Engineering Data Reduction and Expansion Rules for the Maximum Weight Independent Set Problem

Konrad Straube

March 27, 2024

3606810

Master Thesis

at

Algorithm Engineering Group Heidelberg
Heidelberg University

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

Co-Advisor:
Ernestine Großmann

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Dr. Christian Schulz, as well as my co-advisor, Ernestine Großmann, for their expertise, patience, guidance, and unwavering support throughout this project. I am also thankful to my friend Jonathan for his emotional support, feedback, and additional proofreading.

Abstract

In this thesis, we study a data reduction concept for the *NP*-hard Maximum Weight Independent Set problem. We examine the idea of applying not only decreasing, but also increasing transformations to a graph in order to better solve the problem. We present a number of so-called *backward reduction rules* which we analyze and then integrate into a new pre-processing algorithm. We demonstrate through the results of our experiments on a dataset containing a large number of real-life graphs, that by using this algorithm, we are able to improve the quality of solutions computed with several state-of-the-art solvers. Thereby, we show the effectiveness of the newly developed backward rules.

Contents

CHAPTER 1

Introduction

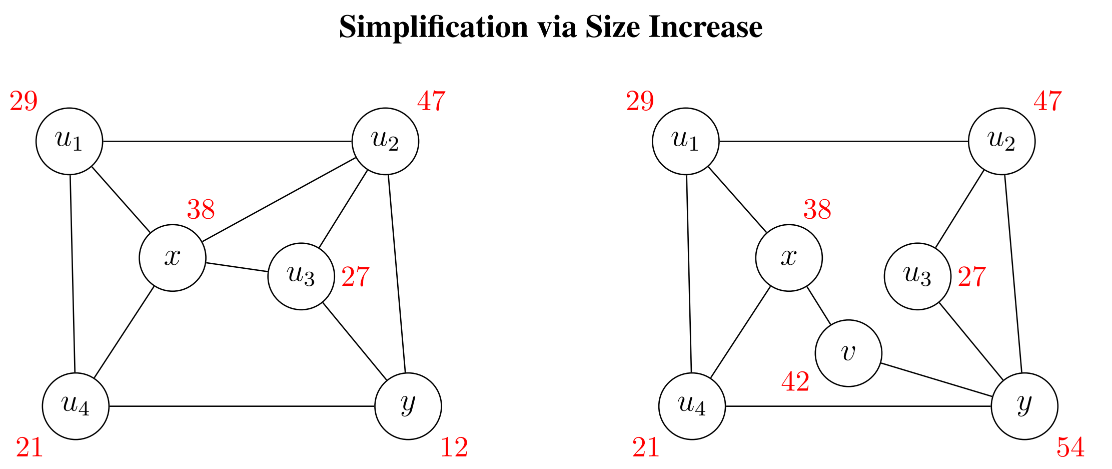
This thesis is concerned with developing a new algorithm based on data reductions for simplifying or solving the Maximum Weight Independent Set problem on large and sparse vertex-weighted graphs. The core question that we seek to answer is if we can use backward reduction rules, which are transformations *increasing* the problem size, in order to ultimately make it easier to solve the problem.

1.1 Motivation

The Role of Data Reductions. Data reductions are known as an important and frequently effective tool that enables solving the MWIS problem on many real-life graph instances. They change the problem instance in some way, typically decreasing its size, while still allowing us to find an optimal solution for problem on the original graph. Their effectiveness can vary greatly from graph to graph, with there also being many examples of complex instances where reduction rules are entirely ineffective in shrinking the graph. Assuming $P \neq NP$, we recognize that the MWIS problem, as an NP-hard problem, cannot be expected to be solvable by polynomial-time data reductions alone in the general case. Typically, data reductions play a role as a pre-processing step for some other solver, or are integrated into a sub-routine of a *branch-and-reduce* solver.

Reduction Rule Order Matters. Even in graphs where reduction rules are generally effective, the order in which they are applied can significantly affect the outcome. Most substantial is the effect on running time: In [8], the authors reported an increase in average running time of a factor of up to 4.78 when changing the order of reduction rules. Additionally, the solution quality can also be influenced significantly by the chosen order of reductions [8][6]. Whenever a reduction rule is applied to a part of a graph, it might prevent other reduction rules from being applied in that region. In many cases, a kernel which can no longer be shrunk any further by decreasing reduction rules could be smaller if reduction rules had been applied in a different order. Unfortunately, there is no known way to efficiently predict which choice of reduction rule would lead to the most favourable outcome.

The Value of Increasing the Graph Size. Since our goal is ultimately to decrease the size of the graph as much as possible, in order to allow branch-and-reduce algorithms to compute a solution more effectively, it might appear nonsensical to use increasing transformations. However, as described in the previous paragraph, a reducible graph might be reducible in multiple ways, and one way may be preferable over the other for enabling further reductions in the future. Therefore, the idea at the core of increasing transformations is to take one step back to then take two steps forward. An example of this can be seen in Figure 1.1: Through an increasing transformation, the graph G was changed to become larger, but since it also allowed us to remove some edges, the modified graph G' is now structurally simpler. Before the transformation, no decreasing transformation was applicable to G, but on the larger graph G', multiple different reductions can be utilized.



(a) Unreducible graph G with $n_G = 6$ **(b)** Reducible, inflated graph G' with $n_{G'} = 7$

Figure 1.1: An example application of a data expansion rule in the MWIS problem. The graph G in (a) is a graph which cannot be reduced further by previously known reduction rules. In (b), a data expansion rule has been applied that increased the number of vertices, but also changed the weights and neighborhoods of some vertices.

1.2 Our Contribution

Our contribution to the topic of reducing MWIS problem instances is the development and evaluation of *backward reduction rules*. Like regular reduction rules, they are modifications to a graph instance G which result in a different but equivalent instance G' of the MWIS problem. However, instead of generally reducing the size of they graph, they instead increase it. From the inflated instance G', different forward reductions might be applicable, which can then ultimately lead to an instance G'' that is even smaller than G.

The concept of increasing the size of a graph to then ultimately decrease it further, applied to the MWIS problem, has already been explored by Gellner et al. [7], who made use of new variants of WEIGHTED STRUCTION reductions, which have the capability to both decrease and increase the graph size.

