Engineering Hypergraph Independent Set Algorithms

Antonie Lea Wagner

March 30, 2025

4221674

Bachelor Thesis

at

Algorithm Engineering Group Heidelberg
Heidelberg University

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

Co-Supervisor:
Ernestine Großmann

Acknowledgments

I would like to express my sincere gratitude to Prof. Dr. Christian Schulz for his inspiring teaching and guidance, not only during this thesis, but throughout my entire studies. I am especially grateful to Ernestine Großmann for her continuous support and thoughtful advice. She was always generous with her time and offered help whenever I needed it. Lastly, I thank my family and friends for standing by me throughout this journey.

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, 31. März 2025

Antonie Lea Wagner

Abstract

This thesis addresses the well-known Maximum Independent Set (MIS) problem in the context of hypergraphs. While the MIS problem has been extensively studied on graphs, this work focuses on its strong extension to hypergraphs, where edges may connect any number of vertices. We propose a novel kernelization algorithm based on exact reduction rules, specifically designed for the hypergraph variant of the problem. To balance solution quality and runtime efficiency, we evaluate different configurations of the reduction rules through comprehensive tuning experiments involving parameter restrictions. The kernelization algorithm serves as a preprocessing step for subsequent solvers. We analyze its impact on two approaches: the commercial ILP solver Gurobi as an exact method, and a greedy algorithm as a heuristic. Our results demonstrate significant improvements in both runtime and solution quality. The kernelization reduces Gurobi's runtime by up to a factor of 10. On instances that cannot be solved within the time limit, it enables the solver to find independent sets up to 7% larger. In addition, one more instance can be solved to optimality within the time limit. For the heuristic approach, the kernelization leads to solution quality improvements of up to 50%.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The Maximum Independent Set (MIS) problem is one of the most extensively studied and fundamental NP-hard problems in combinatorial optimization. Given a graph $G = (V, E)$, a maximum independent set $\mathcal{I} \subseteq V$ is a set of pairwise non-adjacent vertices with maximum cardinality. The problem of finding an MIS in a graph has a wide range of applications across various domains. In cartography, it plays a crucial role in dynamic map labeling [23], which involves determining optimal placements for labels associated with geographic features. In computational biology, the MIS problem is utilized in DNA computing and nanotechnology applications [34, 45], where independent sets help model molecular self-assembly and sequence selection. Additionally, it serves as a powerful analytical tool in network science, particularly in the study of social interactions and influence within multilayer social networks [30].

While the MIS problem is traditionally studied in the context of graphs, many real-world problems exhibit higher-order relationships that cannot be captured by pairwise interactions alone. In such cases, hypergraphs provide a more expressive framework, allowing edges to connect multiple vertices simultaneously. Extending the MIS problem to hypergraphs leads to the problem of finding the largest subset of vertices such that no two vertices in the set belong to the same edge. While we specifically focus on the strong variant of the problem, an alternative extension is the weak variant, where an independent set is a set that does not fully contain any edge. As the MIS problem on hypergraphs remains NP-hard, exact algorithms quickly become infeasible for large instances.

A key strategy for tackling complex combinatorial optimization problems is the use of reduction algorithms. These techniques aim to simplify problem instances by removing redundant structures and reducing the input size while preserving the equivalence of the solution space. Reduction methods have proven to be successful for the MIS problem in graphs, particularly in the context of exact algorithms and

preprocessing for heuristics [10, 13, 32, 37, 38]. By identifying and eliminating fast solvable components, these approaches can significantly enhance the performance of solving methods, making them an essential tool for handling large-scale instances. This success motivates the use of reduction techniques as a preprocessing phase for solving the MIS problem in hypergraphs.

1.2 Our Contribution

In this work, we design a preprocessing algorithm to enhance the solvability of the MIS problem in hypergraphs by reducing problem complexity before applying subsequent solvers. To achieve this, we introduce a set of problem-specific exact data reduction rules by following two main strategies: adapting well-known reduction techniques from the MIS problem in graphs to the hypergraph context, and developing novel reductions specifically for the MIS problem in hypergraphs, inspired by related combinatorial problems such as the Hitting Set problem. The reduction rules are integrated into a preprocessing framework. In this thesis, we focus on optimizing its configuration to achieve the best performance. To evaluate the effectiveness of our approach, we analyze the impact of the reduction algorithm on the performance of subsequent solvers. For this purpose, we employ an integer linear programming (ILP) solver as an exact method and a greedy algorithm as a heuristic approach. We systematically compare both methods with and without the preprocessing stage to assess its influence on solution quality and computational efficiency. As part of our approach, we extract two distinct kernelization configurations:

KSTRONG, designed to produce a high-quality kernel, aims to enhance the solution quality of subsequent solvers. In 75% of the instances that the ILP solver Gurobi fails to solve within a 30-minute time limit, KSTRONG improves the solution by finding an independent set up to 7% larger. In addition, it enables Gurobi to solve one more instance to optimality within the time limit. The solution quality of the heuristic algorithm is improved in over 80% of the instances. With KSTRONG, it finds independent sets up to twice as large as those found by the heuristic alone. For about 20% of the instances, KSTRONG increases the independent set, found by the heuristic, by at least 10%.

The other configuration, KFAST, is optimized for speed. When combined with KFAST, over 30% of exactly solved instances experience a speedup of at least factor 2 compared to Gurobi alone. The maximum speedup achieved through kernelization reaches a factor of 10.

1.3 Structure

The remainder of this thesis is organized as follows. In Chapter 2, we introduce fundamental concepts related to the problem and provide a detailed definition of the Maximum Independent Set problem in hypergraphs. Chapter 3 presents an overview of the related work and the current state of research in this field. Our main contribution, including the problem-specific reduction rules and the complete preprocessing framework, is described in Chapter 4. Additionally, we explain the two solver methods used: an exact and a heuristic approach. Chapter 5 contains the experimental evaluation of our algorithm. Finally, in Chapter 6, we conclude this work with a discussion and an outlook on future research directions.

CHAPTER 2

Fundamentals

This chapter begins with an introduction of important terms and notations in Section 2.1. In Section 2.2, we provide a short explanation of the general concept of reduction rules, before defining the Maximum Independent Set problem on hypergraphs and its related problems in Section 2.3.

2.1 General Definitions

An *undirected hypergraph* $H = (V, E)$ is defined as a set of n vertices V and a set of m hyperedges E, also referred to as edges. Each edge $e \in E$ is a subset of the vertex set V, i.e. $E \subseteq 2^V$. The set of vertices of a hyperedge e is described with e and $|e|$ is the edge size. A vertex is *incident* to an edge e iff $v \in e$. The set of edges incident to a vertex v is denoted by $E(v) := \{e \in E \mid v \in e\}$. The degree of a vertex v is $d(v) := |E(v)|$ and the maximum degree of H is $\Delta_V := \max_{v \in V} d(v)$. A hypergraph is *r-uniform* if all edges have size r and it is *d-regular* if all vertices have degree d. Two vertices u and v are *adjacent* if both are incident to the same edge, i.e. $E(u) \cap E(v) \neq \emptyset$. The neighborhood $N(v) := \{u \in V \mid \exists e \in E : \{u, v\} \subseteq e\}$ of a vertex v is the set of vertices that are adjacent to v. More generally, for a subset of vertices $S \subseteq V$ the neighborhood is extended to $N(S) = \bigcup_{v \in S} N(v) \setminus S$. The same applies for the *closed* neighborhood $N[v] = N(v) \cup \{v\}$ and its extension $N[S] = N(S) \cup S$. The *two-neighborhood* $N^2(v) := N(N[v])$ of a vertex v is the set of all vertices $w \in V$ that share a common neighbor with v, but are not directly adjacent to v. Given a subset $V' \subset V$, the *subhypergraph* $H_{V'}$ is defined as $H_{V'} := (V', \{e \in E \mid e \cap V' \neq \emptyset\})$. If the subhypergraph contains only those hyperedges that are entirely contained within V', i.e. $E' = \{e \in E \mid e \subseteq V'\}$, we refer to it as an *induced subhypergraph* and denote it by $H[V']$. The removal of a vertex $v \in V$ is denoted as $H' = H - v$ instead of explicitly writing $H' = (V \setminus \{v\}, \{e \in E \mid e \setminus \{v\} \neq \emptyset\})$. The same notation applies for subsets $X \subseteq V$ with $H' = H - X$ instead of $H' = (V \setminus X, \{e \in E \mid e \setminus X \neq \emptyset\})$.

A *clique* is a set $C \subseteq V$ such that all vertices are pairwise adjacent. The trivial case of a clique in a hypergraph is a hyperedge.

An *undirected graph* $G = (V, E)$, with $n = |V|$ and $m = |E|$ is a 2-uniform hypergraph with $E \subseteq \binom{V}{2}$. A common method to transform an undirected hypergraph $H = (V, E)$ into an undirected graph $G = (V, E')$ is the *clique conversion*. It replaces the hyperedges of the hypergraph with cliques in the graph. That means for each hyperedge $e \in E$ and for every pair of vertices $u, v \in e$, an edge $\{u, v\}$ is inserted to the edge set E' of the graph.

2.2 Kernelization and Reduction Rules

In parameterized complexity, the concept of *fixed-parameter tractability* (FPT) is central to addressing NP-hard problems. Exact and deterministic algorithms for NP-hard problems require exponential time with respect to the total size of the input. In contrast, fixed-parameter tractable algorithms run in time that is only exponential in a fixed parameter k while remaining polynomial in the input size. A problem is classified as FPT if it can be solved in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, with the input size $|I|$, a parameter k and a computable function $f(k)$ solely depending on k. Consequently, it is polynomial in the parameter k and feasible to solve for small values of k.

Kernelization plays a fundamental role in fixed-parameter tractability. A kernelization algorithm is a polynomial-time preprocessing procedure that reduces the input size of the problem into a smaller, equivalent instance, ensuring that the solution space remains unchanged. The reduced problem instance is called a *kernel*. In the context of FPT, its size is bounded by a computable function which is only dependent on the parameter k. While FPT is primarily of theoretical interest, the concept of kernelization is widely applied in practice, even in cases where no explicit dependence on a parameter k exists.

To compute a kernel, so-called *data reduction rules* are applied. They are designed to identify and remove redundant structures, simplify constraints or decompose complex elements into manageable components, while maintaining equivalence to the input. For a given hypergraph H, a reduction rule transforms H in polynomial time into an equivalent instance H' such that $n' \leq n$ or $m' \leq m$. An optimal solution $\mathcal{I}'$ to the problem on the kernel H' can then be reconstructed to an optimal solution $\mathcal{I}$ for the original instance H by undoing the reductions. If the reduced instance H' is empty, the solution to the problem can be directly obtained by simply undoing the reductions.

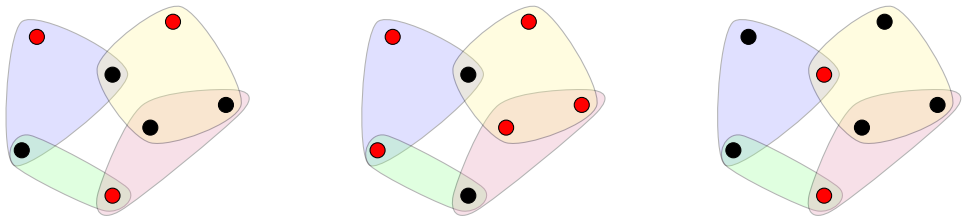
2.3 Problem Definition

A *weak independent set* in H is a set $\mathcal{I} \subseteq V$ that does not contain any edge of H, i.e. $\forall e \in E : |\mathcal{I} \cap e| < |e|$. In case $\mathcal{I}$ intersects any edge in E in at most one element, i.e. $\forall e \in E : |\mathcal{I} \cap e| \leq 1$, it is said to be a *strong independent set*. In the context of graphs, there is no distinction between a weak and a strong independent set. In this thesis, we focus only on strong independent sets. For simplicity, we refer to them as independent sets (IS) throughout this thesis. An IS $\mathcal{I}$ is *maximal*, if there exists no other IS $\mathcal{I}'$ with $\mathcal{I} \subsetneq \mathcal{I}'$. That is, no vertex $v \in V \setminus \mathcal{I}$ can be added to $\mathcal{I}$ without violating the independence property. The *size* of an IS is the number of vertices it contains. With $\mathrm{IS}(H)$ we denote the set of all independent sets of H.

A *maximum independent set* (MIS) $\mathcal{I} \in \mathrm{IS}(H)$ is an independent set with maximum cardinality. The size of the maximum independent set is the *independence number* of H and denoted by $\alpha(H)$. Note that there may exist several independent sets of H with size $\alpha(H)$. The *Maximum Independent Set problem* on hypergraphs asks for an MIS in a given unweighted, undirected hypergraph. The solution to the MIS problem on a hypergraph remains the same when transforming the hypergraph to a graph using the clique conversion.

A *maximum weighted independent set* (MWIS) is defined on a weighted hypergraph $H = (V, E, \omega)$, where $\omega : V \to \mathbb{R}_{\geq 0}$ is a weight function assigning weights to the vertices. An MWIS is an independent set $\mathcal{I} \subseteq V$ that maximizes the total weight $\omega(\mathcal{I}) = \sum_{v \in \mathcal{I}} \omega(v)$.

A *hitting set* is a subset $S \subseteq V$ that intersects ("hits") every edge in H, i.e. $e \cap S \neq \emptyset$ for all $e \in E$. It is known as a *vertex cover* in the context of graphs. A *d-hitting set* ensures that the size of every hyperedge e is upper-bounded by a fixed number d, making the problem fixed-parameter tractable. The *Hitting Set problem* asks for a minimum hitting set in a hypergraph. It is closely related to the MIS problem on hypergraphs when considering *weak* independent sets. Given a minimum hitting set S, its complement $V \setminus S$ forms a weak MIS in H. This complementary relationship does not hold for the *strong* MIS studied in this thesis, since it has a stricter condition which ensures that at most one vertex from each hyperedge is included in the IS. This condition does not necessarily apply to the complement of a hitting set, as it may contain multiple vertices from the same edge. Nevertheless, we show that many reduction rules for the Hitting Set problem can still be applied to the MIS problem. Figure 2.1 shows an example of a *strong* MIS, a *weak* MIS and a minimum hitting set.



(a) A strong MIS **(b)** A weak MIS **(c)** A minimum hitting set

Figure 2.1: Examples of the different problems. The named sets are highlighted in red.

CHAPTER 3

Related Work

This chapter discusses existing work related to the MIS problem in hypergraphs. Since the problem has been extensively studied for graphs, we begin by covering these results in Section 3.1. In Section 3.2, we explore research on the less studied problem of finding independent sets in hypergraphs. This primarily focuses on weak independent sets, as this variant of the problem has been studied more extensively than strong independent sets. Additionally, in Section 3.3, we examine research on the Hitting Set problem due to its close connection to the hypergraph independent set problem and the broader research on kernelization algorithms in this area.

3.1 Independent Sets in Graphs

The MIS problem on graphs is one of the best studied NP-hard problems. Given the strong relationship between graphs and hypergraphs, we examine the existing research on both exact and heuristic approaches for solving it. In particular, we focus on the most recent work that incorporates reduction strategies.

3.1.1 Exact Algorithms

Over the past decade, a high amount of research has been dedicated to developing exact algorithms for the MIS problem. Since computing an MIS is NP-hard, the best-known algorithms have worst-case exponential time complexity in terms of the number of vertices and follow the *branch-and-bound* paradigm. *Branching* is a problem-solving technique that systematically divides a problem into smaller subproblems by making a series of decisions. Each decision creates different possible solution paths, which are explored recursively until an optimal solution is found. A common branching strategy for the MIS problem is to branch on vertices with the highest degree. This splits the solution space into two cases, one that includes the vertex to the solution

and one that does not. The best solution among all branches is selected as the final result. To reduce the search space, branching is combined with *bounding*, which cuts off branches that cannot lead to a better solution than the best one found so far. A clique cover serves as a typical upper bound for the MIS problem [2].

The first non-trivial branch-and-bound schemed algorithm, designed by Tarjan and Trojanowski in 1977 [38], runs in $O(2^{\frac{n}{3}})$ time and requires polynomial space. Since then, much research has focused on reducing the base of the exponent. Fomin et al. [21] improved the time complexity to $O(1.2201^n)$ by introducing the *measure-and-conquer* approach. Bourgeois et al. [10] further refined this bound to $O(1.2114^n)$ by combining this approach with the bottom-up method. Currently, the fastest known algorithm is the $O(1.1996^n)$-time polynomial-space algorithm developed by Xiao and Nagamochi [44].

Data reduction rules proved to be a useful tool in *branch-and-reduce* methods, as they often reduce the problem size to smaller instances that can be solved exactly. The branch-and-reduce paradigm applies reduction rules to decrease the input size and uses a branching strategy when no further reduction rules are possible. Over time, various reduction rules have been developed for the MIS problem and the closely related Vertex Cover problem. Butenko et al. [11] introduced simple reductions such as *isolated vertex removal*, while Chen et al. [14] proposed *vertex folding* to deal with degree-2 vertices. More complex methods, such as the *critical independent set reduction* [12], have later proven to be effective on sparse graphs. Akiba and Iwata [2] proposed a comprehensive set of advanced reduction rules combined with sophisticated branching strategies to successfully compute an exact minimum vertex cover, and thus an exact maximum independent set.

