instances 60, 86, 89, 95, and 96. It is important to recognize that instances 62 and 74 additionally have a long reduction pre-processing time of 62.77 and 347.72 seconds respectively, as can be seen in Table 5.3. In comparison to the previous 50 instances, Gurobi+R actually gains significant performance benefits from the reduction process.

Looking at UWrMaxSat and UWr+R, we observe that 25 instances can be solved by plain UWrMaxSat and 24 instances can be solved by UWr+R. UWr+R is able to solve instances 67 and 83, while UWrMaxSat cannot find a solution for instance 100. On these graphs, UWr+R is able to significantly improve solving time on only 4 instances. On the other hand, solving time is notably slower than plain UWrMaxSat on 4 instances as well.

Overall, we notice Gurobi is able to solve 6 instances more than UWrMaxSat while Gurobi+R is able to solve 7 instances more than UWr+R. Gurobi and Gurobi+R together are able to find a solution on 33 instances, where only 3 instances were solved neither by UWrMaxSat nor by UWr+R. On the other hand, UWrMaxSat and UWr+R together are also able to solve 33 instances, including 4 that neither Gurobi nor Gurobi+R could solve. Hence, when looking at the combined approach of using both solver variants, UWrMaxSat seems to benefit more from the reduction process.

Since UWrMaxSat and Gurobi delivered similarly strong performance on both instance sets, we now examine their overall performance in Figure 5.3. As before, we notice that the

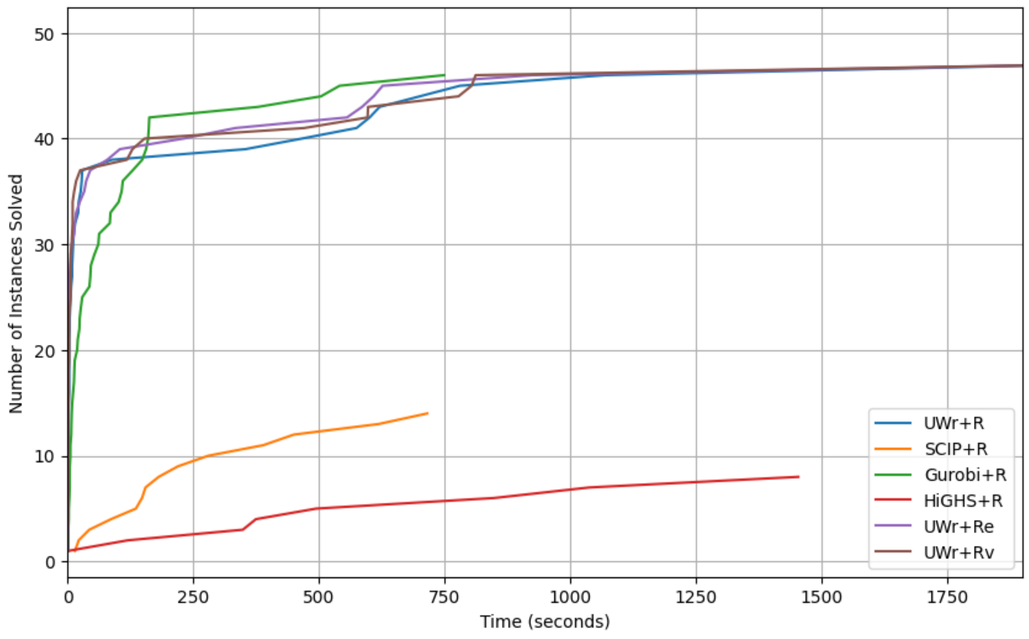


Figure 5.2: Plot of the solver performances on the graphs exact_001.gr to exact_050.gr, where reduction was applied beforehand. The plot shows the cumulative number of instances solved over time, with the x-axis representing time in seconds.