In this thesis, we present a number of new graph modifications which are specifically defined to only increase the number of vertices in the graph. They are derived from previously known forward reduction rules, and essentially reverse changes that could be made to a graph by such forward rules.

Additionally, we contribute three new simple reduction rules which do not affect the number of vertices in a graph G, but instead allow us to decrease the number of edges.

1.3 Structure

The remainder of this thesis is organized as follows: In Chapter 2, we define the terms and notation used throughout the thesis, and give an exact description of the problem we are aiming to solve. In Chapter 3, we give an overview of previously developed methods for solving the MWIS problem, with a focus on work based on reduction rules and cyclic routines. Chapter 4 represents the main body of our work. First we restate the definitions of previously developed forward reduction rules in Section 4.1, as well as the special STRUCTION reductions, which can be used both for increasing and decreasing graph instances, in Section 4.2. In Section 4.3, we briefly describe reductions for deleting a number of edges from the graph before presenting the backward reductions, our main focus, in Section 4.4. In the remainder of Chapter 4, we present our cyclic reduction algorithm which makes use of all the previously mentioned reduction rules. In Chapter 5, we first improve our cyclic algorithm through parameter tuning, and then analyze the performance of our algorithm in comparison with various state-of-the-art solvers. Lastly, we reflect on our contributions in Chapter 6 and provide input for future work.

CHAPTER 2

Fundamentals

In this chapter, we provide definitions for graphs, the problem we are trying to solve, and the notation we use throughout this thesis.

2.1 General Definitions

2.1.1 Vertex-Weighted Graphs

A graph G is a data structure (V, E), consisting of the set of vertices V and the set of edges E, and conceptually represents some sort of network.

Vertices. Vertices[1] are the finite-size set of points in the graph. Each vertex is labelled with a unique nominal identifier ($V \subset \mathbb{N}_{\geq 1}$), and the number of vertices in the graph is denoted by $n = |V|$.

Edges. Edges are links between pairs of vertices, and the edge set is denoted by $E \subseteq V \times V$, with the size of the edge set denoted as $m = |E|$. In this thesis, we are considering *simple* graphs, where the two vertices u and x may not have more than one edge between them (as opposed to *multigraphs*, where there can be multiple edges with the same two endpoints). The edge between two vertices u and x is written as $e(u, x)$. Two vertices u and x are called *adjacent* iff there exists an edge with the endpoints u and x, i.e., iff $e(u, x) \in E$. In an *undirected* graph, the adjacency relationship is symmetrical, meaning $e(u, x)$ and $e(x, u)$ are considered one and the same edge. An edge from and to the same vertex x (i.e., an edge $e(x, x)$) is called a *self-loop*, however, this thesis is only concerned with graphs which do not contain any self-loops.

[1]Vertices are also known as *nodes*.

Neighborhoods. For any vertex u, the set of vertices that u is adjacent to is called the *neighborhood* of u and denoted by $N(u) = \{x \in V : e(u,x) \in E\}$. For a set of vertices $S \subseteq V$, $N(S)$ denotes the union of neighborhoods of the individual vertices in S but excluding S itself, meaning $N(S) = \cup_{x \in S} N(x) \setminus S$. Additionally, the *closed neighborhood* is denoted with square brackets and defined as $N[u] = N(u) \cup \{u\}$ and similarly, $N[S] = N(S) \cup S$. The number of neighbors of u is called the *degree* of u and denoted by $d(u) = |N(u)|$.

Paths and Cycles. Paths are sequences of vertices $(v_1, v_2, \ldots, v_k, v_{k+1}) \in V^{k+1}$ such that there exists an edge connecting each pair of successive vertices in the sequence, i.e., $\forall i \in \{1, 2, \ldots, k\}$, there is $e(v_i, v_{i+1}) \in E$. A path consisting of k edges is a k-path.

Similarly, cycles are also sequences of vertices $(v_1, v_2, \ldots, v_k) \in V^k$ where each pair of successive vertices is adjacent, but additionally, the first and last vertices v_1 and v_k are also adjacent. A cycle consisting of k vertices is a k-cycle.

Different Graphs. When discussing different graphs, say G and H, we use subscript notation to clarify which graph an element belongs to, e.g., V_G refers to the vertex set of G and $N_H(u)$ for a vertex $u \in H$ refers to $\{x \in V_H : e(u,x) \in E_H\}$.

Subgraphs. A graph H is called a *subgraph* of G iff the vertex and edge sets of H are subsets of the vertex and edge sets of G, i.e., $V_H \subseteq V_G$ and $E_H \subseteq E_G \cap (V_H \times V_H)$.

Induced Subgraphs. A subgraph of G induced by S, denoted by $G[S]$, is a special type of subgraph of G. $S \subseteq V_G$ is called the *inducing* subset, and forms the vertex set of $G[S]$. The edge set $E_{G[S]}$ is not missing any edge in E_G that is between two vertices in S (i.e., $G[S]$ contains every edge that is present in G, where S contains both endpoints: $E_{G[S]} = \{e(u,x) \in E_G : \{u,x\} \subseteq S\}$).

Independent Sets. A set of vertices $I \in V$ is called an *independent set*[2] iff no two vertices in I are adjacent to each other. For an independent set I of G, the induced subgraph $G[I]$ contains no edges.

Complement Graphs. The *complement graph* $\overline{G}$ of a graph G is the graph with the same set of vertices as G and the inverted edge set of G. Two vertices u and x are adjacent in $\overline{G}$ iff they are *not* adjacent in G.

[2]Independent sets are also known as *stable sets*.

Cliques. A set of vertices $C \in V_G$ is called a *clique* in G iff E_G contains an edge between each pair of two distinct vertices $\{u, x\} \subseteq C$. A clique containing $k = |C|$ vertices is called a k-clique. For a k-clique C, the induced subgraph $G[C]$ contains all $\frac{k(k-1)}{2}$ possible edges. A k-clique C is called *maximal* if it cannot be extended by another vertex, i.e., there is no $v \in N(C)$ such that $C \cup \{v\}$ is a $(k+1)$-clique. A graph that consists exclusively of a single clique is called a *complete graph*. A clique in G is an independent set in the complement graph $\overline{G}$.