The current state-of-the-art branch-and-reduce solver for the MIS and MWIS problems is KaMIS, developed by Lamm et al. [33]. It is designed to efficiently solve both the weighted and unweighted variants of the problem. Gellner et al. [22] extended this work by introducing the *increasing transformation* strategy. Unlike standard data reduction techniques, these transformations may temporarily increase the input size to simplify the problem, thereby enable the application of additional reduction rules.

3.1.2 Heuristic Algorithms

Heuristic algorithms aim to find near-optimal solutions very fast. A common heuristic is the *iterated local-search* technique, that iteratively improves an initial solution by making incremental changes. Andrade et al. [5] introduced the concept of (1,2)-swaps as a local-search strategy for the MIS problem. In (1,2)-swaps, a single vertex is removed from the solution and replaced by two others to improve the current solution by one. It proved to be a very successful heuristic for small to medium-size instances. Dahlum et. al. [17] combine the iterated local-search algorithm based on (1,2)-swaps with kernelization, resulting in the algorithm KerMIS. Additionally, they introduce OnlineMIS, an online approach that applies only the isolated vertex

removal reduction for vertices of degree zero, one, and two. During local search, it marks isolated vertices and their neighbors as removed.

Lamm et al. [31] integrated iterated local-search into their evolutionary algorithm, EvoMIS. An evolutionary algorithm begins with a population of individuals (in this case, independent sets) and iteratively evolves them over multiple rounds. In each round, a selection rule identifies high-quality individuals, which are then combined to generate improved offspring. EvoMIS employs a combination of graph partitioning and local search, using a combine operation based on node separators to tackle the MIS problem. Lamm et al. [32] later introduced ReduMIS, an algorithm that combines branch-and-reduce techniques with EvoMIS, utilizing both exact and inexact reductions to reduce the input size. Großmann et al. [25] developed a state-of-the-art memetic algorithm for near-optimal solutions to the MWIS problem. Their algorithm combines a genetic approach, based on recombination operations and graph partitioning techniques, with local search while incorporating advanced data reductions.

3.2 Independent Sets in Hypergraphs

Regarding independent sets in hypergraphs, a significant amount of research has focused on their theoretical approximability for specific classes of hypergraphs. The study of independent sets in hypergraphs was first introduced by Hansen and Lorea in 1976 [27]. They defined an independent set in a hypergraph as a set $S \subseteq V$ that does not contain any edge $e \in E$ with $|e| \geq 2$, which corresponds to what is known as a weak independent set. As a result, most subsequent research has adopted this definition and primarily focuses on weak independent sets.

Hofmeister and Lefmann [28] present a polynomial-time approximation algorithm for approximating the (weak) independence number in k-uniform hypergraphs, where $k \geq 2$ is fixed. Alon et al. [3] explore an application of weak independent sets by examining the relationship between the hypergraph independent set problem and the routing problem. They derive lower bounds on the performance of online algorithms for both problems. More recent research has focused on analyzing the number of weak independent sets in uniform, regular and linear hypergraphs [6, 7, 16, 36]. Here, *linear* refers to hypergraphs where the intersection of any two hyperedges contains at most one vertex.

For the strong independent set problem in hypergraphs, Halldórsson and Losievskaja [26] proposed the two greedy algorithms GreedyD and GreedyN, and established their approximation ratios. Both algorithms iteratively construct an MIS by selecting either the vertex of minimum degree (GreedyD) or the vertex with the fewest neighbors (GreedyN).

3.3 d-Hitting Sets

Since kernelization is a major preprocessing technique for FPT problems, it is widely used as a strategy to solve the d-Hitting Set problem. Many kernelization algorithms for this problem combine the two *domination rules*, first introduced by Weihe [41], with other reduction techniques. The *vertex domination rule* removes a vertex $v \in V$ if there exists another vertex $w \in V$ such that $E(v) \subset E(w)$. Similarly, the *edge domination rule* eliminates an edge $e_2 \in E$ if there exists an edge $e_1 \in E$ such that $e_1 \subset e_2$. Abu-Khzam [1] presents a kernelization algorithm for the 3-Hitting Set problem and a general kernelization for the d-Hitting Set problem. This approach combines the domination rules with *crown decomposition*, an FPT technique introduced by Fellows et. al. [15] for the Vertex Cover problem. Erdős and Rado [20] introduced in their work about the Δ-System Theorem the concept of sunflowers (see Definition 1). Since then, they have been used for polynomial-time data reduction for the d-Hitting Set problem. Van Bevern [39] proposes a linear-time kernelization algorithm that finds sunflowers to transform a d-hitting set instance into an equivalent instance with $O(k^d)$ hyperedges and vertices. To transfer the problem into the dynamic setting, Bannach et al. [8] introduce *b-flowers* as a generalization of sunflowers, since they are easier to identify.

CHAPTER 4

Algorithms for the MIS Problem on Hypergraphs

This chapter presents various algorithms designed for the Maximum Independent Set problem on hypergraphs. The main focus is on Section 4.1, which describes different reduction rules embedded in a preprocessing algorithm to obtain a kernel. To solve the MIS problem on hypergraphs, we present an exact approach using an *integer linear program* (ILP) in Section 4.2 and apply a greedy algorithm as a heuristic method, explained in Section 4.3.

4.1 Kernelization Preprocessing

The goal of kernelization preprocessing is to reduce the hypergraph to a smaller, equivalent instance, allowing following algorithms to solve the problem more efficiently. In general, there are three ways how our reduction rules can modify the hypergraph: by including a vertex, excluding a vertex, or removing a hyperedge. If a vertex v is included in the IS, all its neighbors are excluded from the solution and $N[v]$ is removed from the hypergraph. If a reduction rule excludes a vertex from the solution, it is simply removed from the graph. The removal of vertices can lead to empty hyperedges, which are removed as well.

When explaining the reductions, we provide details on how they are performed. This may include the construction of the reduced hypergraph H', the relationship between the cardinality of an MIS on the reduced hypergraph $\alpha(H')$ and the cardinality of an MIS on the original hypergraph $\alpha(H)$, or the procedure for lifting a solution $\mathcal{I}'$ from the reduced instance to a solution $\mathcal{I}$ in the original hypergraph.

We distinguish two classes of reductions: *edge reductions*, introduced in Section 4.1.1, and *vertex reductions* that are described in Section 4.1.2. In Section 4.1.3,

we provide an overview of the complete reduction framework, which combines the reductions into a single algorithm.

4.1.1 Edge Reductions

Although edge reductions do not explicitly determine whether a vertex v belongs to the MIS, they significantly enhance the effectiveness of vertex reductions. By removing edges from the hypergraph, the structural complexity is reduced, resulting in vertices with lower degrees. We start with a simple reduction considering hyperedges of size one.

Reduction 1 (Size-One Edges)**.** *If there exists an edge $e \in E$ with $|e| = 1$, remove e from H. We obtain $H' = (V, E \setminus \{e\})$, while $\alpha(H') = \alpha(H)$ and $\mathcal{I}' = \mathcal{I}$ remain the same.*

Proof. An edge of size one does not affect the adjacency of vertices. Therefore, it can be safely removed to produce an equivalent instance. □

Since this reduction is straightforward, it is not implemented explicitly but is covered by other reductions such as the following.

Reduction 2 (Edge Domination)**.** *If there exist two distinct edges $e_1, e_2 \in E$ with $e_1 \subseteq e_2$, remove e_1 from H, s.t. $H' = (V, E \setminus \{e_1\})$, $\alpha(H) = \alpha(H')$ and $\mathcal{I} = \mathcal{I}'$. We say e_2 dominates e_1 and call e_1 the dominated edge.*

Proof. Let e_1 be dominated by e_2. When e_1 is removed from H, the adjacencies of any vertex $v \in e_1$ remain unchanged, as all its neighbors $u \in e_1$ are still contained in e_2. This ensures the equivalence of the solutions in H and H' after the removal of e_1. □

Edge and vertex reductions are closely connected. When a vertex is removed by either including or excluding it from the MIS, its associated hyperedges are modified. As a result, the size of the hyperedges decreases, making it more likely that they are dominated by another edge and allowing the application of the Edge Domination. In turn, the removal of edges trough the Edge Domination reduces vertex degrees and simplifies the hypergraph's structure, enabling more vertex reductions.

4.1.2 Vertex Reductions

The vertex reductions are further classified into *low-degree reductions*, *structural reductions* and a *global reduction*. The order in which we present the reductions is motivated by their increasing complexity.

Low-Degree Reductions

The first two vertex reductions we present are well-known reduction rules for the MIS problem on graphs [2, 13, 32]. They both concern the degree of a vertex and are easily transferred to the hypergraph variant of the problem.

Reduction 3 (Degree-Zero)**.** *If there exists a vertex $v \in V$ not contained in any edge, i.e. $d(v) = 0$, we can include v in the solution. This results in $H' = H - v$ and $\mathcal{I} = \mathcal{I}' \cup \{v\}$. It holds that $\alpha(H) = \alpha(H') + 1$.*

Proof. Let $v \in V$ with $d(v) = 0$. Since v has no neighbors in V, it is part of every MIS in H. □

Reduction 4 (Degree-One)**.** *Let a vertex $v \in V$ be only contained in one edge, i.e. $d(v) = 1$. Then, there is an MIS containing v and the vertices in $N(v)$ can be removed. It holds that $\mathcal{I} = \mathcal{I}' \cup \{v\}$ with $H' = H - N[v]$ and the offset $\alpha(H) = \alpha(H') + 1$.*

Proof. Let $v \in V$ with $d(v) = 1$ and $v \in e$. Assume there is an MIS $\tilde{\mathcal{I}}$ that does not contain v. We can distinguish two cases:

1. There exists a neighbor $u \in N(v)$, such that $u \in \tilde{\mathcal{I}}$. Then, we can construct a new independent set $\mathcal{I}^* = (\tilde{\mathcal{I}} \setminus \{u\}) \cup \{v\}$ with the same cardinality by including v instead of u. Since v is a degree-one neighbor of u, replacing u with v does not affect any other vertices in the independent set.

2. No neighbor of v is included in $\tilde{\mathcal{I}}$, i.e. $N[v] \cap \tilde{\mathcal{I}} = \emptyset$. That means no vertex of e is contained in the solution. Thus, v can safely be added to $\tilde{\mathcal{I}}$, otherwise $\tilde{\mathcal{I}}$ would not be maximal. We get $H' = H - N[v]$ and $\alpha(H) = \alpha(H') + 1$.

□

Remark 1. Reduction 4 allows the degree-one vertex v to have any number of neighbors, depending on the size of the edge in which the vertex is contained. In the graph context, the degree-one reduction explicitly requires $|N(v)| = 1$. This difference makes the Degree-One for hypergraphs more powerful, as it has the potential to remove a larger number of vertices by including a degree-one vertex in the solution.

Remark 2. We briefly explain how Reduction 1 is covered by Reduction 2 and 4. Let $e_1 = \{v\}$ be an edge of size one. If v is *only* contained in e_1, meaning $d(v) = 1$, the Degree-One would include v in $\mathcal{I}$, removing the vertex along with the resulting empty edge e_1. If v has a degree $d(v) > 1$, it is contained in at least one other edge e_2 with $|e_1| \leq |e_2|$ and $e_1 \subseteq e_2$. In this case, e_1 is dominated by e_2 and would be removed by the Edge Domination.

Structural Reductions

We start the structural reductions with the twin reduction. Two vertices u and v are *twins* if they are not adjacent and have the same neighborhood. The difference between twins in a hypergraph and those in a graph is that twins in a graph always have the same degree, whereas twins in a hypergraph can have different degrees and still be considered twins. All twins form a set of vertices $T \subset V$, where either all or none of the vertices are included in an MIS. This concept is also known as a *simultaneous set*, introduced by Xiao et al. [42].

Reduction 5 (Twins)**.** *Let $T \subset V$ be a set of twins, i.e. for all $s,t \in T$ holds $N(s) = N(t)$ and s,t are not adjacent. Let $\Delta_T := min_{t \in T}\, d(t)$ be the minimum degree of all twins in T. If $|T| \geq \Delta_T$, all vertices in T are part of an MIS in H. We obtain $H' = H - N[T]$, with $\mathcal{I} = \mathcal{I}' \cup T$ and $\alpha(H) = \alpha(H') + |T|$.*

Proof. The vertices in a twin set $T \subset V$ form a *simultaneous set* [42]. That means if one vertex of T is included in an IS, all vertices of T can simultaneously be included in the independent set and increase its size by $|T|$. For the subhypergraph $H_{N(T)}$, it holds that either T or some vertices from $N(T)$ are in the MIS. To see why we can include all vertices in T if $|T| \geq \Delta_T$, we need to show that the independence number of the subhypergraph $H_{N(T)}$ depends on Δ_T. Recall that $H_{N(T)}$ contains all edges incident to the vertices in T while not containing the twins themselves.

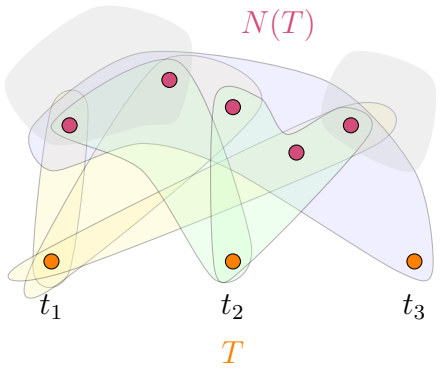
The independence number $\alpha(H_{N(T)})$ is bounded by the minimum number of edges required to cover all vertices in $N(T)$; we say a set $\mathcal{E} \subset E$ covers a set of vertices $U \subset V$, if for every $u \in U$ there exists an edge $e \in \mathcal{E}$ such that $u \in e$. Since every vertex in T is adjacent to all of $N(T)$, Δ_T represents the minimum number of edges required to cover $N(T)$ while still being incident to the vertices in T. Consequently, $\alpha(H_{N(T)}) \leq \Delta_T$ holds.

If $|T| \geq \Delta_T$, the number of twins that can be added to the IS exceeds the number of vertices in $N(T)$ that could possibly be included, as we have shown that $\alpha(H_{N(T)}) \leq \Delta_T$. Therefore, we include T in the IS and obtain $H' = H - N[T]$.

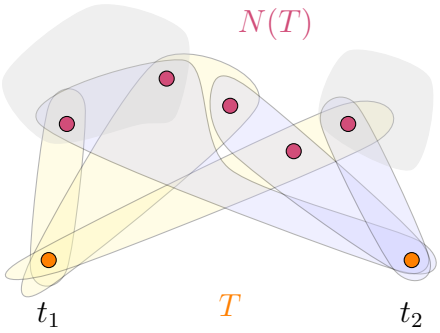
If $|T| < \Delta_T$, on the other hand, $\alpha(H_{N(T)}) \leq |T|$ is *not* assured and it might be possible that the independent set in $H_{N(T)}$ is larger than $|T|$. In this case, we do not include the vertices of T. □

We use hashing to detect vertices with identical neighborhoods, which makes the process of finding twins highly efficient. Figure 4.1 illustrates two examples of twin sets T along with their neighborhoods $N(T)$, where the gray areas represent further elements of the hypergraph. In 4.1a, the twins can be included in the solution. In contrast, 4.1b demonstrates a scenario where the twins cannot directly be added to the solution due to the constraints imposed by the reduction rule.

The next reduction focuses on detecting *sunflower* structures in the hypergraph. These structures originate from the sunflower lemma by Erdős and Rado [20], who addressed the question of when a hypergraph is sufficiently large to necessarily contain a



(a) A twin set $T = \{t_1, t_2, t_3\}$ with $\Delta_T = 1$, s.t. $|T| > \Delta_T$. Thus, all vertices in T can be added to the solution $\mathcal{I}$ and all $v \in N(T)$ can be removed.



(b) Here, the twin set $T = \{t_1, t_2\}$ is of size 2, but $\Delta_T = 3$. In this case, we do not add the vertices in T to the solution.

Figure 4.1: Two examples of twins: In (a), all twins are added to $\mathcal{I}$ according to the reduction rule, whereas in (b) they are not.

sunflower. Since its introduction, the sunflower lemma has been used in kernelization algorithms and the parameterized complexity analysis of the d-Hitting Set problem [39]. In our approach, we use the sunflower structure as a reduction rule for the MIS problem to exclude vertices from the hypergraph. We first provide a definition of a sunflower in a hypergraph.