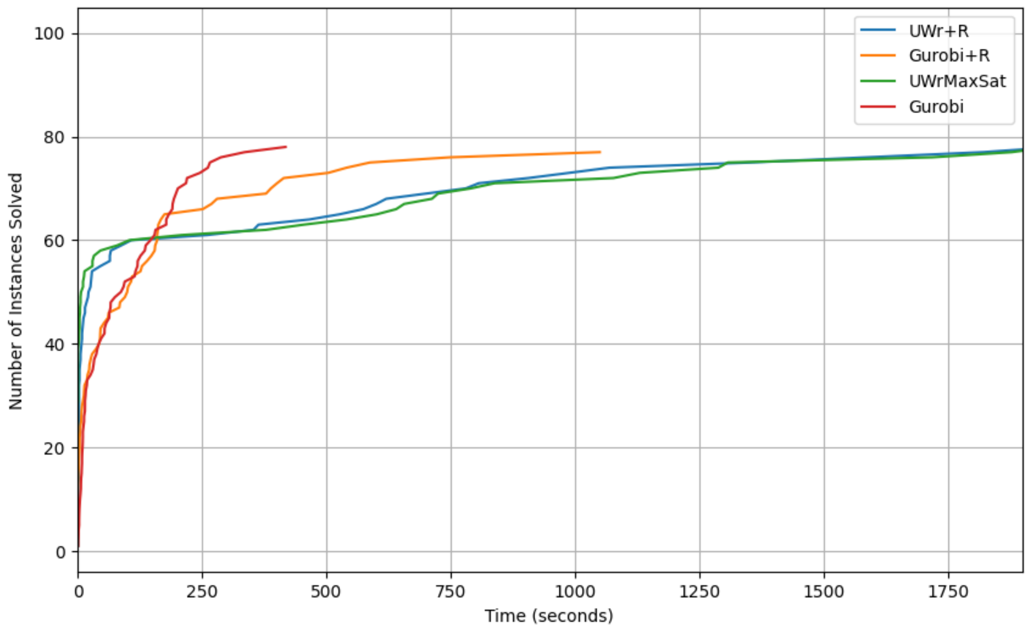


Figure 5.3: Plot of the solver performances on the graphs exact_001.gr to exact_100.gr, providing a direct comparison between Gurobi and UWrMaxSat. The plot shows the cumulative number of instances solved over time, with the x-axis representing time in seconds.

UWrMaxSat variants are able to solve significantly more instances in under 100 seconds than the Gurobi variants. We notice that all solvers are able to solve 60 instances in about 150 seconds. Thereafter, Gurobi and Gurobi+R both solve their remaining instances faster than UWrMaxSat and UWr+R, which slowly catch up in the end.

It is important to note that Gurobi is not only able to find solutions more quickly than Gurobi+R, but also solves more instances overall. This can mostly be attributed to the first 50 instances, as discussed above. When comparing UWrMaxSat to UWr+R, we observe that although UWrMaxSat performs better initially, UWr+R solves instances more quickly later on. In the end, both variants solve the same number of instances.

CHAPTER 6

Discussion

6.1 Conclusion

This thesis investigated the effectiveness of combining Integer Linear Programming (ILP) and MaxSAT solving techniques with a pre-processing reduction phase to address the DOMINATING SET and HITTING SET problems. The experimental results demonstrate that the proposed reduction techniques can significantly reduce problem sizes, possibly contributing to improved solver performance. Among the evaluated approaches, the dedicated hitting set solver FINDMINHS was unable to match the performance of ILP and MaxSAT-based methods. Notably, the open-source MaxSAT solver UWrMaxSat consistently outperformed the widely used open-source ILP solvers SCIP and HiGHS, successfully solving the majority of exact DOMINATING SET instances from this year's PACE Challenge. Although we originally planned to use a portfolio approach, the results showed that it is not feasible due to the significant performance difference between the open-source solvers. Furthermore, UWrMaxSat proved to be competitive even when compared to the commercial ILP solver Gurobi. While the application of reductions did not improve Gurobi's performance for most instances, UWrMaxSat benefited significantly from different reduction configurations, often resulting in faster solution times and a higher number of solved instances.

6.2 Future Work

Several promising directions for future work emerge from this thesis. A key opportunity lies in optimizing the reduction process. Currently, both edge and vertex domination rules are implemented in a relatively naive fashion. Enhancing the efficiency of these implementations could further improve the performance of UWrMaxSat when used in combination with reductions. Additionally, the approach presented here could be extended by identifying new reduction rules to reduce problem instances even further before solving.