Vertex-Weights. A *vertex-weighted* graph is a graph with an additional weight function $\omega : V \to \mathbb{N}_{\geq 1}$ (i.e., a data structure (V, E, ω)).

This means that there is a positive integer weight value mapped to every vertex. For a vertex $v \in V$, we write $\omega(v)$ to refer to this weight. For any set of vertices $U \subseteq V$, we write $\omega(U)$ to denote the sum of weights $\sum_{v \in U} \omega(v)$. If H is a subgraph of G, then $\omega_H(v) = \omega_G(v)$ for each $v \in V_H$.

Other works deal with *edge-weighted* graphs, where values are assigned to each edge. This thesis however does not, and therefore all other references to *weights* refer to vertex-weights only.

2.1.2 Problem Definition

We are now giving the definition for our problem of interest, Maximum Weight Independent Set, and its classic special case Maximum Independent Set.

Maximum Independent Set. Given an undirected, unweighted graph $G = (V, E)$, the Maximum Independent Set (MIS) problem (stated as an optimization problem) asks to find an independent set $\mathcal{I} \in V$, such that no larger independent set $S \in G$ exists with $|S| > |\mathcal{I}|$.

The MIS problem is a classic problem that has been extensively studied. It is one of the 21 original *NP*-hard problems proven by Karp [11], meaning that unless $P = NP$, one cannot expect to develop an algorithm that solves the problem efficiently in the general case.

Maximum Weight Independent Set. The Maximum Weight Independent Set (MWIS) problem is a generalization of MIS, applicable to vertex-weighted graphs $G = (V, E, \omega)$. Instead of an independent set with maximum *size*, MWIS asks to find an independent set $\mathcal{I} \subseteq V$ with maximum *weight*, such that no independent set $S \subseteq V$ exists with $\omega(S) > \omega(\mathcal{I})$, i.e. no other independent set in G has a greater sum of weights. An independent set which fulfills this condition is called a *maximum weight independent set* of G, or $MWIS(G)$ for short.

Such a solution set does not necessarily have to be unique. We denote the weight of every $MWIS(G)$ with $\alpha_\omega(G)$. An acceptable solution to the MWIS problem is *any* optimal solution set $MWIS(G)$, it is not one of our objectives to list more than one independent set with the same optimal weight $\alpha_\omega(G)$.

2.1.3 Data Reductions

Data Reduction Rules are polynomial-time algorithms which transform one problem instance into another problem instance, equivalent to the original. If a solution is found for the modified instance, it can be transformed into a solution to the original problem in polynomial time. Problem instances for MWIS are graphs $G = \{V, E, \omega\}$. Reduction rules applied to G produce a modified graph instance G', as well as instructions for translating a solution set for G' into a solution for G. Graph instances where no reduction rule can be applied to decrease n are called unreducible kernels.

CHAPTER 3

Related Work

Since MWIS is known to be *NP*-hard , making exhaustive search infeasible for large instances, a lot of collective work has gone into developing methods to systematically explore and prune the search space. Such a method is called *branch-and-bound*. Combining branch-and-bound techniques with data reduction results in a framework called *branch-and-reduce*. This framework is widely used in algorithms that aim to solve the MWIS problem. It is based on data reduction rules, which are typically used to identify certain patterns in subgraphs of the problem instance, which allows us to simplify or remove them.

Weighted Reduction Rules. In recent years, a number of reduction rules for the MWIS problem have been introduced, after the approach of data reduction has found wide use in methods for solving the Vertex Cover and MIS problems [13].

In 2007, Butenko et al. [2] proposed the CRITICAL WEIGHTED INDEPENDENT SET REDUCTION, which allows us to remove an independent set fulfilling certain conditions as well as its neighborhood from the graph. Xiao et al. [21] introduced the concept of *confinedness* for the MIS problem in 2013, and adapted it to the MWIS problem in [20]. In 2019, Lamm et al. [13] developed a number of reduction rules for the MWIS problem, including many adapted from the MIS problem, and applied them in the first branch-and-reduce algorithm for MWIS. In 2020, Zheng et al. [22] presented the TWO-VERTEX REDUCTION. Huang et al. [10] developed a number of reduction rules, many of which are based on degree-2 vertices.

Other Methods. For solving the problem inexactly, local search is widely used, including the Hybrid Iterated Local Search (HILS) algorithm by Nogueira et al. [15]. Reduction-based heuristic algorithms include ReduMIS by Lamm et al. [12], HtWIS by Gu et al. [9] and m^{2}wis by Großmann et al. [8].

3.1 Cyclic Reduction Algorithms

There are a few works which deal with methods that temporarily increase the size of the problem instance, in order to later decrease it even more, as is also the goal for our project.

Backward Reductions for Vertex Cover. The core idea of this thesis is inspired by the work of Figiel et al. [6], who developed an algorithm making use of backward reduction rules for solving the Vertex Cover problem.

Struction Reductions. Similar to our algorithm presented here, Lamm et al. [13] developed a cyclic algorithm making use of increasing and decreasing variants of so-called STRUCTION reductions to solve the MWIS problem.

CHAPTER 4

Cyclic Reduction Algorithm

This thesis describes a cyclic algorithm for transforming an instance for the MWIS problem into an equivalent, smaller one. This chapter will present the details of this cyclic reduction algorithm. The algorithm consists of two main phases, the *decreasing phase* and the *increasing phase*.

The decreasing phase mainly utilizes forward reduction rules, which are the focus of Section 4.1.

The increasing phase makes use of their inversions, the backward reduction rules, which are detailed in Section 4.4. Both phases additionally make use of STRUCTION transformations, which are explained in Section 4.2. A description of the procedures for the decreasing and the increasing phases is given in Sections 4.5 and 4.6, respectively. Finally, details of the complete cyclic algorithm are given in Section 4.7.

4.1 Weighted Forward Reduction Rules

A forward reduction rule is a transformation which shrinks the problem instance, transforming a graph G into an equivalent smaller graph G', i.e., $n_{G'} < n_G$.