Definition 1 (Sunflower)**.** *A k-sunflower is a collection of edges $P_1, \ldots, P_k \in E$, such that*

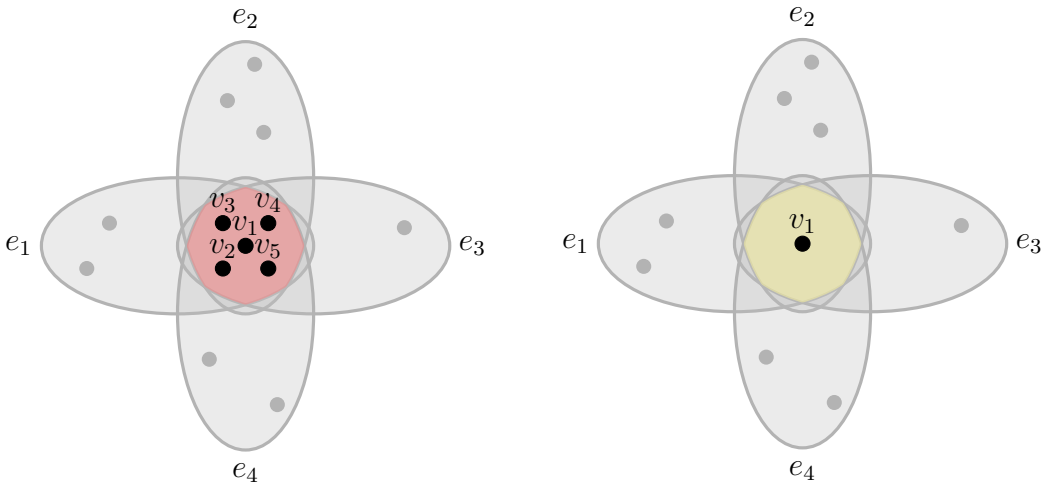
$$P_i \cap P_j = \bigcap_{t=1}^{k} P_t, \quad \forall i \neq j.$$

We call $P_1, \ldots, P_k$ the petals and $C = \bigcap_{t=1}^{k} P_t$ the core of the sunflower.

The petals $P_1, \ldots, P_k$ are distinct hyperedges that share the same intersection. The reduction using this sunflower structure looks as follows. It additionally requires that all vertices in the core of the sunflower are not incident to any other edges than the petals.

Reduction 6 (Sunflower)**.** *Let $P_1, \ldots, P_k \in E$ be a k-sunflower in H. Let C be the core of the sunflower and for all $c \in C$ holds $E(c) = \{P_1, \ldots, P_k\}$. Let $v \in C$. Then, there exists an MIS in H not containing any vertices $u \in C \setminus \{v\}$. Thus, we can remove them and obtain $H' = H - (C \setminus \{v\})$ with $\alpha(H) = \alpha(H')$ and $\mathcal{I} = \mathcal{I}'$.*

Proof. Since all vertices in the core C of a sunflower share the same set of incident edges, they are adjacent to each other. Consequently, at most one vertex $v \in C$ can be included in an IS. This vertex v can be chosen arbitrarily, as the identical incidence



(a) A sunflower with the core $C = \{v_1, \ldots, v_5\}$ marked in red and 4 petals $e_1, \ldots, e_4$. The gray parts of the petals might be further integrated into the hypergraph.

(b) The sunflower after the sunflower reduction was applied. All vertices of C except of one vertex v_1 are removed from the hypergraph.

Figure 4.2: A 4-sunflower before (a) and after (b) the sunflower reduction.

structure ensures that all vertices in C have the same neighborhood. Therefore, we exclude all vertices in the core of the sunflower except of one vertex v. □

Figure 4.2 illustrates an example of a sunflower and the reduction we apply. Similar to the Twins, we use hashing to find vertices that share the core of a sunflower. In this case, vertices are hashed based on the IDs of their incident edges, which are identical for all vertices in the core of a sunflower.

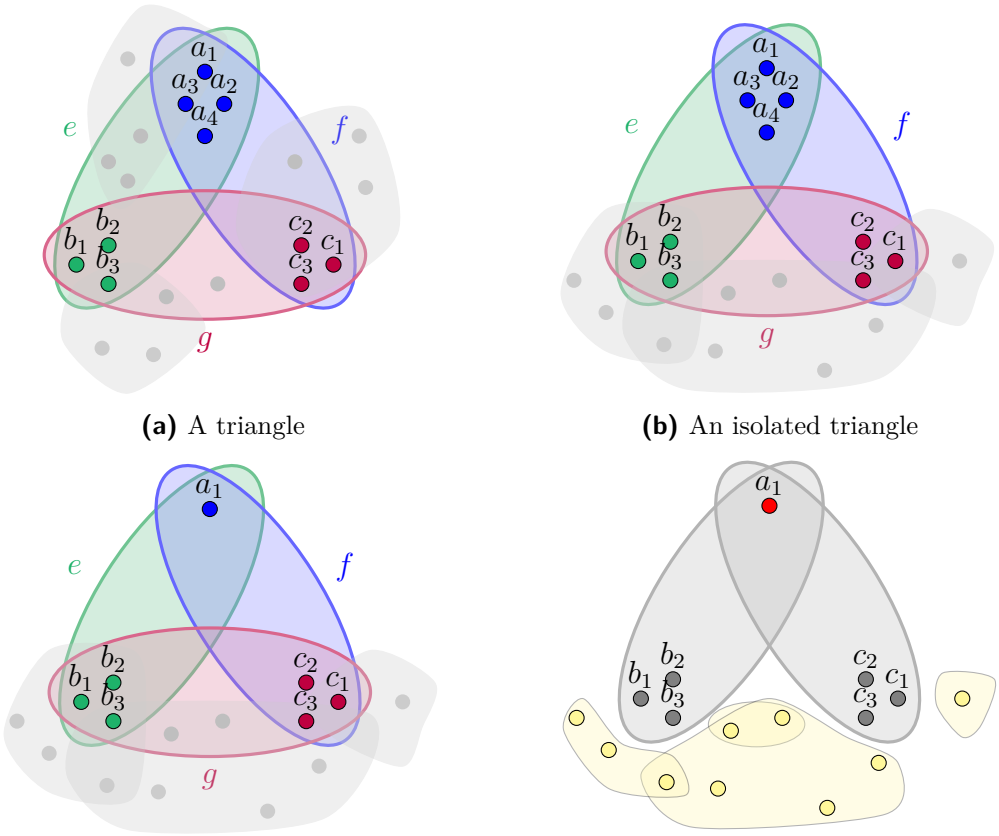
For the next reduction, we introduce structures in the hypergraph called *triangles*. An example of a triangle is shown in Figure 4.3a.

Definition 2 (Triangle)**.** *A triangle is formed by three hyperedges $e, f, g \in E$ and three subsets $A, B, C \subset V$, such that $e \cap f = A, f \cap g = B, g \cap e = C$, $e \cap f \cap g = \emptyset$ and $A, B, C \neq \emptyset$. The union $A \cup B \cup C$ are the vertices of the triangle.*

A subset A of the triangle is isolated, if (w.l.o.g) all vertices in A are only incident to the edges of the triangle and the edges e, f contain no vertices outside the triangle. More specifically, for all $a \in A$ holds $d(a) = 2$ and $A \cup B = e$, $B \cup C = f$. If a triangle contains an isolated subset, it is referred to as an isolated triangle.

Remark 3. A triangle forms a clique, as all vertices in a triangle are pair-wise adjacent. Thus, at most one vertex of a triangle can be included in an indepedent set of H.

Using the observation in Remark 3, we formulate a reduction rule that considers an isolated triangle as a specific type of triangle, ensuring that one of its vertices can always be contained in a solution.



(a) A triangle

(b) An isolated triangle

(c) The isolated triangle after the sunflower reduction

(d) The triangle after the isolated triangle reduction with a_1 being included in the IS. Elements not removed are marked in yellow.

Figure 4.3: An example of a triangle (a), an isolated triangle (b) and the Isolated Triangle reduction (c), (d). The gray elements demonstrate how the triangle could be further integrated into the hypergraph.

Reduction 7 (Isolated Triangle)**.** *Let $e, f, g \in E$ and $A, B, C \subset V$ be an isolated triangle where A is an isolated subset as defined above. There exists an MIS that contains exactly one vertex $v \in A$. Thus, we include v to the independent set and remove all vertices of the triangle. We obtain $H' = H - (A \cup B \cup C)$ with $\alpha(H) = \alpha(H') + 1$ and $\mathcal{I} = \mathcal{I}' \cup \{v\}$.*

Proof. Only one vertex of an isolated triangle can be contained in an independent set $\tilde{\mathcal{I}}$ of H, because all vertices of the triangle are adjacent to each other. Let A be the isolated subset of an isolated triangle. Assume no vertex $v \in A$ belongs to $\tilde{\mathcal{I}}$. We differ two cases:

1. A vertex u of the subsets B or C, which are not necessarily isolated, is included in the independent set. Then, we can construct a new independent set $\mathcal{I}^* = (\tilde{\mathcal{I}} \setminus \{u\}) \cup \{v\}$ that includes any $v \in A$ instead of u and has the same cardinality.

2. No vertex of the triangle belongs to $\tilde{\mathcal{I}}$. Then, a vertex $v \in A$ can safely be added to the independent set, since it has no neighbors outside of the triangle. We can choose $v \in A$ arbitrarily because of the identical incidence structure of all vertices in A. By including v in $\tilde{\mathcal{I}}$, itself and all its neighbors, i.e. the vertices of the triangle, are removed from the hypergraph and it holds $\alpha(H) = \alpha(H') + 1$.

□

Figure 4.3 shows an isolated triangle before (b) and after (d) the Isolated Triangle reduction was applied.

Remark 4. The vertices in the isolated subset A of the triangle form the core of a 2-sunflower with two petals e and f. Based on this observation, it makes sense to apply the Isolated Triangle *after* the Sunflower. This ensures that for the isolated subset A holds $|A| = 1$, making the implementation significantly more efficient, as we only need to verify that $v \in A$ is isolated and all its neighbors share at least one common edge, see Figure 4.3b and c.

The key idea behind Reduction 7 is that a triangle forms a clique (as mentioned in Remark 3), and in a clique, at most one vertex can be part of an independent set. While in the context of graphs, the size of a clique C is typically defined as the number of vertices it contains, we define the clique size $|C|$ for hypergraphs as the number of edges in the induced subhypergraph $H[C]$. Thus, a triangle is simply a clique of size three, and the Isolated Triangle can be seen as special case of a more general reduction that applies to cliques of any size.

This generalized approach in graphs is known as the *isolated clique reduction*, also referred to as *simplicial vertex removal* in the literature. It has proven to be highly effective as an MIS reduction for many real-world graph instances, as shown in [37]. We extend this reduction to the more complex setting of hypergraphs. First, we give a definition of a *simplicial vertex*.

Definition 3 (Simplicial Vertex)**.** *A simplicial vertex is a vertex v whose neighborhood forms a clique. Specifically, there exists a clique $C \subseteq V$ such that $C = N(v)$.*

Vertices with this property can be reduced according to the following reduction rule.

Reduction 8 (Isolated Clique)**.** *Let $C \subseteq V$ be a clique and let v be a simplicial vertex with $N(v) = C$. There exists an MIS that includes v, therefore we can remove C and v from H. Thus, we obtain $H' = H - (C \cup \{v\})$ and $\mathcal{I} = \mathcal{I}' \cup \{v\}$ with $\alpha(H) = \alpha(H') + 1$.*

Proof. The proof works by the same principle as for Reduction 7. At most one vertex from a clique C can be included in the MIS and there must exist a maximum independent set that includes v, as v has no neighbors outside of C. Therefore, v can be included in $\mathcal{I}$ and removed from H along with its neighbors. □

Although this reduction theoretically applies to cliques of any size, verifying whether the neighborhood of a vertex v forms a clique takes $O(|N(v)|^2 \cdot |C|)$ time, making the reduction more computationally expensive as the clique size grows and the degrees of its vertices increase. For this reason, our kernelization algorithm employs only the Isolated Triangle, which is restricted to cliques of size three and isolated vertices of degree two, as the more general reduction is not computationally efficient in the hypergraph setting.

It is worth noting that the Degree-Zero and the Degree-One are special cases of the Isolated Clique, corresponding to cliques of size zero and one, respectively.

The final structural reduction is the Vertex Domination rule, which was introduced by Weihe [41] and later used by Niedermeier and Rossmanith [35] as one of two preprocessing rules in a kernelization algorithm for the 3-Hitting Set problem. For the MIS problem, we apply a reversed version of this reduction similar to the one described by Chang et al. [13] for graphs.

Reduction 9 (Vertex Domination)**.** *If there exist two adjacent vertices $u, v \in V$ such that $N[v] \subseteq N[u]$, we say u dominates v. In this case, there exists an MIS of H that excludes u. Thus, we can remove u and obtain $H' = H - u$ with $\alpha(H) = \alpha(H')$.*

Proof. Let $u, v \in V$ with $N[v] \subseteq N[u]$ and let $\tilde{\mathcal{I}}$ be an MIS of H. If u is not in $\tilde{\mathcal{I}}$, we are done. Otherwise, suppose $u \in \tilde{\mathcal{I}}$. Then, we can construct a new MIS $\mathcal{I}^* = (\mathcal{I} \setminus \{u\}) \cup \{v\}$ with the same cardinality, since the domination property ensures that any vertex adjacent to v is also adjacent to u. Thus, $\mathcal{I}^*$ is an MIS of H excluding u. □

Global Reduction

The following *unconfined reduction* is a global reduction, as it is not restricted to a local region but can potentially extend to the entire hypergraph. This reduction was first proposed for graphs by Xiao and Nagamochi in [43]. The key idea is to remove a vertex v if the assumption, that every MIS of a graph G includes v, leads to a contradiction. This principle extends to the hypergraph context in the same way as it does in the graph context. Before we explain the reduction, we define the concept of a *child* in the hypergraph, which is an essential part of the reduction.

Definition 4 (Child)**.** *Given an independent set S of H, a vertex $u \in N(S)$ is a child of S if it has exactly one neighbor $s \in S$, i.e. $|N(u) \cap S| = 1$. The vertex s is then called the parent of u.*

Reduction 10 (Unconfined)**.** *To determine whether a vertex v is unconfined, we initialize $S = \{v\}$ and apply the following simple algorithm:*

1. *Find $u \in N(S)$ such that u is a child of S. If there exists no such vertex, v is confined.*
2. *One of the conditions holds:*
   a) *If $N(u) \setminus N[S] = \emptyset$, v is unconfined.*
   b) *If $|N(u) \setminus N[S]| > 1$, v is confined.*
   c) *If $N(u) \setminus N[S]$ is a single vertex w, add w to S and repeat the algorithm from 1.*

If the algorithm terminates at a), the assumption that every MIS in H contains v is false, and we say v is unconfined. Any unconfined vertex can be excluded and thus removed from H, as for $H' = H - v$ holds $\alpha(H) = \alpha(H')$. If the algorithm terminates at step b), the set S is said to confine the vertex v, meaning v can not be excluded. The algorithm then proceeds with a new vertex $u \in V$, initializing $S = \{u\}$.

Since this reduction is global, the size of S is only limited by n. To make the reduction less time-consuming, the size of S can be restricted to a constant p_s with $|S| \leq p_s$. If S reaches the size p_s, we classify v as confined and do not exclude it. In Section 5.3, we experiment on different values for p_s.

4.1.3 The Framework

Having introduced all reduction rules, we now describe the overall reduction framework. This includes the order in which the reductions are applied and details on how they are combined. A reduction is considered to have made progress if it decreases the number of vertices or edges in the hypergraph. The reductions follow a specific order, and whenever a reduction makes progress, the algorithm restarts from the first reduction. This iterative progress continues until no further reduction can be applied, meaning the instance becomes irreducible. To avoid exhaustively applying reductions to every vertex and edge of the hypergraph in each iteration, we introduce the concept of *vertex and edge markers*.

Excluding vertices from the solution has an impact on their neighbors that remain in the hypergraph, as their direct neighborhood changes. These affected vertices are added to the vertex marker and are considered *marked*. In all iterations of a reduction rule, except for the first, only marked vertices are examined. These are the vertices whose properties have changed, making them more likely to be eligible for a reduction. Note, that when a vertex is included, all its neighbors are excluded and subsequently removed from the instance. This affects the vertices in the second neighborhood $N^2(v)$, who are then marked and considered in the next iteration. The

Algorithm 1 Reduction Framework

Input: A hypergraph $H = (V, E)$, vertex reduction vector $\mathcal{R}$, edge reduction ε
Output: Reduced hypergraph H', independent set $\mathcal{I}$

```
Procedure ReduceHypergraph(H, R, ε):
    I ← ∅
    while progress ∧ V ≠ ∅ do
        // apply all vertex reductions first
        foreach ρ ∈ R do
            ρ.reduce(H, I)
            if progress then
                break
            end
        end
        // only if no vertex reduction is applicable, we proceed with
           edge reduction
        if no progress then
            ε.reduce(H)
        end
    end
return I
```

same principle applies for the edge marker. Whenever a vertex is removed due to excluding it from the solution, its incident edges are modified. Therefore, they are marked by the edge marker and in the following iteration, only the marked edges are considered for the edge reduction. Meanwhile, whenever an edge e is removed, the incident structure of the vertices $v \in e$ is changed, an they are in turn marked by the vertex marker. Overall, this reduces the search space of the reduction rules while increasing their effectiveness in finding reducible vertices.