Another interesting direction would be to explore time allocation strategies. Since solutions are typically found within approximately 1 100 seconds, splitting the total available time across multiple configurations of the UWrMaxSat solver might increase the likelihood of solving more instances within the time limit.

Lastly, given the strong performance of UWrMaxSat compared to open-source ILP solvers, it would be worthwhile to investigate other MaxSAT solvers featured in the MaxSAT Evaluation 2024 [3]. It is possible that some solvers can solve instances that UWrMaxSat misses, and vice versa, making a portfolio-based approach in this direction potentially beneficial.

Zusammenfassung

Diese Arbeit behandelt das Problem des DOMINATING SET und des HITTING SET, die beide Teil der PACE Challenge 2025 sind, und nutzt dafür bestehende Lösungstechniken. Da das DOMINATING SET auf das HITTING SET Problem reduziert werden kann, untersuchen wir den Einsatz eines spezialisierten Hitting Set Solvers sowie ILP- und MaxSAT-Ansätze zur Lösung von DOMINATING SET-Instanzen. Zur Verbesserung der Solver-Leistung wenden wir Datenreduktionsverfahren an, die darauf abzielen, die Gesamtgröße des Problems vor dem Lösen zu verringern. Unsere Ergebnisse zeigen, dass ein Open-Source MaxSAT-Solver deutlich bessere Ergebnisse liefert als gängige Open-Source ILP-Solver, während der spezialisierte Hitting Set-Solver mit keiner der beiden Methoden mithalten kann. Darüber hinaus stellen wir fest, dass der MaxSAT-Solver, kombiniert mit Reduktionen, sogar mit Gurobi, einem kommerziellen ILP-Solver, konkurrieren kann.

Bibliography

[1] Pace challenge. https://pacechallenge.org/, 2025. Accessed: 2025-04-19.

[2] Florent Avellaneda. Evalmaxsat. https://github.com/FlorentAvellaneda/EvalMaxSAT, 2024. Accessed: 2025-04-18.

[3] Jeremias Berg, Matti Järvisalo, Ruben Martins, Andreas Niskanen, and Tobias Paxian. Maxsat evaluation 2024. https://maxsat-evaluations.github.io/2024/index.html, 2024. Accessed: 2025-04-18.

[4] Armin Biere. Kissat sat solver. https://github.com/arminbiere/kissat, 2024. Accessed: 2025-04-18.

[5] Thomas Bläsius, Tobias Friedrich, David Stangl, and Christopher Weyand. An efficient branch-and-bound solver for hitting set. *CoRR*, abs/2110.11697, 2021. URL https://arxiv.org/abs/2110.11697.

[6] Jeremy Blum, Min Ding, Andrew Thaeler, and Xiuzhen Cheng. Connected dominating set in sensor networks and manets. *Handbook of Combinatorial Optimization: Supplement Volume B*, pages 329–369, 2005.

[7] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The SCIP Optimization Suite 9.0. Technical report, Optimization Online, February 2024. URL https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/.

[8] Alejandra Casado, Sergio Bermudo, AD López-Sánchez, and Jesús Sánchez-Oro. An iterated greedy algorithm for finding the minimum dominating set in graphs. *Mathematics and Computers in Simulation*, 207:41–58, 2023.

[9] Yibin Chen, Sean Safarpour, Andreas Veneris, and Joao Marques-Silva. Spatial and temporal design debug using partial maxsat. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, pages 345–350, 2009.

[10] Doblalex. Crgone, 2023. URL `https://github.com/Doblalex/CRGone`. Accessed: 2025-04-22.

[11] Rainer Grapa et al. pace-2022-dfvs-solver, 2022. URL `https://gitlab.informatik.uni-bremen.de/grapa/java/pace-2022-dfvs-solver`. Accessed: 2025-04-22.

[12] Mario Grobler. Pace2025-instances: Instances for the pace challenge 2025. `https://github.com/MarioGrobler/PACE2025-instances`, 2025. Accessed: 2025-04-18.

[13] Gurobi Optimization, LLC. Gurobi optimizer, 2024. URL `https://www.gurobi.com/`. Accessed: 2024-11-04.

[14] Juris Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90, 1982.