A large number of weighted reduction rules for the MWIS problem have been developed by different authors. This section lists the subset of rules that is used in our algorithm, and does not attempt to be an exhaustive list of all known reduction rules.

4.1.1 Sufficient Conditions for Inclusion

Certain reductions identify simple parts of the problem instance, where for a vertex $v \in V_G$, a decision to include it in the solution set $\mathit{MWIS}(G)$ can be made directly.

Heavy Vertices. The authors of [20] define the concept of a *heavy vertex*:

Definition 1 (Heavy Vertex [10][20])
A vertex $v \in V$ is called a heavy vertex iff for any independent set I in the induced subgraph $G[N(v)]$, it holds that $\omega(v) \geq \omega(I)$.

If a vertex $v \in V_G$ is a heavy vertex, then at least one $MWIS(G)$ $\mathcal{I}$ with $v \in \mathcal{I}$ is guaranteed to exist. Therefore, we can shrink G by removing the heavy vertex v alongside its entire neighborhood.

Unfortunately, determining if a vertex v of arbitrary degree is a heavy vertex can be as complex as solving the MWIS problem itself in the general case. If the degree of v is bounded by a constant, however, it is possible to determine if it is a heavy vertex in constant time [10].

Section 4.2 will provide the definition for two STRUCTION rules, which are applied to vertices up to a fixed degree and which, as part of their routine, identify heavy vertices.

A cheaper way to determine with certainty that a vertex v is a heavy vertex is to test if an upper bound to $\alpha_\omega(G[N(v)])$ does not exceed $\omega(v)$. The most simple such upper bound is just the sum of weights $\omega(N(v))$, as is used in the NEIGHBORHOOD REMOVAL reduction[1]:

Reduction 1 (Neighborhood Removal [13][22])
For any $v \in V$, if $\omega(v) \geq \omega(N(v))$, then v is a heavy vertex and thus guaranteed to be in some $MWIS(G)$. Let $G' = G[V_G \setminus N_G[v]]$ and $MWIS(G) = MWIS(G') \cup \{v\}$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(v)$.

Extension to 2 Vertices. In [22], the authors introduce a similar reduction that can be applied specifically when NEIGHBORHOOD REMOVAL has been performed on G:

Reduction 2 (Two-Vertex [22])
Let $\{u, v\} \subseteq V$ be 2-hop-neighbors, i.e., nonadjacent vertices sharing at least one neighbor: $e(u, v) \notin E$, $|N(u) \cup N(v)| \geq 1$. If $\omega(u) < \omega(N(u))$ and $\omega(v) < \omega(N(u))$, but $\omega(u) + \omega(v) \geq \omega(N(u) \cup N(v))$, then we can obtain G' by removing $N[\{u, v\}]$. It then holds that $MWIS(G) = MWIS(G') \cup \{u, v\}$ and $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(u) + \omega_G(v)$.

Critical Weighted Independent Set. A more complex rule to identify vertices that can be directly included in the solution set was introduced by [2] in form of the CRITICAL WEIGHTED INDEPENDENT SET (CWIS) reduction:

Definition 2 (Critical Weighted Independent Set [2])
An independent set $\mathcal{I}_C \subseteq V$ is called a critical weighted independent set iff $\omega(\mathcal{I}_C) - \omega(N(\mathcal{I}_C)) = \max\{\omega(U) - \omega(N(U)) : U \subseteq V \text{is an independent set}\}$.

The CRITICAL WEIGHTED INDEPENDENT SET reduction is the only *global* reduction used in our algorithm, as the critical weighted independent set $\mathcal{I}_C$ is computed for the entire graph. This set $\mathcal{I}_C$ can be found in polynomial time via various methods [2]. In our

[1]Called *Single-Vertex Reduction* in [22]

implementation, we find them by constructing a bipartite flow graph from G and solving the *Max-Flow-Min-Cut* problem on this flow graph.

Reduction 3 (Critical Weighted Independent Set [2])
Let $\mathcal{I}_C \subseteq V$ be a critical weighted independent set. Then there exists at least one $MWIS(G)$ $\mathcal{I}^*$ such that $\mathcal{I}_C \subseteq \mathcal{I}^*$. Thus, G' is obtained by removing $N[\mathcal{I}_C]$ from G. It holds that $MWIS(G) = MWIS(G') \cup \mathcal{I}_C$ and $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(\mathcal{I}_C)$.

4.1.2 Sufficient Conditions for Removal

While reductions like NEIGHBORHOOD REMOVAL can shrink the graph when a vertex can be shown to be included in at least one solution set $MWIS(G)$, another way to directly remove parts of the graph is to identify a vertex which is *absent* in at least one solution set $MWIS(G)$.

Definition 3 (Removable Vertices [10])
A vertex $v \in V_G$ is called removable *iff there is at least one $MWIS(G)$ that does not include v. For a removable vertex v in G, it holds that $\alpha_\omega(G[V \setminus \{v\}]) = \alpha_\omega(G)$.*

Again, no efficient algorithm exists to decide whether any given vertex is removable in the general case. Otherwise, a solution for the MWIS problem as a whole could be found efficiently by process of elimination. There are, however, several useful reductions which check if a sufficient condition for the removal of a vertex is met.

Basic Single-Edge. The following reduction, BASIC SINGLE-EDGE, was introduced in [9] and directly compares two adjacent vertices v and x, testing if v is always at least as good of a choice for inclusion in the solution as x is.

In the context of this reduction, the term $\omega(N(v) \setminus N[x]) + \omega(x)$ serves as an upper bound for the weight of an independent subset $I_x \subseteq N(v)$ which includes $x \in I_x$.

Intuitively, if this upper bound does not exceed $\omega(v)$, then v can be selected instead of such a set I_x. Therefore, at least one optimal solution does not include x, and G can be simplified by removing x.

Reduction 4 (Basic Single-Edge [9])
Given an edge $e(v, x) \in E$, if $\omega(v) \geq \omega(N(v) \setminus N[x]) + \omega(x)$, then $G' = G[V \setminus \{x\}]$ is obtained by deleting the removable vertex x, and $\alpha_\omega(G) = \alpha_\omega(G')$.