Algorithm 1 presents high-level pseudocode of the preprocessing framework and Algorithm 2 provides an abstract generalization of the reduce function for vertex reduction. In this framework, vertex reductions are treated separately from the edge reduction and handled differently. The more vertices are removed through vertex reductions, the higher is the chance of finding edge dominations. Moreover, removing vertices through vertex reductions may lead to empty hyperedges. Since empty edges are removed from the hypergraph, vertex reductions can also contribute to reducing the number of edges. The Edge Domination should not remove edges that would have been removed by vertex reductions anyway, because the edge reduction is computationally more expensive. Instead, it should focus on edges, that cannot be removed by vertex reductions. To ensure this, the kernelization algorithm moves on to the edge reduction only after all vertex reductions have been exhaustively applied

and no further progress is possible. If the edge reduction successfully removes edges from the hypergraph, the algorithm restarts with vertex reductions, continuing this process iteratively. The reduce function in Algorithm 2 demonstrates how vertices are included or excluded, following a consistent pattern across all reductions. This procedure includes maintaining the vertex and edge markers. However, it does not specify the exact criteria for deciding whether a vertex is included or excluded, as these conditions vary for each reduction rule. Whenever a reduction determines that a vertex v can be included in the solution, v is added to the independent set, and all its neighbors are removed. The edges incident to these neighbors are marked by the edge marker, as their structure has changed. Similarly, all second-degree neighbors of v (i.e., the neighbors of v's direct neighbors that remain in the hypergraph) are marked by the vertex marker, since their neighborhoods are modified by the reduction. Finally, v itself is removed from the hypergraph. Whenever a reduction rule excludes a vertex u, it is processed similarly to the neighbor of an included vertex. Its incident edges are marked by the edge marker, its direct neighbors are marked by the vertex marker, and u is then removed from the hypergraph.

Reduction Ordering

The vertex reductions are applied in a predefined order. In our algorithm, we adopt the following sequence for vertex reductions, prioritizing those that are most effective in simplifying the hypergraph early in the process.

1. Degree-Zero (applied once)
2. Degree-One
3. Twins
4. Sunflower
5. Isolated Triangle
6. Vertex Domination
7. Unconfined

This ordering follows an intuitive approach by arranging the reductions based on their complexity, from the simplest to the most complex. Additionally, it takes potential dependencies between the reductions into account, such as the computational dependency between the Sunflower and Isolated Triangle, mentioned in Remark 4. We did not experiment with alternative orderings, as Großmann et al. [25] demonstrated for the Maximum Weighted Independent Set problem that their way of ordering the reduction rules provides an intuitive ordering for the closely related problem.

Remark 5. The Degree-Zero is the simplest reduction rule and is only applied once at the beginning. Initially, all degree-zero vertices are included in the independent set and subsequently removed from the hypergraph. Since no other reduction can create new degree-zero vertices, this rule does not need to be reapplied.

Parallel Computation and Restrictions

The Unconfined requires searching a large space of vertices and edges to determine whether the reduction is applicable. This makes it the most complex and least efficient of all reductions. To address this, parts of its implementation are executed in parallel. In Section 5.3.1, we explore the impact of different thread counts on performance.

A similar issue arises for reductions that explore the neighborhood of multiple vertices, particularly when these neighborhoods become significantly large. To handle this, we impose a restriction, allowing these reductions to consider only vertices whose neighborhood size remains below a predefined threshold. Similarly, the edge reduction can become computationally expensive when applied to instances with a high number of large edges. In such cases, it might be beneficial to restrict the reduction to edges of a certain size. In Section 5.3, we analyze which reductions require specific parameter restrictions and evaluate different values for those parameters through tuning experiments. It is possible, that some reductions do not need to be restricted other than by the time limit of the kernelization algorithm.

Algorithm 2 Abstract Vertex Reduction

Input: A hypergraph $H = (V, E)$
Output: Reduced hypergraph H', independent set $\mathcal{I}$

```
Function reduce(H, I):
    // in the first iteration of a reduction all vertices are marked
    if reduction has not run then
        mark all v ∈ V
    end
    foreach marked v ∈ V do
        // depending on the reduction rule, a vertex is either included
           or excluded
        if v ∈ V ∧ v can be included in I then
            I ← I ∪ {v}
            foreach u ∈ N(v) do
                foreach e ∈ E(u) do
                    mark e
                end
                V ← V \ {u}
            end
            foreach w ∈ N²(v) do
                mark w
            end
            V ← V \ {v}
        end
        else if v ∈ V ∧ v can be excluded from I then
            foreach e ∈ E(v) do
                mark e
            end
            foreach u ∈ N(v) do
                mark u
            end
            V ← V \ {v}
        end
        else
            // if the vertex was not reducible, it is not considered for
               the next iteration of this specific reduction
            unmark v
        end
    end
return I
```

4.2 ILP

A *linear program* (LP) is an optimization technique that aims to maximize or minimize an objective function subject to a system of linear inequalities, known as constraints. An *integer linear program* (ILP) is an LP with the additional constraint, that all variables take integer values. Since solving an ILP is NP-hard, specialized solvers are needed. By encoding the independence constraints and objective function as linear inequalities, state-of-the-art ILP solvers such as Gurobi or CPLEX act as black box solvers and provide the optimal solution. One can state the MIS problem on hypergraphs as an integer linear program as follows:

$$\text{maximize} \quad \sum_{v \in V} x_v \tag{4.1}$$

$$\text{subject to} \quad \sum_{i \in e} x_i \leq 1, \quad \forall e \in E, \tag{4.2}$$

$$x_v \in \{0, 1\}, \quad \forall v \in V. \tag{4.3}$$

The decision variables (4.3) indicate whether a vertex v is included in the MIS ($x_v = 1$) or excluded ($x_v = 0$). Maximizing the sum of all decision variables corresponds to finding a maximum independent set. The constraints (4.2) ensure the independence property by guaranteeing that, for every edge e in the hypergraph, at most one vertex in e can be included.

While ILPs are a widely used method for solving NP-hard problems like MIS, they are themselves NP-hard, and finding exact solutions can take exponential time in the worst case. The scalability of ILP-based methods is directly influenced by the number of vertices and hyperedges, as they increase the number of constraints. Moreover, the efficiency of solving an ILP heavily depends on the chosen solver framework. For this reason, ILPs can benefit from being applied to reduced problem instances, such as kernels, rather than serving as the sole algorithmic approach.

4.3 Heuristic Algorithm

Combining a kernelization algorithm with an ILP solver can still be computationally expensive when searching for an optimal solution. In many cases, however, an exact solution is not necessary and finding a high-quality, near-optimal solution is sufficient. Heuristic algorithms can efficiently provide such solutions. Some of them also serve as approximation algorithms, offering a theoretical guarantee on how close their solution is to the optimum, known as the *approximation ratio*.

For the Maximum Independent Set problem on hypergraphs we use the simple heuristic algorithm GreedyN. It iteratively constructs an MIS by selecting vertices with the fewest neighbors. Losievskaja showed in [26] that GreedyN has an approximation ratio of $\Delta - \frac{\Delta - 1}{r}$ in r-uniform hypergraphs. After combining the greedy

algorithm with kernelization in Section 5.4.2, we compare its performance to the optimal solution computed by the ILP in Section 5.4.3. Note, that our test instances are *not* restricted to be r-uniform.

We now briefly explain how the greedy algorithm works. It starts by iteratively choosing the vertex v with the minimum number of neighbors among all vertices in V, i.e. $v = argmin_{u \in V} |N(u)|$, and includes it in the independent set. Then, v and all its neighbors are removed from the hypergraph. The vertices in H whose neighborhood changes are updated and the algorithm repeats until no vertices are left. More detailed pseudocode for the algorithm can be found in Algorithm 3.

While the heuristic approach is computationally efficient, the quality of the solution, which is determined by the size of the found independent set, may be significantly worse compared to an exact algorithm. A common strategy to improve the solution quality is to run the heuristic multiple times with different random seeds. In GreedyN, the order in which vertices are included in the independent set is determined by the size of their neighborhood. However, when multiple vertices have the same neighborhood size, the solution can vary depending on which vertex is selected. To introduce variability and improve the overall solution quality, we apply a random perturbation factor γ, where $0.9 \leq \gamma \leq 1.1$. This factor is independently assigned to each vertex and slightly modifies the neighborhood size used for the selection. Thus, instead of selecting the next vertex v solely based on the smallest neighborhood size, we compute: $v = argmin_{u \in V} \gamma \cdot |N(u)|$. By incorporating this randomized adjustment, the heuristic explores different solution paths across multiple runs, increasing the likelihood of finding a larger independent set.

Algorithm 3 GreedyN

Input: A hypergraph $H = (V, E)$
Output: A maximal independent set $\mathcal{I}$

Procedure `GreedyN(`H`):`

```
I ← ∅
neighbors[n] ← |N(v)| for v ∈ V // we use a MinHeap
while V ≠ ∅ do
    v = argmin_{u∈V} neighbors[u]
    V ← V \ {v}
    I ← I ∪ {v}
    for u ∈ N(v) do
        V ← V \ {u}
    end
    for w ∈ N^2(v) do
        update neighbors[w]
    end
end
return I
```

CHAPTER 5

Experimental Evaluation

In this chapter, we present the experimental results of our new kernelization algorithm designed to improve the solvability of the MIS problem on hypergraphs. We begin with an overview of the methodology in Section 5.1 and a description of the set of instances used in Section 5.2. In Section 5.3, we conduct tuning experiments to determine the optimal parameter restrictions for specific reductions and the best configuration for the overall reduction framework, aiming to balance solution quality and computational efficiency. Finally, in Section 5.4, we evaluate the optimized kernelization configuration through a comparative analysis. We assess its performance combined with the exact ILP solver Gurobi against Gurobi without preprocessing. Similarly, we evaluate the solution quality of the heuristic approach GreedyN on its own and when combined with our kernelization algorithm.

5.1 Methodology

All algorithms and data structures are implemented in C++17 and compiled using g++ (gcc) 9.4 with compiler optimizations enabled (release mode, -O3 flag). The experiments are executed on a machine equipped with an AMD EPYC 9754 processor (128 cores, 1.5 GHz base, 3.1 GHz boost) with 256 MB of L3 cache and 755 GiB of main memory. Parallel computations are executed using OpenMP 4.5. For exact optimization, we use Gurobi, a state-of-the-art commercial ILP solver, under an academic license. The experiments are scheduled in parallel, utilizing up to the number of physical cores available on the machine, while restricting Gurobi to a single core for concurrent execution. The experiments to measure the efficiency of the reductions under different parameter restrictions in Section 5.3.1 are only executed once for each parameter. All subsequent experiments concerning solution quality or computation time are run 4 times per instance, taking the geometric mean as a result for the computation time and the arithmetic mean as a result for the independent set size.

As a comparison, we use performance profiles as proposed by Dolan and Moré [19]. Solution quality is evaluated based on the size of the independent set, where the largest independent set found by any method within the time constraints is considered the best solution. In the performance profiles, we plot the fraction of instances as a function of the performance ratio τ. Here, τ represents how much smaller the independent set found by a method is compared to the best-performing approach for each instance. A method that achieves a high fraction of instances for large τ values is considered to have good solution quality, as this indicates that the independent sets found are close to optimal. Similarly, we use performance profiles to compare computation times across different methods. In this context, τ is greater than or equal to 1, as it represents the time required by a method relative to the fastest approach. A value of $\tau = 2$, for example, means that the method took twice as long as the fastest approach. Since the experiments are conducted under time constraints, some methods may reach the time limit before solving an instance exactly. In cases where all compared methods reach the time limit, the best time for this specific instance is set as the time limit and thus, $\tau = 1$ for all methods. If only one method successfully solves the instance, the other method is considered incapable of solving it and we do not include the instance to its performance profile. As a result, not all methods reach a fraction of 100%.

5.2 Instances

For all experiments, we use a randomly selected subset of the M_{HG} dataset of *hypergraph instances*, provided by Gottesbüren et al. [24]. The dataset contains *general hypergraphs* and consists of 488 instances across four different use cases. These include: 18 instances for circuit design (Ispd98, [4]), 10 instances from the DAC 2012 Routability-Driven Placement Contest (Dac2012, [40]), 184 instances derived from general matrices in the SuiteSparse Matrix Collection (SPM, [18]), and 276 instances derived from the International SAT Competition 2014 (SAT14, [9]). The input files follow the *hMetis* format as described in [29]. However, when processing an instance, we ignore any vertex and edge weights, as they are not relevant to our problem. For the tuning experiments in Section 5.3, all tests are conducted on the same randomly selected subset of 13 instances, covering a wide range of different hypergraph sizes. For the full experiments in Section 5.4, we use a larger subset of 60 instances. The test instances are listed in Table A.1, with the tuning set highlighted. To guarantee comparability across different methods, we only selected test instances that are reducible by at least one reduction of the kernelization algorithm.

Table 5.1: Kernelization results for the tuning set with all selected parameter values, limiting the kernelization time by 600 seconds. The kernel size is given as the reduced number of vertices n' and edges m' in % compared to the original input size. All time values t are given in seconds.

reductions	n'	m'	t	$t_{\max}$	offset
Low-Degree					
	80.39	88.80	0.0042	5.58	4 548.1
Twins					
	79.58	88.26	0.0554	4.06	5 560.2
Sunflower					
	71.05	88.13	0.0480	2.55	5 569.4
Isolated Triangle					
	66.22	83.47	0.0156	53.15	9 163.5
Vertex Domination					
	48.51	75.28	0.0394	12.95	10 089.2
Unconfined					
	46.28	72.84	0.5073	26.26	10 140.7
Edge Domination					
	30.53	29.91	0.5823	260.75	18 388.5

5.3 Tuning Experiments

To determine the most efficient configuration of the kernelization algorithm, we conduct tuning experiments. These focus on two main aspects: First, restricting specific parameters in the reduction rules to optimize the selection of vertices and edges considered for a reduction. Second, examining different configurations of the reduction framework. The tuning experiments related to reduction parameters are presented in Section 5.3.1, where we also evaluate the individual impact of each reduction. In Section 5.3.2, we examine different kernelization time limits and reduction applications. This is done by comparing their effects on the ILP solver, considering both solution quality and computation time.

5.3.1 Reduction Parameters

Since some data reductions can be computationally expensive, we impose parameter restrictions during the search. To determine suitable parameter values, we evaluate the reductions under different settings in parameter tuning experiments. The reduc-

tions are tested in the order defined in Section 4.1.3. We start with the low-degree reductions as the initial step. In each experiment, we add one more reduction to the reduction vector, and the kernelization algorithm is tested again. Consequently, the number of applied reductions increases throughout the experimental sequence. This approach allows us to examine the impact of each reduction on the kernel and the offset. Additionally, we can determine which restriction value of a given parameter is the best choice for a specific reduction. Note that the impact of adding a reduction in our sequential experiments is not necessarily caused by that reduction alone. The introduction of a new reduction may alter the structure of the hypergraph in a way that allows previously applied reductions to become more effective. As a result, a further decrease in kernel size or an increase in the offset size may not be solely attributed to the newly added reduction, but also to the enhanced impact of those already applied. Therefore, a further kernel size reduction or increase in offset size may result not only from the new reduction rule, but also from a stronger effect of previous reductions.

For the tuning experiments, the kernelization algorithm is given a time limit of 600 seconds. The overall results of these experiments on the tuning set are presented in Table 5.1. The reported results include the kernel size as a percentage of the original input size, based on the reduced number of vertices n' and edges m'. All time measurements t are given in seconds. Both kernel size and time t are presented as the geometric mean across all instances. For some instances, the kernelization algorithm is not able to include vertices in the solution and the offset remains zero. Therefore, we determine the offset size using the arithmetic mean of all resulting offsets after applying the reductions. Additionally, we report the maximum time t_{max} required for a reduction, as this value can deviate significantly from the mean time and may provide valuable insights.

Low-Degree Reductions

In the given order, the first two reductions applied are the low-degree reductions. Since the Degree-Zero is applied only once and works similar to the Degree-One, we begin the tuning experiments with both reductions combined. The results are shown in the first row in Table 5.1. As the fastest among all reductions, these two are not restricted by any parameters. Even without restrictions, they already reduce the kernel size by nearly 20% in terms of the number of vertices and over 10% in terms of the number of edges.

Structural Reductions

The first structural reduction applied after the low-degree reductions are the Twins. For tuning, we bound the size of a vertex's neighborhood with a parameter N_{twin}. Specifically, only vertices v with $|N(v)| \leq N_{twin}$ are considered for the reduction. Table 5.2 shows that setting N_{twin} above 20 has no effect on the kernel size and

Table 5.2: Tuning experiments on the Twins using neighborhood size as a parameter. The previous low-degree reductions yielded a kernel size (n') of 80.39% and an offset of 4 548.1.

N_{twin}	n'	t	$t_{\max}$	offset
5	79.61	0.0144	2.25	5 543.0
20	79.58	0.0309	6.42	5 560.2
50	79.58	0.0554	4.06	5 560.2
100	79.58	0.1100	22.85	5 560.2
500	79.58	0.1319	36.77	5 560.2

the offset. The number of detected twins remains unchanged, while the mean and maximum computation time increase with increasing N_{twin}. Although this result might be specific to the tuning set, the likelihood of finding vertices with identical neighborhoods naturally decreases as neighborhood size increases. Consequently, we set N_{twin} to 50 in the following experiments. This allows the detection of twins with larger neighborhoods in subsequent iterations while keeping both the mean and maximum computation times low. Regarding the kernel size reduction achieved by the Twins, the effect appears to be quite small. This may be because it is a highly specific reduction and, therefore, not applicable to many hypergraph instances.

The structural reduction following the Twins is the Sunflower. Here, the results are more promising than those of its predecessor, even without the need of any parameter restrictions (see Table 5.1). The kernel size, measured by the number of vertices, is further reduced by 8%, while the runtime remains the most stable among all reductions. The slight increase in offset size indicates that the Sunflower enabled previous reductions to be applied again, as it alone does not add vertices to the solution. Both the mean and maximum times are fast, with the smallest difference between them. This is likely due to the efficient computation enabled by hashing.