[15] Abdel-Rahman Hedar and Rashad Ismail. Hybrid genetic algorithm for minimum dominating set problem. In *Computational Science and Its Applications–ICCSA 2010: International Conference, Fukuoka, Japan, March 23-26, 2010, Proceedings, Part IV 10*, pages 457–467. Springer, 2010.

[16] Abdel-Rahman Hedar and Rashad Ismail. Simulated annealing with stochastic local search for minimum dominating set problem. *International Journal of Machine Learning and Cybernetics*, 3:97–109, 2012.

[17] Chin Kuan Ho, Yashwant Prasad Singh, and Hong Tat Ewe. An enhanced ant colony optimization metaheuristic for the minimum dominating set problem. *Applied Artificial Intelligence*, 20(10):881–903, 2006.

[18] Hao Hu, Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. Learning optimal decision trees with maxsat and its integration in adaboost. In *IJCAI-PRICAI 2020, 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence*, 2020.

[19] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. ISSN 1867-2957. doi: 10.1007/s12532-017-0130-5. URL `https://doi.org/10.1007/s12532-017-0130-5`.

[20] IBM. Cplex website, ibm products, 2024. URL `https://www.ibm.com/de-de/products/ilog-cplex-optimization-studio`. Accessed: 2024-11-04.

[21] Yoichi Iwata. A faster algorithm for dominating set analyzed by the potential method. In *Parameterized and Exact Computation: 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers 6*, pages 41–54. Springer, 2012.

[22] Hua Jiang and Zhifei Zheng. An exact algorithm for the minimum dominating set problem. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 5604–5612, 2023.

[23] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi: 10.1007/978-1-4684-2001-2_9. URL `https://doi.org/10.1007/978-1-4684-2001-2_9`.

[24] Laura L. Kelleher and Margaret B. Cozzens. Dominating sets in social network graphs. *Mathematical Social Sciences*, 16(3):267–279, 1988. ISSN 0165-4896. doi: https://doi.org/10.1016/0165-4896(88)90041-8. URL `https://www.sciencedirect.com/science/article/pii/0165489688900418`.

[25] Fabian Kuhn and Rogert Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 25–32, 2003.

[26] MaxSAT Evaluation. Maxsat evaluations, 2021. URL `https://maxsat-evaluations.github.io/`. Accessed: 2025-04-18.

[27] Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 132–136, 2020. doi: 10.1109/ICTAI50040.2020.00031.

[28] Günther R Raidl and Jakob Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. *Hybrid metaheuristics: An emerging approach to optimization*, pages 31–62, 2008.

[29] Mostafa Rezvani, Mohammad Kazem Akbari, and Bahman Javadi. Resource allocation in cloud computing environments based on integer linear programming. *The Computer Journal*, 58(2):300–314, 2015.

[30] Alexander Schidler. dfvs, 2022. URL `https://github.com/ASchidler/dfvs`. Accessed: 2025-04-22.

[31] Eike Schweissguth, Peter Danielis, Dirk Timmermann, Helge Parzyjegla, and Gero Mühl. Ilp-based joint routing and scheduling for time-triggered networks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pages 8–17, 2017.

[32] Hesamoddin Tahami and Hengameh Fakhravar. A literature review on combining heuristics and exact algorithms in combinatorial optimization. *European Journal of Information Technologies and Computer Science*, 2(2):6–12, 2022.

[33] HiGHS Development Team. Highs: High-performance parallel linear optimization software, 2025. URL `https://highs.dev/`. Accessed: 2025-04-18.

[34] TheoryInPractice. hydraprime, 2023. URL `https://github.com/TheoryInPractice/hydraprime`. Accessed: 2025-04-22.

[35] Massimo Tornatore, Guido Maier, and Achille Pattavina. Wdm network design by ilp models based on flow aggregation. *IEEE/ACM Transactions On Networking*, 15(3): 709–720, 2007.

[36] Johan M.M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, 2011. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2011.07.001. URL `https://www.sciencedirect.com/science/article/pii/S0166218X11002393`.

[37] Peng-Jun Wan, Khaled M Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9:141–149, 2004.

[38] Michael Wien. pingpong, 2023. URL `https://github.com/mwien/pingpong`. Accessed: 2025-04-22.