Extended Single-Edge. Another reduction which can potentially identify multiple removable vertices at once is the following:

Reduction 5 (Extended Single-Edge [9])
Let $\{u, v\} \subseteq V$ be two adjacent vertices, and let $C = N(u) \cap N(v)$. If $\omega(u) + \omega(v) \geq \min\{\omega(N(u)), \omega(N(v))\}$, then the common neighbors C are removable and $\alpha_\omega(G) = \alpha_\omega(G[V_G \setminus C])$.

Unconfined Vertices. Another way to identify removable vertices is through the notion of *unconfined* vertices, defined for weighted graphs in [10] and [20]. According to [20], "the idea of UNCONFINED is that we first assume a vertex v is in all $MWIS(G)$, and then try to find some contradictions. If there are some contradictions found, then the vertex v is not in some $MWIS(G)$ and we can delete it from the graph."

Definition 4 (Child, Extending Child, Satellite)
Let $S \subseteq V$ be an independent set. A vertex $u \in N(S)$ is called a child of S iff $\omega(u) \geq \omega(S \cap N(u))$. A child u is called an extending child of S if $|N(u) \setminus N[S]| = 1$, and the vertex $v \in N(u) \setminus N[S]$, which is adjacent to u, is called a satellite of S.

Definition 5 (Confinedness [10])
Let $v \in V$ be some vertex. The following procedure determines if v is confined or unconfined:

(1) Let $S \leftarrow \{v\}$.
(2) If S has at least one extending child in $N(S)$, then let S' be the set of satellites corresponding to the extending children of S. Update $S \leftarrow S \cup S'$.
(3) If S is no longer independent or if there is a child u of S such that $N(u) \setminus N[S] = \{\}$, then we halt and conclude that v is unconfined.
(4) If S has no extending child (i.e., $|N(u) \setminus N[S]| \geq 2$ for all children $u \in N(S)$), then we halt and conclude that v is confined by S.

Repeat (2) until (3) or (4) holds.

This check for confinedness can be performed in polynomial time [10]. Showing that a vertex v is unconfined means that v is absent in at least one $MWIS(G)$ and is hence removable:

Reduction 6 (Unconfined [10] [20])
For any vertex $v \in V$, if v is determined to be unconfined via the above procedure, then we obtain $G' = G[V_G \setminus \{v\}]$ by removing v from G. It holds that $MWIS(G) = MWIS(G')$ and $\alpha_\omega(G) = \alpha_\omega(G')$.

4.1.3 Simultaneous Set Merging

In some cases, it can be determined for a set of vertices S that all of S simultaneously is either included or excluded in some $MWIS(G)$. The following reduction covers one such case:

Reduction 7 (Twin [13][20])
If two nonadjacent vertices $\{x, y\} \subseteq V$ have the same neighbor set (i.e., $N(x) = N(y)$), then they can be merged. We obtain G' by removing x and v and adding a new vertex v^* with $\omega_{G'}(v^*) \leftarrow \omega_G(x) + \omega_G(y)$ and $N_{G'}(v^*) \leftarrow N_G(x)$. It holds that $\alpha_\omega(G') = \alpha_\omega(G)$ and $\{x, y\} \in MWIS(G)$ iff $v^* \in MWIS(G')$.

4.1.4 Alternative Sets

The concept of *alternative sets* for weighted graphs was defined in [20]:

Definition 6 (Alternative Set [20])
Let $\mathcal{I}_A \subseteq V$ be an independent set in G. If there is at least one $\mathit{MWIS}(G)$ which contains either all of $\mathcal{I}_A$ or all of $N(\mathcal{I}_A)$, then $\mathcal{I}_A$ is called an alternative set.

Let the graph G contain an alternative set $\mathcal{I}_A$. Since either $\mathcal{I}_A$ or $N(\mathcal{I}_A)$ must be contained in some $\mathit{MWIS}(G)$, there is a binary decision to be made. Therefore $N[\mathcal{I}_A] = \mathcal{I}_A \cup N(\mathcal{I}_A)$ can be replaced by a single vertex v^* (and can be said to be *folded into* v^*), representing exactly this binary decision.

In the following, the reductions DEGREE-1 FOLD and V-SHAPE MAX (Reductions 8 and 9) are presented, which are reductions based on alternative sets and vertex folding.

Low-Degree Reductions. Vertices with low degree frequently allow for reductions to be applied. A degree-1 vertex $v \in V$ with $N(v) = \{u\}$ can be removed via the NEIGHBORHOOD REMOVAL rule (Reduction 1) if $\omega(v) \geq \omega(u)$.

Otherwise (i.e., if $\omega(v) < \omega(u)$), the DEGREE-1 FOLD reduction defined below applies. In such a case, the vertex v is an example of an alternative set: Either $\{v\}$ or $N(\{v\}) = \{u\}$ must be a subset of every $\mathit{MWIS}(G)$.

Reduction 8 (Degree-1 Fold [9])
Let $v \in V$ be a degree-1 vertex with $N(v) = \{u\}$ and $\omega(v) < \omega(u)$. Update $G' = G[V_G \setminus \{v\}]$ and set $\omega_{G'}(u) \leftarrow \omega_G(u) - \omega_G(v)$. From an $\mathit{MWIS}(G')$ $\mathcal{I}'$, we get an $\mathit{MWIS}(G)$ $\mathcal{I}$ as follows: If $u \in \mathcal{I}$, then $\mathcal{I} = \mathcal{I}'$, otherwise $\mathcal{I} = \mathcal{I}' \cup \{v\}$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(v)$.