The next row in Table 5.1 presents the results for the Isolated Triangle which is also not restricted. Both kernel sizes are further reduced by 5%, while the mean offset strongly increases. However, its maximum computation time exceeds 50 seconds, making it the slowest among the structural reductions. Despite this, we choose not to impose any parameter restrictions, as the high computation time affects only a single instance in the tuning set. In this case, the reduction eliminates nearly 100 000 vertices and more than triples the offset size, making the kernelization result for this specific instance highly effective.

The tuning experiments for the Vertex Domination are presented in Table 5.3. Again, we restrict the size of a vertex's neighborhood by a parameter N_{dom}. To give more insights, we also report the reduction's effect for different values of N_{dom}, mea-

Table 5.3: Tuning experiments on the Vertex Domination using neighborhood size as a parameter. The previous Isolated Triangle yielded a kernel size (n') of 66.22% and an offset of 9 163.5.

N_{dom}	n'	t	$t_{\max}$	offset	effect
5	62.46	0.0021	0.34	9 427.2	3 968.5
20	53.00	0.0098	0.33	10 060.5	13 816.4
50	50.00	0.0236	0.34	10 098.8	16 692.9
100	48.94	0.0284	0.63	10 103.8	19 580.0
200	48.51	0.0394	12.95	10 089.2	45 106.7
500	48.35	0.0458	13.08	10 162.0	47 027.9

sured as the arithmetic mean of the number of vertices removed by the Vertex Domination. For $N_{dom} \geq 20$, the reduction produces very strong results while remaining computationally efficient, further reducing the number of vertices by up to 18%. Although the kernel size improves only slightly for $N_{dom} \geq 200$, the reduction effect increases significantly when raising N_{dom} from 100 to 200. One might argue not to restrict this reduction at all, given its strong performance and fast runtime. However, this could be specific to the tuning set. Since the Vertex Domination compares neighborhoods of multiple vertices, not restricting $|N(v)|$ at all could lead to a high computation cost on large instances. To prevent this, we select $N_{dom} = 200$ for the following experiments. With this setting, the reduction contributes to a total kernel size decrease of over 50% in terms of the number of vertices when combined with all previous reductions. Moreover, the Vertex Domination increases the mean offset more than the Sunflower, even though neither reduction adds vertices directly to the solution. This shows that the large number of vertices removed by this reduction enables further applications of the other reductions that are able to include vertices in the IS.

Global Reduction

The Unconfined is the most complex reduction, which is reflected in its results. It is the reduction that requires the most restriction parameters, as presented in Table 5.4. We begin with restricting the neighborhood size by a parameter N_{unc}. It can be observed that setting N_{unc} beyond 20 barely affects the kernel size, while the mean computation time increases significantly. The offset reaches its peak for $N_{unc} = 20$, but the maximum time t_{max} consistently gets close to the time limit, regardless of N_{unc}. The mean time is also relatively high compared to other reductions.

Table 5.4: Tuning experiments on the Unconfined using multiple parameters with a time limit of 600 seconds. The previous Vertex Domination yielded a kernel size (n') of 48.51% and an offset of 10 089.2.

N_{unc}	p_t	p_s	p_{iter}	n'	t	$t_{\max}$	offset
5				47.98	0.14	326.74	10 293.8
20				44.85	3.92	528.04	10 639.7
50				44.62	13.22	529.19	10 628.2
100				44.60	17.78	596.93	10 628.4
200				44.56	20.13	596.96	10 628.4
20	**2**			44.81	2.82	532.08	10 639.7
20	**4**			44.78	2.12	532.75	10 639.7
20	**8**			44.59	1.68	534.45	10 882.2
20	**8**	**3**		44.84	0.97	397.37	10 890.0
20	**8**	**5**		44.67	1.07	495.60	10 901.1
20	**8**	**10**		44.45	1.24	522.03	10 853.9
20	**8**	**5**	**10 000**	46.88	0.40	16.20	10 124.3
20	**8**	**5**	**20 000**	46.28	0.51	26.26	10 140.7
20	**8**	**5**	**50 000**	45.94	0.62	51.15	10 152.5

As mentioned before, the Unconfined's implementation includes parts that can be executed in parallel. Therefore, we introduce the number of threads, which was previously set to one, as a second parameter p_t and test the reduction with 2, 4 and 8 threads while keeping N_{unc} fixed at 20. This slightly reduces the mean computation time, but has only a minor impact on the kernel size and the maximum computation time still almost reaches the time limit.

To address this issue, we fix $p_t = 8$ and introduce a third parameter p_s. This one limits the size of the set S, s.t. $|S| \leq p_s$, which is constructed during the reduction until a given vertex is determined to be either unconfined or confined (see Reduction 10). The results in Table 5.4 show that p_s, in contrast to the previous parameters, is able to reduce the maximum computation time while also improving the mean time. The kernel size remains at a similar quality and the offset is slightly improved for $p_s = 3$ and $p_s = 5$. Before introducing this parameter, the set S could grow arbitrarily large, potentially reaching a bottleneck at a small number of vertices.

By limiting $|S|$, we prevent the Unconfined from getting stuck at vertices where the reduction would take excessively long because S reaches a large size. Consequently, the reduction focuses on vertices that can be processed more efficiently, allowing more vertices to be handled in less time. This, in turn, can lead to a larger offset size, as more reductions contribute to the overall process. However, even with this restriction, t_{max} remains considerably high compared to other reductions and still is close to the time limit. This issue becomes particularly problematic when a stricter time limit is imposed on the kernelization algorithm, as there is a high risk that it gets stuck at the Unconfined for certain instances.

To prevent this, we fix $|S|$ at 5 and introduce a final parameter p_{iter} that limits the number of iterations for which the reduction is applied. As a result, not all currently marked vertices are considered for the reduction, but only a predefined number of them. Although this strategy drastically reduces both mean and maximum computation time, it comes at a high cost, as it compromises kernel quality and offset size. At first glance, this may not seem like the optimal choice. However, we will see that it is essential for the algorithm to quickly proceed with the Edge Domination rather than spending excessive time on the Unconfined. Since increasing p_{iter} from 20 000 to 50 000 does not yield significant improvements in kernel or offset size, we set $p_{iter} = 20\,000$ for the following experiments.

Edge Domination

The Edge Domination yields very promising results, as it immensely decreases both the number of edges and vertices in the kernel (see the last row in Table 5.1). The kernel size, in terms of the number of edges, is reduced by more than 40%, while the number of vertices decreases by 16%, despite this reduction only directly affecting edges. Furthermore, we observe the highest increase of the offset set size among all reductions. This significant improvement demonstrates that the edge removal has a strong impact on the hypergraph structure and effectively enables subsequent vertex reductions to be applied again.

In the tuning experiments, we evaluate the Edge Domination using two parameters, presented in Table 5.5. We choose the edge size as a first parameter p_e, meaning that we only consider edges e with $|e| \leq p_e$. The results indicate that both the kernel and the offset size are satisfactory for all values of p_e. However, we observe the same issue as with the Unconfined: the maximum computation time t_{max} consistently reaches the limit, regardless of p_e. This primarily affects instances in the tuning set with a very large number of hyperedges ($m \geq 1\,000\,000$). As a result, the algorithm risks spending too much time on the edge reduction, preventing it from returning to vertex reductions.

As an alternative parameter, we test stopping the Edge Domination after a predefined number p_{re} of edges have been removed. Initially, we evaluate this restriction on its own, without considering the edge size parameter p_e. A significant decrease in

Table 5.5: Tuning experiments on the Edge Domination comparing the edge size p_e and the number of removed edges p_{re} as restriction parameters with a 600 seconds time limit. The previous Unconfined yielded a kernel size (n'/m') of 46.28/72.84 and an offset of 10 140.7.

p_e	p_{re}	n'	m'	t	$t_{\max}$	offset
5		29.36	32.52	0.66	599.57	21 771.5
20		29.45	28.54	1.21	598.09	20 821.2
100		29.31	28.34	1.14	599.39	21 780.5
200		29.4	28.46	1.18	599.31	20 876.7
500		29.38	28.47	1.20	599.09	20 915.5
	10 000	40.37	50.61	0.37	155.64	12 317.2
	50 000	31.38	31.3	0.86	597.30	14 231.8
	100 000	30.63	30.07	1.00	598.07	17 775.0
200	**10 000**	40.36	50.55	0.19	47.14	12 540.1
200	**50 000**	31.14	31.04	0.47	123.41	15 518.4
200	**100 000**	30.53	29.91	0.58	201.43	18 388.5

t_{max} is only observed for $p_{re} = 10\,000$. At the same time, the kernel size is noticeably worse for this setting.

As a third variant, we fix $p_e = 200$ and combine it with p_{re}. This achieves the desired effect by ensuring a limited computation time that stays within the time limit, while both the kernel and offset size maintain close to the same quality as when using p_e alone. To optimize the overall kernel size of the algorithm, we initially choose $p_{re} = 100\,000$ for one iteration of the Edge Domination in the following experiments.

Remark 6. The tuning experiments for the Unconfined suggested, that restricting the number of iterations by p_{iter} might not lead to the optimal kernel. Therefore, we also tested the Edge Domination without setting p_{iter} for the Unconfined. Interestingly, there is no noticeable difference in the kernel and offset size when comparing the results of these two configurations. This is likely because, on the one hand, a shorter Unconfined allows for more extensive edge reduction. This, in turn, enables a greater number of subsequent vertex reductions, which are generally more efficient than the Unconfined. On the other hand, although the Unconfined can remove vertices that no other reduction can directly remove, redundant edges may block vertex reductions that would otherwise remove the same vertices. By strongly restricting the Unconfined and quickly transitioning to the Edge Domination, the hypergraph structure

Table 5.6: All evaluated parameter constraints for the reductions that led to the results in Table 5.1.

Reduction	Parameter	Bound	Value
Twins	N_{twin}	$\|N(v)\|$	50
Vertex Domination	N_{dom}	$\|N(v)\|$	200
Unconfined	N_{unc}	$\|N(v)\|$	20
	p_t	#threads	8
	p_s	$\|S\|$	5
	p_{iter}	iterations	20 000
Edge Domination	p_e	edge size	200
	p_{re}	removed edges	100 000

Table 5.7: The size of the tuning set instances before and after kernelization. The values n' and m' are reported as percentages of the original input sizes n and m. The value t denotes the time in seconds required for kernelization[1].

Instance	n	m	n'(%)	m'(%)	t
ABACUS_shell_hd.mtx.hgr	23 412	23 412	38.14	38.27	0.92
dac2012_superblue6.hgr	1 011 662	1 006 629	52.90	61.00	227.11
dictionary28.mtx.hgr	52 652	39 327	8.37	17.61	0.67
fd18.mtx.hgr	16 428	16 428	81.87	83.47	0.20
garon2.mtx.hgr	13 535	13 535	26.98	10.72	0.17
ISPD98_ibm11.hgr	70 558	81 454	66.01	64.32	5.82
lp_pds_20.mtx.hgr	108 175	33 798	61.48	52.61	1.50
nlpkkt120.mtx.hgr	3 542 400	3 542 400	90.64	97.18	103.74
NotreDame_actors.mtx.hgr	127 823	383 640	12.55	15.22	9.19
psse2.mtx.hgr	11 028	28 634	0.56	0.34	0.07
sat14_dated-10-11-u.cnf.hgr	283 720	629 461	25.27	31.17	10.90
sat14_transport-transport-city-sequential-	361 750	1 934 720	85.00	91.79	21.37
sls.mtx.hgr	62 729	1 748 122	97.04	87.38	203.22

is significantly simplified. As a result, other vertex reductions can remove vertices in a much shorter time than the Unconfined would have required for the same set of vertices. Another reason to prioritize edge reduction over the Unconfined is that removing edges during the kernelization process decreases the number of constraints for the ILP solver (see Section 4.2).

Table 5.6 provides an overview of all parameters evaluated in the tuning experiments, which led to the kernelization results in Table 5.1. Table 5.7, lists all tuning set instances, including their sizes before and after applying the kernelization algorithm.

[1]full name of instance sat14_transport-transport-city-sequential- = sat14_transport-transport-city-sequential-25nodes-1000size-3degree-100mindistance-3trucks-10packages-2008seed.030-NOTKNOWN.cnf.primal.hgr

Additionally, we give the runtime of the kernelization in seconds. Notably, none of the tuning set instances reaches the time limit of 600 seconds.

Conclusion

In conclusion, this parameter configuration yields a relatively small kernel size, with a geometric mean of around 30%, and no individual reduction exceeding the time limit. However, it might not be the optimal choice in terms of accelerating subsequent solvers. While the Edge Domination has a significant impact on the kernelization result, it also has by far the highest maximum computation time. This becomes critical when the maximum kernelization time is further reduced in the following experiments. If the algorithm spends too much time on the Edge Domination, it may not be able to proceed with vertex reductions, which are crucial for minimizing the kernel size in terms of both vertices and edges. Therefore, in the next section, we not only examine the impact of different time limits on the kernelization preprocessing but also explore alternative configurations to optimize reduction efficiency under stricter time limits.

5.3.2 Configuration Tuning

This part of the experiments aims to optimize the impact of our kernelization algorithm on solving the MIS problem with Gurobi as an open source black-box solver. Initially, we tested two methods:

(1) Gurobi: Solving the ILP directly using Gurobi.

(2) K10strong+Gur: Combining Gurobi with kernelization, where the preprocessing applies all evaluated parameters from Table 5.6 and has an initial time limit of 600 seconds.

As we later introduce methods with modified parameter restrictions, we refer to the method that applies the parameters from Table 5.6 as the *strong* variant, because it is designed to optimize the kernel size. Both methods are given a total time limit of 1800 seconds. Whenever we combine Gurobi with kernelization, the time spent on kernelization is subtracted from the total time limit, and Gurobi is executed within the remaining time. Our analysis focuses on two key aspects: whether the kernelization process is able to speedup the ILP solver, and wether it improves the solution quality compared to the solution of Gurobi without preprocessing. By solution quality, we refer to the size of the IS computed by the algorithm. If both methods terminate before reaching the time limit, the solution is an MIS and thus, identical for both solvers. However, for hard instances that remain unsolved within the given time, the size of the best IS found is reported, which may differ. The comparison is conducted on the same tuning set of 13 instances as in the previous section. Figure 5.1 presents

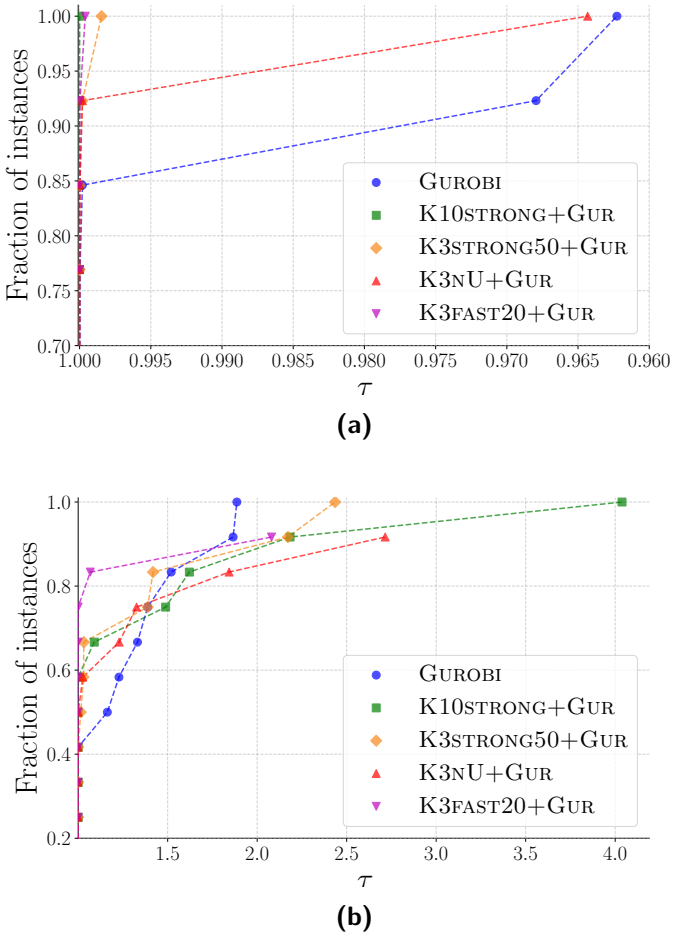


Figure 5.1: Performance profiles for solution quality (a) and computation time (b), comparing the ILP solver's performance with different kernelization configurations on the tuning set.

the performance profiles for both solution quality and runtime, including methods that will be introduced later.