V-Shape. A *v-shape* is a 2-path $(x, v, y) \in V^3$ where $d(v) = 2$ with $N(v) = \{x, y\}$. x and y are not adjacent. In other words, a v-shape is present when there is a degree-2 vertex v, and $N[v]$ does not form a 3-clique, also known as a triangle. Our algorithm makes use of several reductions involving v-shape patterns, including V-SHAPE MAX[2]:

Reduction 9 (V-Shape Max [9][13])
Let $v \in V$ be a degree-2 vertex with $N(v) = \{x, y\}$, such that $e(x, y) \notin E$ and $\max(\omega(x), \omega(y)) \leq \omega(v) < \omega(x) + \omega(y)$. Then G' is obtained by removing the vertices v, x and y and replacing them with a new vertex v', with $\omega_{G'}(v') \leftarrow \omega_G(x) + \omega_G(y) - \omega_G(v)$ and $N_{G'}(v') \leftarrow N_G(x) \cup N_G(y)$. From an $\mathit{MWIS}(G')$ $\mathcal{I}'$, we obtain the $\mathit{MWIS}(G)$ $\mathcal{I}$ as follows: If $v' \in \mathcal{I}'$, then $\{x, y\} \subseteq \mathcal{I}$, otherwise $v \in \mathcal{I}$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(v)$.

With $\max(\omega(x), \omega(y)) \leq \omega(v) < \omega(x) + \omega(y)$, there must be at least one optimal solution $\mathit{MWIS}(G)$ $\mathcal{I}$ where either $\{x, y\} \subseteq \mathcal{I}$ or $v \in \mathcal{I}$. This binary decision can be

[2]Also called VERTEX FOLDING in [13].

represented by just one vertex, the folded vertex v'. In this way, $\{v\}$ is another example of an alternative set.

4.1.5 Other Reductions Based on Degree-2 Vertices

Let there be a v-shape structure with degree-2 vertex v and its nonadjacent neighbors x and y such that $\omega(x) \leq \omega(v) < \omega(y)$.

For the subset of these three vertices that is included in an optimal solution, $S = \{v, x, y\} \cap \mathit{MWIS}(G)$, we only need to consider the three options $S = \{v\}$, $S = \{x, y\}$ and $S = \{y\}$. $S = \{\}$ is not an option, because there is no reason not to include at least v, since it is not connected to any other vertices. The option $S = \{x\}$ can be disregarded because $\{v\}$ is at least as good. All other combinations are not independent sets.

The V-SHAPE MID reduction exploits the fact that only these three options are possible, as this information can be compressed into just 2 vertices.

Reduction 10 (V-Shape Mid [9])
Let $v \in V$ be a degree-2 vertex with $N(v) = \{x, y\}$, such that $e(x, y) \notin E$ and $\omega(x) \leq \omega(v) < \omega(y)$. Then G' is obtained by removing the vertex v and updating $\omega_{G'}(y) \leftarrow \omega_G(y) - \omega_G(v)$ and $N_{G'}(x) \leftarrow N_G(x) \cup N_G(y)$.

Let $\mathcal{I}'$ be a solution $\mathit{MWIS}(G')$. $\mathcal{I}' \cap \{x, y\}$ can be one of $\{x, y\}$, $\{y\}$ or $\{\}$, but not $\{x\}$, because $N_{G'}(x) \supseteq N_{G'}(y)$. We obtain $\mathit{MWIS}(G)$ $\mathcal{I}$ as follows: If neither $\{x, y\} \subseteq \mathcal{I}'$ nor $\{y\} \subseteq \mathcal{I}'$, then $\mathcal{I} = \mathcal{I}' \cup \{v\}$, otherwise $\mathcal{I} = \mathcal{I}'$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(v)$.

In [10], Huang et al. present several other reduction rules involving multiple degree-2 vertices:

Reduction 11 (3-Path [10])
Let there be a 3-path (v_1, v_2, v_3, v_4) such that $d(v_2) = d(v_3) = 2$ and $\omega(v_4) \leq \omega(v_3) \leq \omega(v_2) \leq \omega(v_1)$. Then G' is obtained by removing v_2 and v_3, adding $e(v_1, v_4)$ if it did not already exist, and updating $\omega_{G'}(v_1) \leftarrow \omega_G(v_1) + \omega_G(v_3) - \omega_G(v_2)$. If $v_1 \in \mathit{MWIS}(G')$ then $\mathit{MWIS}(G) = \mathit{MWIS}(G') \cup \{v_3\}$, otherwise $\mathit{MWIS}(G) = \mathit{MWIS}(G') \cup \{v_2\}$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(v_2)$.

Reduction 12 (4-Cycle [10])
Let there be a 4-cycle (v_1, v_2, v_3, v_4) such that $d(v_2) = d(v_3) = 2$ and $\omega(v_3) \leq \omega(v_2) \leq \omega(v_1)$. Then G' is obtained by removing v_2 and v_3, and updating $\omega_{G'}(v_1) \leftarrow \omega_G(v_1) + \omega_G(v_3) - \omega_G(v_2)$. If $v_1 \in \mathit{MWIS}(G')$ then $\mathit{MWIS}(G) = \mathit{MWIS}(G') \cup \{v_3\}$, otherwise $\mathit{MWIS}(G) = \mathit{MWIS}(G') \cup \{v_2\}$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(v_2)$.

Reduction 13 (4-Path [10])
Let there be a 4-path $(v_1, v_2, v_3, v_4, v_5)$ such that $d(v_2) = d(v_3) = d(v_4) = 2$, $\omega(v_3) \leq \omega(v_2) \leq \omega(v_1)$ and $\omega(v_3) \leq \omega(v_4) \leq \omega(v_5)$. Then G' is obtained by the following: Remove v_2 and v_4, add edges $e(v_1, v_3)$ and $e(v_3, v_5)$, and update $\omega_{G'}(v_1) \leftarrow \omega_G(v_1) + \omega_G(v_3) - \omega_G(v_2)$ and $\omega_{G'}(v_5) \leftarrow \omega_G(v_5) + \omega_G(v_3) - \omega_G(v_4)$.

We obtain $MWIS(G)$ $\mathcal{I}$ from an $MWIS(G')$ $\mathcal{I}'$ in the following way:

If $\mathcal{I}'$ contains neither v_1 nor v_5, then $\mathcal{I} = (\mathcal{I}' \setminus \{v_3\}) \cup \{v_2, v_4\}$.

If $\mathcal{I}'$ contains both v_1 and v_5, then $\mathcal{I} = \mathcal{I}' \cup \{v_3\}$.

If $\mathcal{I}'$ contains v_1 but not v_5, then $\mathcal{I} = \mathcal{I}' \cup \{v_4\}$.

If $\mathcal{I}'$ contains v_5 but not v_1, then $\mathcal{I} = \mathcal{I}' \cup \{v_2\}$.

Additionally, it holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(v_2) + \omega_G(v_4) - \omega_G(v_3)$.

Reduction 14 (5-Cycle [10])
Let there be a 5-cycle $(v_1, v_2, v_3, v_4, v_5)$ such that $d(v_2) = d(v_3) = d(v_5) = 2$, $\min\{d(v_1), d(v_4)\} \geq 3$, $\omega(v_3) \leq \omega(v_2) \leq \omega(v_1)$ and $\omega(v_3) \leq \omega(v_4) \leq \omega(v_5)$. Then G' is obtained by the following: Remove the vertices v_2 and v_3 and update the weights

$\omega_{G'}(v_1) \leftarrow \omega_G(v_1) - \omega_G(v_2)$,

$\omega_{G'}(v_4) \leftarrow \omega_G(v_4) - \omega_G(v_3)$, *and*

$\omega_{G'}(v_5) \leftarrow \omega_G(v_5) - \omega_G(v_3)$.

We obtain $MWIS(G)$ $\mathcal{I}$ from an $MWIS(G')$ $\mathcal{I}'$ in the following way:

If $\mathcal{I}'$ contains both v_1 and v_4, then $\mathcal{I} = \mathcal{I}'$.

If $\mathcal{I}'$ contains v_1 but not v_4, then $\mathcal{I} = \mathcal{I}' \cup \{v_3\}$.

Otherwise, $\mathcal{I} = \mathcal{I}' \cup \{v_2\}$.

It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(v_2) + \omega_G(v_3)$.

Cliques. In any clique $C \subseteq V_G$, at most one vertex $v \in C$ can be part of an $MWIS(G)$. If a vertex $u \in V_G$ has a *clique neighborhood*, then we can exclude any vertex $x \in N(u)$ that does not offer a higher weight than $\omega(u)$, because it could always be replaced by u in the solution.

Definition 7 (Isolated Vertex [13])
For any vertex $u \in V$, if $N[u]$ is a clique (i.e., $G[N[u]]$ is a complete graph), then u is called an isolated vertex[3]*.*

Reduction 15 (Isolated Weight Transfer [13])
Let $u \in V$ be an isolated vertex. Then G' is obtained from G by removing u and applying the following change to each $x \in N_G(u)$:

1. *If $\omega_G(x) \leq \omega_G(u)$, then remove x.*
2. *Otherwise, decrease the weight of x to $\omega_{G'}(x) \leftarrow \omega_G(x) - \omega_G(u)$.*

Let $\mathcal{I}'$ be an $MWIS(G')$. Then $MWIS(G)$ $\mathcal{I}$ can be obtained from the following: It holds that if $x \in \mathcal{I}'$ for any remaining $x \in N_{G'}(u)$, then $\mathcal{I} = \mathcal{I}'$, otherwise $\mathcal{I} = \mathcal{I}' \cup \{u\}$. Additionally, $\alpha_\omega(G) = \alpha_\omega(G') + \omega_G(u)$.

[3]Isolated vertices are also known as *simplicial* vertices.

4.2 Struction Reductions

Variants of STRUCTION[4] reductions are some of the most powerful known reduction rules. They can be used to strictly reduce the size of a graph, but they can also be used to transform the graph nontrivially in ways which increase the number of vertices. The effectiveness of these rules in a cyclic reduction algorithm for MWIS has already been demonstrated by Gellner et al. in [7].

Extended Struction.

Reduction 16 (Extended Struction [7])
Let $v \in V$ be some vertex in G. We denote with $\mathcal{C}$ the set of all independent sets S in the induced subgraph $G[N(v)]$ such that $\omega(S) > \omega(v)$. We obtain G' by applying the following changes: Remove the vertices $N[v]$. For each set S of the sufficiently heavy independent sets $\mathcal{C}$, add a new vertex v_S and set its weight to $\omega_{G'}(v_S) \leftarrow \omega_G(S) - \omega_G(v)$. For each set $S \in \mathcal{C}$ and each vertex $x \in N_G(S) \setminus N_G[v]$, add the edge $e(v_S, x)$. Lastly, for each pair of sets $\{S, S'\} \subseteq \mathcal{C}$, add the edge $e(v_S, v_{S'})$, so that the newly added vertices form a clique. From an $MWIS(G')$ $\mathcal{I}'$, we obtain an $MWIS(G)$ $\mathcal{I}$ as follows: If $v_S \in \mathcal{I}'$ for any $S \in \mathcal{C}$, then $\mathcal{I} = (\mathcal{I}' \setminus \{v_S\}) \cup S$, otherwise $\mathcal{I} = \mathcal{I} \cup \{v\}$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$.

Extended Reduced Struction. For any vertex $v \in V$, the previous reduction, EXTENDED STRUCTION, creates a new vertex for *every* independent set in $G[N(v)]$ which has a weight that exceeds $\omega(v)$. The authors state that this can result in up to $\mathcal{O}(2^{d(v)})$ new vertices [7]. Therefore, they also propose a variant, EXTENDED REDUCED STRUCTION, which has the potential for adding fewer vertices by initially creating a new vertex for only a subset of these heavy independent sets in $G[N(v)]$.

A Subset of Heavy Independent Sets. For a vertex $v \in V$, let $\mathcal{C}$ be defined as in Reduction 17 again, i.e., the set of all independent sets S in $G[N(v)]$ with $\omega(S) > \omega(v)$. Let there be some fixed ordering[5] to the vertices, and let $\max(S)$ denote the vertex in S that comes last in the ordering.

Then the set of all independent sets in $G[N(v)]$ that are *just* heavier than $\omega(v)$ is $\mathcal{D} = \{S \in \mathcal{C} \mid (S \setminus \{\max(S)\}) \notin \mathcal{C}\}$. In other words, a set $S \in \mathcal{C}$ is omitted in $\mathcal{D}$ if $\omega(S) - \omega(\max(S))$ is already greater than $\omega(v)$, and the prefix set $S \setminus \{\max(S)\}$ is therefore also in $\mathcal{C}$.