The comparison of the first two methods reveals a clear trend: K10strong+Gur consistently achieves a higher solution quality. This suggests that, for instances where both methods reach the time limit, the kernelization step in K10strong+Gur effectively simplifies the problem, making it easier for Gurobi to solve. As a result, it produces better solutions within the given time constraint. However, when comparing the computation time, K10strong+Gur does not always outperform Gurobi. While K10strong+Gur is able to solve a larger fraction of instances for lower τ, it struggles to maintain this advantage as τ increases. Gurobi overtakes K10strong+Gur and reaches a 100% success rate faster. This indicates that al-

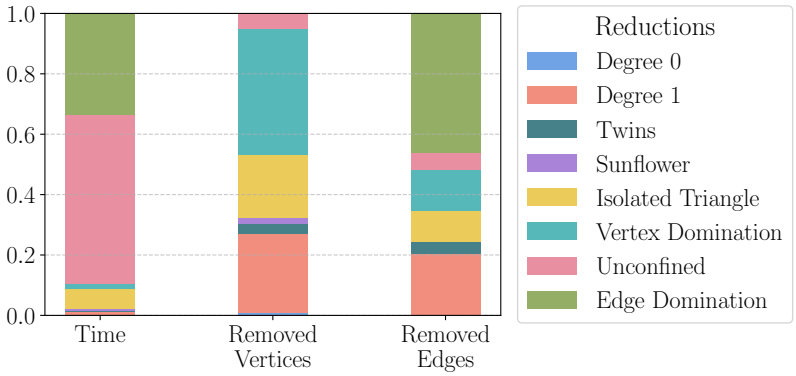


Figure 5.2: Comparing the efficiency of all reduction rules based on the number of removed vertices or edges and the required computation time.

though kernelization improves solution quality, it introduces an overhead that can negatively impact runtime performance for some instances. These might be instances where kernelization takes comparably long, while the ILP solver already solves the instance quite efficiently without preprocessing. In summary, K10Strong+Gur is preferable when solution quality is the primary objective, whereas Gurobi appears to be more efficient compared to K10Strong+Gur when the goal is to solve all instances as quickly as possible. This is why we try to find more efficient configurations that ensure a high solution quality while requiring less time.

Before exploring further methods, we first analyze the effectiveness of the reductions in K10Strong. To do so, we compare each reduction's share of the total computation time with its proportion of removed vertices and edges. Note that the Degree-Zero, by definition, does not remove any edges, while the Edge Domination cannot remove any vertices. The results for all reductions are presented in Figure 5.2. Among them, the Unconfined stands out immediately, as it consumes by far the most computation time while having only a limited effect on the number of removed vertices and edges. This inefficiency might explain why the computation time of K10Strong+Gur is suboptimal for certain instances. It raises the question of whether the kernelization algorithm might perform better without the Unconfined. In contrast, the structural reductions and the Degree-One prove to be highly efficient, especially the Vertex Domination. These reductions can eliminate a substantial number of both vertices and edges while requiring minimal computation time. As expected, the Edge Domination removes the largest proportion of edges. However, together with the Unconfined, these both reductions consume 90% of the total computation time.

To address this issue and reduce the overall computation time of the kernelization algorithm, we introduce the following three additional configurations:

(1) K3Strong50+Gur: This method enforces a maximum runtime of 3 minutes for the kernelization algorithm while using the *strong* parameter configuration,

as summarized in Table 5.6. The only modification in this variant concerns the number of edges removed per iteration of the Edge Domination, which is set to 50 000 instead of 100 000. This adjustment is motivated by the results in Table 5.5, as a single iteration of the kernelization algorithm could otherwise consume nearly the entire time budget for certain instances.

(2) K3nU+Gur: This method is identical to K3strong50+Gur except that it completely omits the Unconfined. As seen in Figure 5.2, the Unconfined is the most time-consuming reduction but has only a minor impact on the kernel size. Thus, this variant examines the algorithm's performance without it.

(3) K3fast20+Gur: This last method follows a different strategy. Instead of removing the Unconfined entirely, it applies much stricter constraints to prioritize computation speed over kernel size optimization. Specifically, for the Unconfined, $|S|$ is restricted to 3, and at most 10 000 iterations of the reduction are allowed per kernelization iteration. Additionally, the maximum kernelization runtime remains at three minutes, and the number of removed edges in the Edge Domination is further lowered to 20 000.

Since the last configuration optimizes computation time trough stricter constraints, we refer to it as the *fast* variant. The results for these three additional methods are added to the performance profiles in Figure 5.1. Regarding solution quality, all methods outperform Gurobi alone for instances that are not solved exactly within the time limit. However, there are notable differences between the methods. While the reduced computation time in K3strong50+Gur leads to only a small decrease in the size of the largest found independent set, the solution quality clearly suffers when the Unconfined is entirely removed in K3nU+Gur. Without the Unconfined, the algorithm performs only slightly better than Gurobi. Surprisingly, despite its stricter constraints and focus on speed, K3fast20+Gur delivers highly competitive results. Its solution quality almost always reaches the optimal K10strong+Gur, demonstrating the overall efficiency gains achieved by prioritizing fast reductions through more restrictive conditions for Unconfined and Edge Domination.

In terms of computation time, all methods that combine the ILP solver with kernelization preprocessing solve a higher fraction of instances for small τ compared to Gurobi alone. However, some struggle more as τ increases. Reducing the computation time to 3 minutes in K3strong50+Gur slightly improves efficiency compared to K10strong+Gur, the strong variant with a longer kernelization time. Remarkably, removing the Unconfined, the most time-consuming reduction, does not lead to better overall performance. In most instances, this variant performs similarly to or worse than K10strong+Gur. This suggests that, despite removing relatively few vertices or edges, the Unconfined eliminates vertices that no other reduction can, making it essential for achieving a high-quality kernel. Among all variants, K3fast20+Gur is the most efficient, solving nearly 80% of instances the fastest

Table 5.8: Summarizing the differences between KSTRONG and KFAST.

Reduction	Parameter	Bound	**Kstrong**	**Kfast**
Unconfined				
	p_s	$\|S\|$	**5**	**3**
	p_{iter}	iterations	**20 000**	**10 000**
Edge Domination				
	p_{re}	removed edges	**100 000**	**20 000**
Maximum kernelization time:			**10 minutes**	**3 minutes**

and outperforming all other methods. This confirms that the strict restriction of reduction parameters successfully achieves the intended effect of significantly improving runtime. Yet, there is one instance that K3FAST20+GUR and K3NU+GUR fail to solve within the time limit, whereas all other methods are able to solve it.

Furthermore, it should be noted that the performance profile for computation time also includes instances that were not solved exactly by any method within the given time. For the tuning set of 13 instances, 6 remained unsolved. These instances are particularly relevant for interpreting the performance profile in Figur 5.1a, as they are the ones where differences in solution quality may arise. Since no method was able to solve them optimally, their computation time is identical across all methods. As a result, these instances contribute equally to the fraction of instances solved best by any method, as none of the methods achieved the fastest runtime for them. To better understand the computational efficiency of the different methods, we can also compare only the 7 instances from the tuning set that were solved exactly within the given time. In this restricted comparison, neither GUROBI nor K3STRONG50+GUR was the fastest solving any method. K3NU+GUR achieved the fastest runtime for one instance, while K10STRONG+GUR was the fastest for two instances. However, K3FAST20+GUR performed best overall, solving 4 out of 7 instances faster than any other method.

Conclusion

Summarizing the results of the configuration tuning, K10STRONG+GUR is the best method in terms of solution quality, while K3FAST20+GUR achieves the highest speedup for the ILP solver. For the subsequent experiments, where we analyze the effect of kernelization preprocessing on a larger set of instances, we focus exclusively on these two methods. For simplicity, we refer to the kernelization method K10STRONG as KSTRONG and K3FAST20 as KFAST in all following experiments. Table 5.8 outlines the differences between these two methods. Any parameters not explicitly mentioned

in the table are set identically for both methods, following the restrictions evaluated in Section 5.3.1.

5.4 Experiments

In the previous section, we extracted two kernelization methods with specific parameter restrictions: KSTRONG, which prioritizes solution quality, and KFAST, which optimizes computation time. Following the approach used for configuration tuning in Section 5.3.2, we begin by comparing the performance of these two preprocessing configurations combined with Gurobi as the exact solver (KSTRONG+GUR and KFAST+GUR), against the performance of Gurobi alone (GUROBI). To ensure a broader and more representative evaluation, we randomly select a larger set of 60 instances from the M_{HG} dataset described in Section 5.2. This selection includes a wide variety of instances, ranging from very small hypergraphs to extremely large and dense ones with millions of vertices or edges, listed in Table A.1. All following experiments are conducted on the same set of instances.

After assessing the impact of KSTRONG and KFAST on the ILP solver in Section 5.4.1, we further analyze how kernelization preprocessing can enhance the performance of the GREEDYN heuristic in Section 5.4.2. For the heuristic approach, our analysis focuses exclusively on solution quality. This is because all heuristic methods are applied under a fixed total runtime of 30 minutes per instance, using the perturbation factor outlined in Section 4.3. In Section 5.4.3, we place the heuristic results into a broader context by relating them to the results of the exact approaches from the previous section. This allows us to compare the solution quality across all approaches in a unified framework.

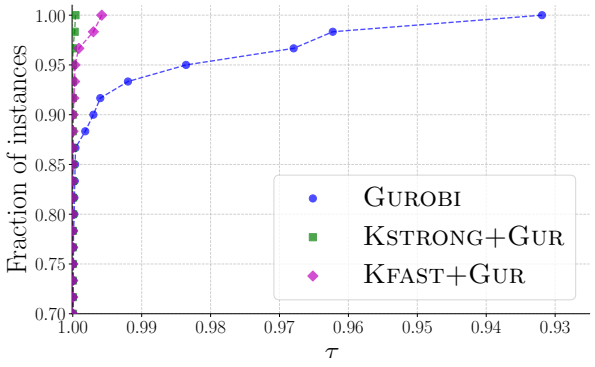
5.4.1 Impact of Kernelization on Gurobi

The experimental results are illustrated in the performance profiles in Figure 5.3. As for the configuration tuning, the total time limit was set to 1800 seconds and the time required for kernelization was subtracted from it. We begin by comparing the independent set sizes found by the three different approaches. In 80% of the instances, all methods achieve the same solution quality. However, this does not imply that these instances were solved within the time limit, but all methods managed to find an independent set of the same size within the available time. For some of those instances that were not solved exactly within the time constraint, differences in independent set sizes emerge. In these cases, the observed trend from the tuning set experiments continues: GUROBI produces the smallest independent sets, while both KSTRONG+GUR and KFAST+GUR find larger independent sets in about 20% of the instances. The improvement in independent set size achieved through kernelization reaches up to 7%. Among the tested methods, KSTRONG+GUR remains the best

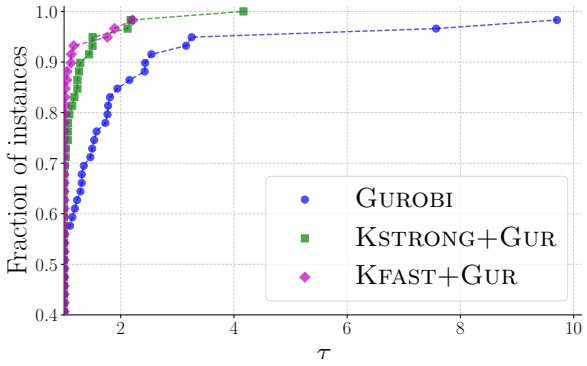
in terms of solution quality, outperforming both Gurobi and Kfast+Gur. Out of the 60 instances, 29 could not be solved exactly within the time limit by any method. Considering only these unsolved instances, Kstrong+Gur found a larger independent set than Gurobi in 75% of the cases.

While the differences between Kstrong+Gur and Gurobi are substantial, the gap between Kstrong+Gur and Kfast+Gur is notably smaller. In the tuning set experiments, Kstrong+Gur consistently achieved the optimal independent set results among all methods. For the larger set, interestingly, we observe a small number of instances where the independent set size computed by Kfast+Gur is slightly better than that of Kstrong+Gur. This suggests that, in rare cases, the stricter parameter constraints in Kfast+Gur may lead to a more efficient kernel. For these instances, a more extensive kernelization does not necessarily result in better solver performance. On the contrary, if kernelization takes a significant amount of time and reaches its time limit, the efficiency results in Figure 5.2 indicate that the algorithm may spend too much time extensively applying the Unconfined without reducing the kernel size a lot. As a consequence, a long preprocessing phase leaves less time for Gurobi to solve the instance effectively. This effect can be mitigated by imposing stricter constraints and reducing the overall kernelization time. With these limitations, the algorithm can focus on efficient vertex reductions while avoiding excessive time consumption on Unconfined and Edge Domination, thereby preserving more ILP solving time. Nevertheless, this issue arises only in a small number of instances. In most cases, Kstrong+Gur consistently achieves the best results, showing that, more often than not, extensive kernelization remains beneficial in terms of independent set size.

After evaluating solution quality, we now turn to the analysis of computation time. The performance profile in Figure 5.1b clearly highlights the positive impact of the kernelization algorithm. Unlike the results from the tuning set, this effect consistently holds for both kernelization methods, Kstrong and Kfast, both of which significantly improve the runtime of Gurobi and achieve a speedup of up to ten times. In the tuning set experiments, Gurobi appeared to surpass Kstrong+Gur in terms of runtime as τ increased. For this more representative test set, this is no longer the case. Here, even the strong kernelization approach, which does not explicitly prioritize computation time, still has a substantial impact on reducing Gurobi's runtime. This suggests that the hypergraph instances are effectively simplified by Kstrong, resulting in an overall speedup that outweighs any potential kernelization overhead. Still, Kfast+Gur remains the fastest method, solving over 80% of the instances the quickest, confirming its optimization for an efficient runtime. The results demonstrate that focusing on fast vertex reductions, while applying only a limited number of Unconfined and Edge Domination, continues to yield the intended improvement across the larger set of instances. As τ slightly increases, however, we observe an overlap between Kfast+Gur and Kstrong+Gur. This suggests that for a small subset of instances, the more extensive kernelization in Kstrong also contributes to



(a) Comparing solution quality



(b) Comparing computation time

Figure 5.3: Performance profiles for solution quality (a) and computation time (b), comparing the ILP solver's performance with different kernelization configurations on the large set of 60 instances.

a more efficient overall solving process. In these cases, the additional preprocessing effort in KSTRONG appears to simplify the instances to a point where the ILP solver ultimately benefits, allowing it to solve them even more efficiently than the same instances processed with KFAST.

As for the tuning set, the performance profile for computation time also includes instances that were not solved exactly within the given time. For the larger test set, 31 instances were solvable within the time limit. When considering only those instances, more than 30% achieve a speedup of at least factor 2 through kernelization. Comparing the three methods in the performance profiles, it is also noteworthy that KSTRONG+GUR is the only approach to reach 100%. This is because it successfully solves the highest number of instances within the time limit. This further emphasizes the impact of kernelization, not only by improving the runtime but also by making instances that GUROBI alone could not solve feasible

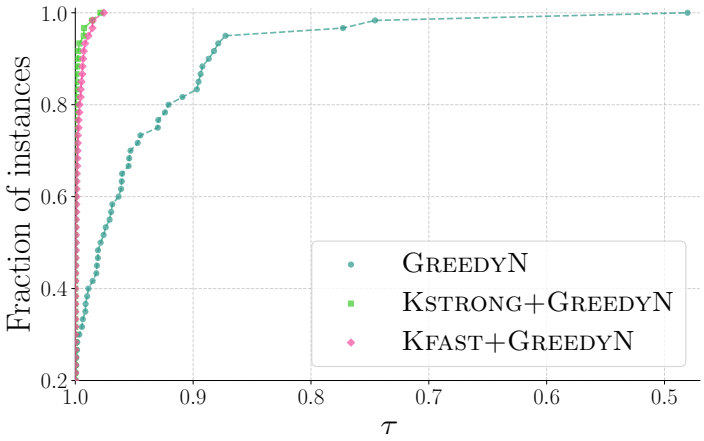


Figure 5.4: Performance profile of the heuristic approach GreedyN compared to its combination with the two kernelization methods Kstrong and Kfast.

within the given time. A complete list of solution quality and computation time for all three methods across the 60 instances is provided in the Tables A.2 and A.3, also including the results of the heuristic approaches.

5.4.2 Impact of Kernelization on GreedyN

In the following, we analyze the impact of the kernelization algorithm as a preprocessing step for the heuristic approach GreedyN to approximate the MIS problem. The experiments were conducted with a fixed total runtime of 1800 seconds. During this time, multiple iterations of GreedyN were executed with varying perturbation factors, as described in Section 4.3, to gradually improve the independent set. For the kernelization-based approaches, the time spent on kernelization is subtracted from the total 1800-seconds runtime and the heuristic GreedyN runs only within the remaining time.

The performance profile in Figur 5.4 compares the three heuristic strategies: GreedyN, Kstrong+GreedyN and Kfast+GreedyN. The impact of kernelization on improving the independent set size found by the heuristic algorithm is striking. Throughout the performance profile, both kernelization approaches, Kstrong+GreedyN and Kfast+GreedyN, consistently outperform GreedyN alone, with Kstrong+GreedyN achieving the best solution quality overall. In less than 30% of the instances, GreedyN alone achieves the same solution quality as when combined with kernelization, indicating that kernelization provides a clear advantage in most cases. For approximately 20% of the instances, kernelization leads to an improvement of more than 10% in the independent set size. Furthermore, the improvement ratio with kernelization reaches up to 50%, meaning that in at least one

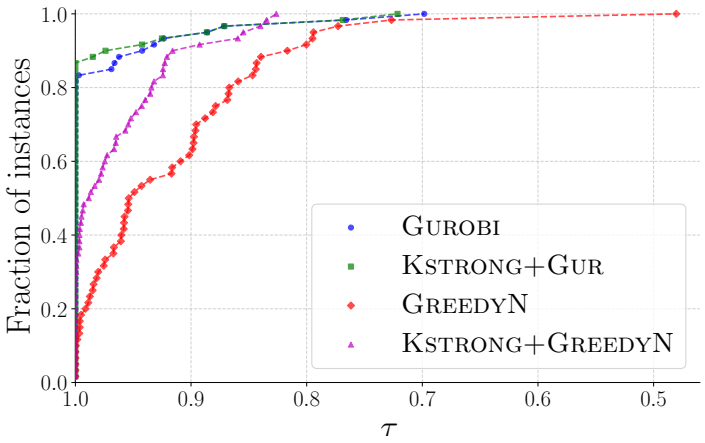


Figure 5.5: Comparing the solution quality of all methods

instance, kernelization enabled GreedyN to find an independent set twice as large as without preprocessing. Overall, this highlights the significant benefit of kernelization in enhancing the heuristic, leading to substantially better solutions within the same computation time.

Comparing the two kernelization methods, Kstrong continues to produce larger independent sets for the heuristic approach. However, the difference appears to be marginal. While Kstrong still provides an advantage over Kfast in improving the performance of GreedyN, the gap between the two is significantly smaller than when comparing either method to GreedyN alone. This suggests that although Kstrong offers a slight additional benefit, the impact of kernelization itself is the dominant factor, rather than the specific choice between Kstrong and Kfast.

5.4.3 Comparing Solution Quality of all Methods

In our final experimental evaluation, we compare the solution quality across all approaches: the heuristic method (GreedyN), the heuristic combined with kernelization (Kstrong+GreedyN), the exact solver Gurobi (Gurobi), and the kernelization algorithm combined with Gurobi (Kstrong+Gur). Since our focus is on the independent set sizes computed by each method, we restrict our analysis to results using Kstrong as the preprocessing step, as it achieves better solution quality than Kfast. In this comparison, the total runtime for the heuristic approaches GreedyN and Kstrong+GreedyN is again fixed at 1800 seconds, following the perturbation-based strategy (see Section 4.3). The exact approaches Gurobi and Kstrong+Gur also have a total time limit of 1800 seconds. However, since these methods can terminate as soon as they find an optimal solution, they may complete significantly earlier than the heuristics.

The performance profile in Figur 5.5 compares the different approaches. As expected, the heuristic method GreedyN alone performs the worst, successfully finding the best solution in only about 7% of the instances. These are primarily small, simple instances, that the exact approaches solve in less than a second. Compared to the exact methods, the limitations of GreedyN become even more apparent than in the previous analysis. For nearly 40% of the instances, it produces an independent set that is more than 10% smaller than the optimal solution, with deviations reaching up to 50%. In contrast, Kstrong+GreedyN delivers significantly better results. It finds the best independent set in over 30% of the instances, and in approximately 90% of the cases, the computed solution deviates by less than 10% from the optimal. Thus, Kstrong+GreedyN demonstrates that the kernelization makes the heuristic approach far more competitive than without preprocessing. Kstrong+Gur achieves the best overall performance among all methods, which aligns with its strong results in previous experiments. However, it now solves only around 85% of the instances optimally, which is about 10% less than in Figur 5.3a. Gurobi is still outperformed by Kstrong+Gur but the effect appears to be less. It continues to solve 80% of the instances optimally, which aligns with its previous results in Section 5.4.1. For both Kstrong+Gur and Gurobi, the remaining unsolved instances exhibit significant deviations, with solution quality dropping by up to 30% for Gurobi and slightly less for Kstrong+Gur. Meanwhile, Kstrong+GreedyN surpasses the exact approaches, reaching a fraction of 1.0 for smaller values of τ compared to Kstrong+Gur and Gurobi. This is particularly notable since Kstrong+Gur and Gurobi guarantee optimal solutions whenever they terminate before the time limit. In most cases, where they fail to solve the problem within the given time, they still produce independent sets larger than those obtained by the heuristic. However, for certain instances where the exact approaches do not succeed within the time limit, Kstrong+GreedyN finds significantly larger independent sets. The test set appears to contain particularly challenging instances where even kernelization preprocessing does not sufficiently reduce the problem complexity for Gurobi to find better solutions than the heuristic. In such cases, Kstrong+GreedyN, and to a lesser extent GreedyN, is able to approximate high-quality solutions much faster than the exact methods. Specifically, this concerns 8 instances of the test set. The detailed results can be found in Table A.2.

Overall, the experimental evaluation highlights the significant impact of kernelization as a preprocessing step for both exact and heuristic approaches to the MIS problem in hypergraphs. While Kstrong+Gur consistently achieves the best solution quality, its advantage diminishes for particularly challenging instances where the exact solver fails to reach optimality within the time limit. In such cases, the heuristic approach combined with kernelization, Kstrong+GreedyN, proves to be a strong competitor, finding independent sets that can even surpass those computed by exact methods under time constraints.

CHAPTER 6

Discussion

In this final chapter, we summarize the results of our kernelization algorithm and its impact on solving the MIS problem in hypergraphs. We reflect on the contributions of this work and highlight potential areas for further improvement in future work.

6.1 Conclusion

In this thesis, we developed and implemented a new kernelization algorithm for the Maximum Independent Set problem on hypergraphs. We adapted existing reduction techniques from the MIS problem in graphs to the hypergraph context and introduced novel reduction rules specifically designed for MIS in hypergraphs. Combined, they reduce the input instance to an average of 30% in terms of both vertices and edges. Among the vertex reduction rules, the structural reductions Vertex Domination and Isolated Clique, along with the low-degree reduction Degree-One, proved to be particularly effective. While having only a marginal impact on the algorithm's computation time, these reductions account for nearly 90% of all removed vertices. To achieve this level of vertex reduction, the Edge Domination plays a crucial role by eliminating redundant edges, thereby enabling a significant number of subsequent vertex reductions. While the Unconfined consumes a high amount of time without contributing to the kernel reduction a lot, it is able to remove vertices no other reduction can. Consequently, a kernelization algorithm including the Unconfined further enhances the performance of subsequent solvers more than kernelization without Unconfined.

The reduction rules were integrated into a reduction framework as a preprocessing step for subsequent solvers. From this, we extracted two kernelization methods: KSTRONG, which prioritizes solution quality by applying an extensive kernelization phase, and KFAST, which is optimized for speed by imposing strict parameter constraints and reducing the kernelization time limit. KFAST can accelerate Gurobi's runtime by up to a factor of 10. For 30% of the solvable instances, kernelization

achieves a speedup of at least factor 2. Additionally, with kernelization, Gurobi is able to solve one more instance optimally within the same time compared to running without preprocessing. For 75% of the instances that remain unsolved within the time limit, KSTRONG in combination with Gurobi can find up to 7% larger independent sets. This improvement of solution quality intensifies when comparing the results of the heuristic GREEDYN to those obtained by combining it with the kernelization method KSTRONG. Kernelization can double the size of the independent set found by GREEDYN, and KSTRONG+GREEDY solves 20% more instances to optimality than GREEDYN alone. Compared to the exact approach of Gurobi, KSTRONG+GREEDYN can compute up to 30% better solutions for particular hard instances where Gurobi reaches the time limit, returning results that are still far from the optimal. In these cases, KSTRONG+GREEDYN also outperforms KSTRONG+GUR, delivering high-quality approximations within significantly shorter time.

Overall, our kernelization algorithm shows significant improvements in computation time of an otherwise time-consuming exact solver like Gurobi for solving the MIS problem on hypergraphs. Additionally, it reduces the complexity of the hypergraph's structure, allowing for the computation of larger independent sets in less time. Our kernelization algorithm also enhances the heuristic approach, enabling it to produce significantly larger independent sets. This makes it a compelling alternative for hard-to-solve instances, where exact solvers like Gurobi require excessive computation time.

6.2 Future Work

While already delivering promising results, this thesis establishes a solid foundation for further improvements in solving the complex combinatorial problem of finding an MIS in a hypergraph. Regarding specific reduction rules, we currently limit the number of iterations for the Edge Domination and the Unconfined. While this improves reduction efficiency, any vertices or edges that were marked but not processed within the allocated iterations are simply ignored. Thus, there might be a better way of handling the vertex and edge marker, that maintains efficiency while ensuring that reducible vertices and edges are still considered for the reduction.

Edge reduction has proven to be a crucial step in the kernelization algorithm, as it strongly reduces the complexity of the hypergraph. Our algorithm currently incorporates only the Edge Domination as an edge reduction. While this reduction alone contributes significantly to kernel quality, it is computationally expensive due to the need to compare the incidence structures of multiple edges. Future research could focus on developing additional edge reductions that, while probably applying to fewer cases than Edge Domination, are easier to compute. This might not only further improve kernel quality, but help to enhance the efficiency of the Edge Domination as well.

The evolutionary approach REDUMIS, developed by Lamm et al. [32], demonstrated that partitioning is highly effective for solving the MIS problem in graphs.

This suggests that a similar strategy could be beneficial for kernelization in the hypergraph context, particularly for hard-to-kernelize instances where the kernelization process reaches its time limit. By breaking down the problem into smaller, more manageable subproblems, partitioning might enable a more efficient reduction process and improve overall performance. Additionally, ReduMIS integrates iterated local search to further refine solutions. In this thesis, we employed a simple greedy algorithm as a heuristic approach for the MIS problem on hypergraphs. While our focus was not on developing an optimized heuristic, but on analyzing the impact of kernelization on different solving techniques, we recognize that a local search technique could lead to better results. Unlike a simple greedy approach, which has limited potential for improving solutions, local search provides a mechanism for iterative improvements. By systematically exploring the solution space, it increases the likelihood of approximating the optimal independent set more effectively. Thus, future work could explore the combination of kernelization with advanced heuristic techniques, such as iterated local search.

APPENDIX A

Appendix

[1]full name of instance sat14_transport-transport-city-sequential- = sat14_transport-transport-city-sequential-25nodes-1000size-3degree-100mindistance-3trucks-10packages-2008seed.030-NOTKNOWN.cnf.primal.hgr

Table A.1: The dataset of 60 instances we used for the experiments[1]. The 13 instances used for the tuning experiments are highlighted.

Instance	n	m
2cubes_sphere.mtx.hgr	101 492	101 492
ABACUS_shell_hd.mtx.hgr	23 412	23 412
BenElechi1.mtx.hgr	245 874	245 874
bips07_1998.mtx.hgr	15 066	15 066
ca-CondMat.mtx.hgr	23 133	23 133
dac2012_superblue6.hgr	1 011 662	1 006 629
dictionary28.mtx.hgr	52 652	39 327
ex19.mtx.hgr	12 005	12 005
fd18.mtx.hgr	16 428	16 428
g7jac040sc.mtx.hgr	11 790	11 790
garon2.mtx.hgr	13 535	13 535
ISPD98_ibm11.hgr	70 558	81 454
ISPD98_ibm12.hgr	71 076	77 240
lhr14.mtx.hgr	14 270	14 270
lp_pds_20.mtx.hgr	108 175	33 798
m14b.mtx.hgr	214 765	214 765
msc10848.mtx.hgr	10 848	10 848
nlpkkt120.mtx.hgr	3 542 400	3 542 400
NotreDame_actors.mtx.hgr	127 823	383 640
pdb1HYS.mtx.hgr	36 417	36 417
poli3.mtx.hgr	16 955	16 955
psse2.mtx.hgr	11 028	28 634
rgg_n_2_18_s0.mtx.hgr	262 144	262 141
sat14_6s11-opt.cnf.hgr	66 552	97 312
sat14_6s12.cnf.hgr	68 066	99 580
sat14_6s130-opt.cnf.hgr	98 654	144 361
sat14_6s130-opt.cnf.primal.hgr	49 327	144 361
sat14_6s133.cnf.primal.hgr	48 215	140 968
sat14_6s16.cnf.primal.hgr	31 483	91 888
sat14_6s184.cnf.hgr	66 730	97 516
sat14_9vliw_m_9stages_iq3_C1_bug7.cnf.primal.hgr	521 147	13 378 010
sat14_ACG-20-10p1.cnf.dual.hgr	1 632 906	381 708
sat14_ACG-20-5p1.cnf.primal.hgr	331 196	1 416 850
sat14_AProVE07-27.cnf.dual.hgr	29 194	7 729
sat14_atco_enc1_opt2_05_4.cnf.dual.hgr	386 163	14 636
sat14_atco_enc1_opt2_10_12.cnf.dual.hgr	147 853	9 495
sat14_atco_enc3_opt2_05_21.cnf.dual.hgr	5 933 456	1 484 061
sat14_atco_enc3_opt2_10_14.cnf.dual.hgr	4 950 217	1 240 678
sat14_blocks-blocks-37-1.130-NOTKNOWN.cnf.primal.hgr	559 571	10 223 027
sat14_bob12m09-opt.cnf.dual.hgr	152 446	51 144
sat14_dated-10-11-u.cnf.hgr	283 720	629 461
sat14_dated-10-17-u.cnf.primal.hgr	229 544	1 070 757
sat14_E02F22.cnf.primal.hgr	13 574	1 301 188
sat14_k2fix_gr_rcs_w9.shuffled.cnf.primal.hgr	11 313	305 160
sat14_manol-pipe-c10nid_i.cnf.primal.hgr	252 516	750 877
sat14_manol-pipe-c10nidw.cnf.dual.hgr	1 291 714	433 601
sat14_MD5-30-5.cnf.dual.hgr	68 103	8 905
sat14_minandmaxor128.cnf.hgr	498 653	746 444
sat14_openstacks-p30_3.085-SAT.cnf.hgr	643 132	1 643 601
sat14_q_query_3_L100_coli.sat.cnf.primal.hgr	315 617	1 522 583
sat14_q_query_3_L150_coli.sat.cnf.hgr	973 984	2 456 708
sat14_SAT_dat.k75-24_1_rule_3.cnf.primal.hgr	1 213 331	4 754 042
sat14_SAT_dat.k85-24_1_rule_3.cnf.hgr	2 712 808	5 395 102
sat14_SAT_dat.k95-24_1_rule_3.cnf.primal.hgr	1 540 071	6 036 162
sat14_transport-transport-city-sequential-	361 750	1 934 720
sat14_UCG-15-10p1.cnf.hgr	400 006	1 019 221
sat14_UCG-15-10p1.cnf.primal.hgr	200 003	1 019 221
ship_001.mtx.hgr	34 920	34 920
sls.mtx.hgr	62 729	1 748 122
xenon2.mtx.hgr	157 464	157 464

Table A.2: Comparing the solution quality of all methods. Each experiment was run 4 times per instance, taking the rounded arithmetic mean of the independent set sizes as a result.

Instance	Gur	Kstrong+Gur	Kfast+Gur	GreedyN	Kstrong+GreedyN
2cubes_sphere.mtx.hgr	3 640	3 639	3 639	4 177	4 177
ABACUS_shell_hd.mtx.hgr	2 540	2 540	2 541	2 502	2 484
BenElechi1.mtx.hgr	4 458	4 533	4 529	4 602	4 569
bips07_1998.mtx.hgr	5 359	5 359	5 359	5 114	5 359
ca-CondMat.mtx.hgr	4 057	4 057	4 057	3 136	4 057
dac2012_superblue6.hgr	249 684	249 685	249 684	233 548	240 878
dictionary28.mtx.hgr	29 631	29 631	29 631	28 371	29 519
ex19.mtx.hgr	985	985	985	974	985
fd18.mtx.hgr	3 749	3 896	3 896	3 733	3 789
g7jac040sc.mtx.hgr	2 960	2 960	2 960	2 601	2 960
garon2.mtx.hgr	400	400	400	400	399
ISPD98_ibm11.hgr	19 660	19 660	19 660	17 725	18 717
ISPD98_ibm12.hgr	19 215	19 217	19 217	16 932	18 387
lhr14.mtx.hgr	4 314	4 314	4 314	2 073	4 314
lp_pds_20.mtx.hgr	14 743	14 743	14 743	14 382	14 665
m14b.mtx.hgr	8 710	8 712	8 712	9 744	9 830
msc10848.mtx.hgr	88	88	88	80	88
nlpkkt120.mtx.hgr	71 240	73 599	73 599	101 519	102 013
NotreDame_actors.mtx.hgr	20 684	20 685	20 685	15 032	20 159
pdb1HYS.mtx.hgr	333	334	333	266	286
poli3.mtx.hgr	3 523	3 523	3 523	3 511	3 523
psse2.mtx.hgr	3 098	3 098	3 098	3 018	3 098
rgg_n_2_18_s0.mtx.hgr	19 202	19 357	19 351	19 510	19 875
sat14_6s11-opt.cnf.hgr	30 343	30 343	30 343	26 307	30 149
sat14_6s12.cnf.hgr	31 353	31 353	31 353	27 201	30 830
sat14_6s130-opt.cnf.hgr	45 782	45 782	45 782	40 650	45 566
sat14_6s130-opt.cnf.primal.hgr	18 727	18 727	18 727	16 771	17 595
sat14_6s133.cnf.primal.hgr	18 577	18 577	18 577	16 679	17 315
sat14_6s16.cnf.primal.hgr	11 807	11 807	11 807	10 611	10 873
sat14_6s184.cnf.hgr	30 630	30 630	30 630	26 606	30 004
sat14_9vliw_m_9stages_iq3_C1_bug7.cnf.primal.hgr	29 466	31 619	31 617	29 813	29 176
sat14_ACG-20-10p1.cnf.dual.hgr	176 391	176 393	176 393	168 978	174 035
sat14_ACG-20-5p1.cnf.primal.hgr	98 451	98 451	98 451	90 212	90 991
sat14_AProVE07-27.cnf.dual.hgr	3 165	3 165	3 165	2 841	2 957
sat14_atco_enc1_opt2_05_4.cnf.dual.hgr	7 086	7 086	7 086	5 949	5 952
sat14_atco_enc1_opt2_10_12.cnf.dual.hgr	4 594	4 594	4 594	3 752	3 835
sat14_atco_enc3_opt2_05_21.cnf.dual.hgr	741 180	741 182	741 182	738 418	738 589
sat14_atco_enc3_opt2_10_14.cnf.dual.hgr	619 347	619 527	619 527	617 147	617 322
sat14_blocks-blocks-37-1.130-NOTKNOWN.cnf.primal.hgr	8 723	8 723	8 723	8 327	8 325
sat14_bob12m09-opt.cnf.dual.hgr	24 209	24 209	24 209	22 972	22 643
sat14_dated-10-11-u.cnf.hgr	90 537	90 537	90 537	86 360	90 452
sat14_dated-10-17-u.cnf.primal.hgr	61 557	61 563	61 554	60 348	61 528
sat14_E02F22.cnf.primal.hgr	594	594	594	594	594
sat14_k2fix_gr_rcs_w9.shuffled.cnf.primal.hgr	1 104	1 106	1 106	1 194	1 196
sat14_manol-pipe-c10nid_i.cnf.primal.hgr	91 911	91 911	91 911	88 294	88 836
sat14_manol-pipe-c10nidw.cnf.dual.hgr	163 439	163 439	163 439	149 852	149 709
sat14_MD5-30-5.cnf.dual.hgr	2 918	2 919	2 919	2 462	2 606
sat14_minandmaxor128.cnf.hgr	231 532	231 553	231 552	228 663	228 876
sat14_openstacks-p30_3.085-SAT.cnf.hgr	166 047	166 047	166 047	166 039	165 463
sat14_q_query_3_L100_coli.sat.cnf.primal.hgr	131 181	131 181	131 181	104 980	108 390
sat14_q_query_3_L150_coli.sat.cnf.hgr	484 565	484 565	484 565	434 238	484 190
sat14_SAT_dat.k75-24_1_rule_3.cnf.primal.hgr	474 297	474 394	474 405	401 712	449 528
sat14_SAT_dat.k85-24_1_rule_3.cnf.hgr	1 228 271	1 228 271	1 228 271	975 404	1 055 785
sat14_SAT_dat.k95-24_1_rule_3.cnf.primal.hgr	601 940	601 978	601 987	508 065	567 414
sat14_transport-transport-city-sequential-	92 880	92 880	92 880	89 788	89 639
sat14_UCG-15-10p1.cnf.hgr	132 284	132 294	132 288	138 182	140 390
sat14_UCG-15-10p1.cnf.primal.hgr	39 541	39 543	39 542	38 244	38 611
ship_001.mtx.hgr	238	238	237	205	220
sls.mtx.hgr	54 721	54 721	54 721	54 699	54 721
xenon2.mtx.hgr	4 240	4 257	4 257	5 317	5 537

Table A.3: Comparing the computation time of Gurobi and the two kernelization methods combined with Gurobi in seconds. Each experiment was run 4 times per instance with 1800 seconds as the total time limit, taking the geometric mean of the computation time as a result.

Instance	t_{Gur}	$t_{Kstrong+Gur}$	$t_{Kfast+Gur}$
2cubes_sphere.mtx.hgr	1800	1800	1800
ABACUS_shell_hd.mtx.hgr	1800	1800	1800
BenElechi1.mtx.hgr	1800	1800	1800
bips07_1998.mtx.hgr	0.1105	0.0819	0.0833
ca-CondMat.mtx.hgr	0.2291	0.1778	0.1861
dac2012_superblue6.hgr	1800	1800	1800
dictionary28.mtx.hgr	5.4362	4.146	4.226
ex19.mtx.hgr	0.1142	0.0455	0.0449
fd18.mtx.hgr	1800	1800	1800
g7jac040sc.mtx.hgr	0.6179	0.613	0.6146
garon2.mtx.hgr	1.4497	1.328	0.921
ISPD98_ibm11.hgr	1800	1800	1800
ISPD98_ibm12.hgr	1800	1800	1800
lhr14.mtx.hgr	0.2398	0.0327	0.0317
lp_pds_20.mtx.hgr	1.0329	2.2473	1.8226
m14b.mtx.hgr	1800	1800	1800
msc10848.mtx.hgr	0.6434	0.5822	0.5882
nlpkkt120.mtx.hgr	1800	1800	1800
NotreDame_actors.mtx.hgr	1800	1800	1800
pdb1HYS.mtx.hgr	1800	1800	1800
poli3.mtx.hgr	0.0406	0.0379	0.0354
psse2.mtx.hgr	0.0883	0.0746	0.0742
rgg_n_2_18_s0.mtx.hgr	1800	1800	1800
sat14_6s11-opt.cnf.hgr	15.0518	10.0112	8.4681
sat14_6s12.cnf.hgr	18.8614	15.2142	12.3111
sat14_6s130-opt.cnf.hgr	152.7722	19.4481	15.74
sat14_6s130-opt.cnf.primal.hgr	110.3004	45.5162	51.0105
sat14_6s133.cnf.primal.hgr	8.7786	13.2011	9.8898
sat14_6s16.cnf.primal.hgr	7.2663	6.052	5.5352
sat14_6s184.cnf.hgr	26.6405	15.8908	12.3629
sat14_9vliw_m_9stages_iq3_C1_bug7.cnf.primal.hgr	1800	1800	1800
sat14_ACG-20-10p1.cnf.dual.hgr	1800	1800	1800
sat14_ACG-20-5p1.cnf.primal.hgr	1800	560.1813	588.6475
sat14_AProVE07-27.cnf.dual.hgr	10.4762	8.0023	7.0021
sat14_atco_enc1_opt2_05_4.cnf.dual.hgr	34.7004	73.5901	76.7474
sat14_atco_enc1_opt2_10_12.cnf.dual.hgr	225.7786	154.1998	157.3126
sat14_atco_enc3_opt2_05_21.cnf.dual.hgr	1800	1800	1800
sat14_atco_enc3_opt2_10_14.cnf.dual.hgr	1800	1800	1800
sat14_blocks-blocks-37-1.130-NOTKNOWN.cnf.primal.hgr	1800	1800	1800
sat14_bob12m09-opt.cnf.dual.hgr	30.5125	16.9088	16.8537
sat14_dated-10-11-u.cnf.hgr	214.2061	131.8634	123.875
sat14_dated-10-17-u.cnf.primal.hgr	1800	1800	1800
sat14_E02F22.cnf.primal.hgr	332.234	145.5142	136.5837
sat14_k2fix_gr_rcs_w9.shuffled.cnf.primal.hgr	1800	1800	1800
sat14_manol-pipe-c10nid_i.cnf.primal.hgr	865.511	889.619	1012.3512
sat14_manol-pipe-c10nidw.cnf.dual.hgr	1800	1800	1800
sat14_MD5-30-5.cnf.dual.hgr	1800	1800	1800
sat14_minandmaxor128.cnf.hgr	1800	1800	1800
sat14_openstacks-p30_3.085-SAT.cnf.hgr	131.2403	52.5305	41.5966
sat14_q_query_3_L100_coli.sat.cnf.primal.hgr	710.7321	218.4437	413.0995
sat14_q_query_3_L150_coli.sat.cnf.hgr	473.9053	366.6449	244.0288
sat14_SAT_dat.k75-24_1_rule_3.cnf.primal.hgr	1800	1800	1800
sat14_SAT_dat.k85-24_1_rule_3.cnf.hgr	1800	1800	1800
sat14_SAT_dat.k95-24_1_rule_3.cnf.primal.hgr	1800	1800	1800
sat14_transport-transport-city-sequential-	1633.2056	1328.9438	1800
sat14_UCG-15-10p1.cnf.hgr	1800	1800	1800
sat14_UCG-15-10p1.cnf.primal.hgr	1800	1800	1800
ship_001.mtx.hgr	1800	1800	1800
sls.mtx.hgr	110.9259	261.7582	62.7911
xenon2.mtx.hgr	1800	1800	1800

Zusammenfassung

Diese Arbeit behandelt das bekannnte kombinatorische Problem der Maximalen Unabhängigen Menge (MIS) im Kontext von Hypergraphen. Während das MIS-Problem auf Graphen bereits umfassend untersucht wurde, konzentriert sich diese Arbeit auf die Generalisierung des Problems auf Hypergraphen, in denen Kanten beliebig viele Knoten verbinden können. Dazu stellen wir einen neuen Kernelisierungsalgorithmus vor, der exakte Reduktionsregeln anwendet, die speziell für die Hypergraph-Variante des Problems entwickelt wurden. Um Lösungsqualität und Laufzeiteffizienz auszubalancieren, untersuchen wir verschiedene Konfigurationen des Algorithmus' anhand umfangreicher Tuning-Experimente, die verschiedene Parameterrestriktionen für die Reduktionsregeln in Betracht ziehen. Der Kernelisierungsalgorithmus dient als Vorverarbeitungsschritt für nachgeschaltete Lösungsalgorithmen. Wir analysieren seinen Einfluss auf zwei unterschiedliche Lösungsansätze: den kommerziellen ILP-Löser Gurobi als exakte Methode sowie einen Greedy-Algorithmus (zu deutsch: gieriger Algorithmus) als heuristischen Ansatz. Unsere Ergebnisse zeigen deutliche Verbesserungen sowohl im Hinblick auf Laufzeit als auch die Lösungsqualität. Die Kernelisierung reduziert die Laufzeit von Gurobi bis zu einem Faktor 10. Gleichzeitig ermöglicht sie das Finden von bis zu 7% größeren unabhängigen Mengen auf Instanzen, die Gurobi innerhalb des Zeitlimits nicht optimal löst. Beim heuristischen Ansatz kann die Kernelisierung die Lösungsqualität um bis zu 50% verbessern.

Bibliography

[1] Faisal N. Abu-Khzam. A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010.

[2] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.

[3] Noga Alon, Uri Arad, and Yossi Azar. Independent sets in hypergraphs with applications to routing via fixed paths. In Dorit S. Hochbaum, Klaus Jansen, José D. P. Rolim, and Alistair Sinclair, editors, *Randomization, Approximation, and Combinatorial Algorithms and Techniques, Third International Workshop on Randomization and Approximation Techniques in Computer Science, and Second International Workshop on Approximation Algorithms for Combinatorial Optimization Problems RANDOM-APPROX'99, Berkeley, CA, USA, August 8-11, 1999, Proceedings*, volume 1671 of *Lecture Notes in Computer Science*, pages 16–27. Springer, 1999.

[4] Charles J. Alpert. The ispd98 circuit benchmark suite. In *Proceedings of the 1998 International Symposium on Physical Design*, ISPD '98, page 80–85, New York, NY, USA, 1998. Association for Computing Machinery.

[5] 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.

[6] Patrick Arras, Frederik Garbe, and Felix Joos. Asymptotically enumerating independent sets in regular k-partite k-uniform hypergraphs. *CoRR*, abs/2412.14845, 2024.

[7] József Balogh, Béla Bollobás, and Bhargav P. Narayanan. Counting independent sets in regular hypergraphs. *J. Comb. Theory A*, 180:105405, 2021.

[8] Max Bannach, Zacharias Heinrich, Rüdiger Reischuk, and Till Tantau. Dynamic kernels for hitting sets and set packing. In Petr A. Golovach and Meirav Zehavi,

editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[9] Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo. Sat 2014 competition. *SAT Competition*, 2014.

[10] Nicolas Bourgeois, Bruno Escoffier, Vangelis Paschos, and Johan Rooij. Fast algorithms for max independent set. *Algorithmica (New York)*, 62, 02 2012.

[11] Sergiy Butenko, Panos Pardalos, Ivan Sergienko, Vladimir Shylo, and Petro Stetsyuk. Finding maximum independent sets in graphs arising from coding theory. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, SAC '02, page 542–546, New York, NY, USA, 2002. Association for Computing Machinery.

[12] Sergiy Butenko and Svyatoslav Trukhanov. Using critical sets to solve the maximum independent set problem. *Oper. Res. Lett.*, 35(4):519–524, 2007.

[13] Lijun Chang, Wei Li, and Wenjie Zhang. Computing A near-maximum independent set in linear time by reducing-peeling. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1181–1196. ACM, 2017.

[14] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[15] Benny Chor, Mike Fellows, and David Juedes. Linear kernels in linear time, or how to save k colors in o(n2) steps. In Juraj Hromkovič, Manfred Nagl, and Bernhard Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science*, pages 257–269, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[16] Emma Cohen, Will Perkins, Michail Sarantis, and Prasad Tetali. On the number of independent sets in uniform, regular, linear hypergraphs. *Eur. J. Comb.*, 99:103401, 2022.

[17] Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Accelerating local search for the maximum independent set problem. In Andrew V. Goldberg and Alexander S. Kulikov, editors, *Experimental Algorithms*, pages 118–133, Cham, 2016. Springer International Publishing.

[18] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), December 2011.

[19] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *CoRR*, cs.MS/0102001, 2001.

[20] P. Erdös and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, s1-35(1):85–90, 1960.

[21] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009.

[22] Alexander Gellner, Sebastian Lamm, Christian Schulz, Darren Strash, and Bogdán Zaválnij. *Boosting Data Reduction for the Maximum Weight Independent Set Problem Using Increasing Transformations*, pages 128–142.

[23] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Evaluation of labeling strategies for rotating maps. *ACM J. Exp. Algorithmics*, 21, April 2016.

[24] Lars Gottesbüren, Tobias Heuer, Nikolai Maas, Peter Sanders, and Sebastian Schlag. Scalable high-quality hypergraph partitioning. *ACM Trans. Algorithms*, 20(1):9:1–9:54, 2024.

[25] Ernestine Großmann, Sebastian Lamm, Christian Schulz, and Darren Strash. Finding near-optimal weight independent sets at scale. *J. Graph Algorithms Appl.*, 28(1):439–473, 2024.

[26] Magnús M. Halldórsson and Elena Losievskaja. Independent sets in bounded-degree hypergraphs. *Discret. Appl. Math.*, 157(8):1773–1786, 2009.

[27] Pierre Hansen and Michel Lorea. Degrees and independent sets of hypergraphs. *Discret. Math.*, 14(4):305–309, 1976.

[28] Thomas Hofmeister and Hanno Lefmann. Approximating maximum independent sets in uniform hypergraphs. In Lubos Brim, Jozef Gruska, and Jirí Zlatuska, editors, *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, MFCS'98, Brno, Czech Republic, August 24-28, 1998, Proceedings*, volume 1450 of *Lecture Notes in Computer Science*, pages 562–570. Springer, 1998.

[29] George Karypis and Vipin Kumar. A hypergraph partitioning package. *ACM Transactions on Architecture and Code Optimization - TACO*, 01 1998.

[30] Mohammad Mehdi Daliri Khomami, Mohammad Reza Meybodi, and Alireza Rezvanian. Efficient identification of maximum independent sets in stochastic multilayer graphs with learning automata. *Results in Engineering*, 24:103224, 2024.

[31] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *CoRR*, abs/1509.00764, 2015.

[32] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *J. Heuristics*, 23(4):207–229, 2017.

[33] Sebastian Lamm, Christian Schulz, Darren Strash, Robert Williger, and Huashuo Zhang. *Exactly Solving the Maximum Weight Independent Set Problem on Large Real-World Graphs*, pages 144–158.

[34] Leran Ma, Ke Chen, and Mingfu Shao. On the maximal independent sets of k-mers with the edit distance, 2023.

[35] Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003. Combinatorial Algorithms.

[36] Guoliang Qiu and Jiaheng Wang. Inapproximability of counting independent sets in linear hypergraphs. *Inf. Process. Lett.*, 184:106448, 2024.

[37] Darren Strash. On the power of simple reductions for the maximum independent set problem. In Thang N. Dinh and My T. Thai, editors, *Computing and Combinatorics - 22nd International Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, 2016, Proceedings*, volume 9797 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2016.

[38] Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977.

[39] René van Bevern. Towards optimal and expressive kernelization for d-hitting set. *Algorithmica*, 70(1):129–147, 2014.

[40] Natarajan Viswanathan, Charles Alpert, Cliff Sze, Zhuo Li, and Yaoguang Wei. The dac 2012 routability-driven placement contest and benchmark suite. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, page 774–782, New York, NY, USA, 2012. Association for Computing Machinery.

[41] Karsten Weihe. Covering trains by stations or the power of data reduction. *Proc. ALEX'98*, 02 1998.

[42] Mingyu Xiao, Sen Huang, Yi Zhou, and Bolin Ding. Efficient reductions and a fast algorithm of maximum weighted independent set. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web*

Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021, pages 3930–3940. ACM / IW3C2, 2021.

[43] Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theor. Comput. Sci.*, 469:92–104, 2013.

[44] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017.

[45] Weixia Yu, Ju Seok Lee, Cole Johnson, Jin-Woo Kim, and Russell Deaton. Independent sets of dna oligonucleotides for nanotechnology applications. *IEEE Transactions on NanoBioscience*, 9(1):38–43, 2010.