While $|\mathcal{D}| \leq |\mathcal{C}|$ and we might therefore be able to create fewer vertices initially, EXTENDED REDUCED STRUCTION requires adding an additional set of vertices to complete the modification. The full procedure for the reduction is as follows:

[4]The name STRUCTION derives from "**ST**ability number **R**ed**UCTION**". The predecessor to the STRUCTION reductions used here was originally conceived [5] for the MIS problem, where the term *stability number* refers to the size of the largest independent set in a graph.

[5]Our implementation uses the labels of the vertices, which are integers, for the ordering.

Reduction 17 (Extended Reduced Struction [7])
Given a vertex $v \in V$ and a set of independent sets $\mathcal{D}$ as described above. We construct the vertices v_S for each $S \in \mathcal{D}$ in the same way as EXTENDED STRUCTION *does for each $S \in \mathcal{C}$. Then we additionally create vertices $v_{S,y}$ for pairs of sets $S \in \mathcal{D}$ and vertices $y \in N_G(v)$ where $y \notin N_G[S]$. Note that $S \cup \{y\}$ is then an independent set that is in $\mathcal{C}$ but not in $\mathcal{D}$. For two such vertices $v_{S,y}$ and $v_{S',y'}$, we add the edge $e(v_{S,y}, v_{S',y'})$ if either $S \neq S'$ or $y \in N_G(y')$. Additionally, add an edge between each such vertex $v_{S,y}$ and every vertex $x \in (N_G(S) \cup N_G(y)) \setminus N_G[v]$. Finally, add an edge between each pair of vertices v_S and $v_{S',y}$ if $S \neq S'$. Given an MWIS(G') $\mathcal{I}'$, we obtain an MWIS(G) $\mathcal{I}$ as follows: If $v_S \in \mathcal{I}'$ for some $S \in \mathcal{D}$, then $\mathcal{I} = (\mathcal{I}' \cap V_G) \cup S \cup \{y \mid v_{S,y} \in \mathcal{I}'\}$. Otherwise, $\mathcal{I} = \mathcal{I}' \cup \{v\}$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') + \omega(v)$.*

Change in graph size. Both versions of STRUCTION are capable of increasing or decreasing the size of the graph. One application of the rule to a vertex $v \in V$ can decrease n by up to $d(v) + 1$ (if v is a *heavy vertex*), but depending on how many heavy independent sets exist in $G[N[v]]$, it can also increase it by much more.

A transformation is called *decreasing* if $n_{G'} < n_G$, *increasing* if $n_{G'} > n_G$, and a *plateau* transformation if n stays the same.

Whenever a check for a STRUCTION reduction is performed, there is a set limit for by how much the graph may increase. While enumerating the independent sets $S \in G[N(v)]$ with $\omega(S) > \omega(v)$, the number of vertices that would get added is compared against this increase limit, reduced by the number of vertices $d(v)+1$ that would get deleted. This way, the search for independent sets in $G[N(v)]$ can terminate early if applying the reduction would create too many new vertices. The limit for how much to increase n by is -1 during the decreasing phase, and regulated by a parameter of the algorithm during the increasing phase, set individually for the two variants.

During the increasing phase, we allow for transformations where n stays the same (plateau transformations), but disallow any decrease in n to ensure that the increasing phase can terminate by reaching the desired target size.

Complexity of Enumerating Independent Sets. It is important to note that the step of finding $\mathcal{C}$, the set of all independent sets in a subgraph of G with weight greater than some threshold, is, in the general case, as complex as solving the MWIS problem itself, meaning it is also *NP*-hard . It is therefore important to restrict the application of struction rules to only vertices v with degree below some constant limit.

4.3 Edge Deletion Reductions

Typically, reduction rules are designed to reduce the number of *vertices* in a graph G. In this section, we present a different kind of reduction rule focused solely on reducing the number of *edges* in G instead. The first two rules are simple variants of V-SHAPE MAX and V-SHAPE MID and only delete optional edges from G, while the third rule allows us to rewrite an edge $e(x, y)$ when x dominates y.

Optional Edge Deletion. These reductions can be applied when V-SHAPE MAX and V-SHAPE MID could also be applied, and in fact, still allow these reductions to be applied afterwards. Their purpose is to serve as an intermediate step, resulting in a more sparse graph and potentially allowing for different reductions to become applicable instead.

Reduction 18 (Optional Edge Deletion 1)
Given a degree-2 vertex v with nonadjacent neighbors $N(v) = \{x, y\}$, $e(x, y) \notin E$, such that $\max(\omega(x), \omega(y)) \leq \omega(v) < \omega(x) + \omega(y)$. Then for each vertex $u \in (N(x) \cap N(y)) \setminus \{v\}$, remove either $e(u, x)$ or $e(x, y)$. It holds that a MWIS(G') is also a MWIS(G), and $\alpha_\omega(G) = \alpha_\omega(G')$.

Reduction 19 (Optional Edge Deletion 2)
Given a degree-2 vertex v with nonadjacent neighbors $N(v) = \{x, y\}$, $e(x, y) \notin E$, such that $\omega(x) \leq \omega(v) < \omega(y)$. Then for each vertex $u \in (N(x) \cap N(y)) \setminus \{v\}$, remove the edge $e(u, x)$. It holds that a MWIS(G') is also a MWIS(G), and $\alpha_\omega(G) = \alpha_\omega(G')$

The result of both of the above reductions can also appear when applying V-SHAPE MAX or V-SHAPE MID followed by their respective backward reduction rules (defined in the following section). Our implementation contains procedures for both OPTIONAL EDGE DELETION reductions anyhow, for use in the decreasing phase of our algorithm.

Domination Edge Deletion. Consider two adjacent vertices $\{x, y\} \subseteq V$, $e(x, y) \in E$. If x is adjacent to at least all the vertices that y is adjacent to (except for x itself), i.e., if $N[x] \supseteq N[y]$, then we say that x *dominates* y [13]. If $\omega(x) \leq \omega(y)$, this is a sufficient condition to remove x from the graph (cf. the DOMINATION reduction in [13]). This case is also covered by the BASIC SINGLE-EDGE reduction (Reduction 4), which is used in our algorithm.

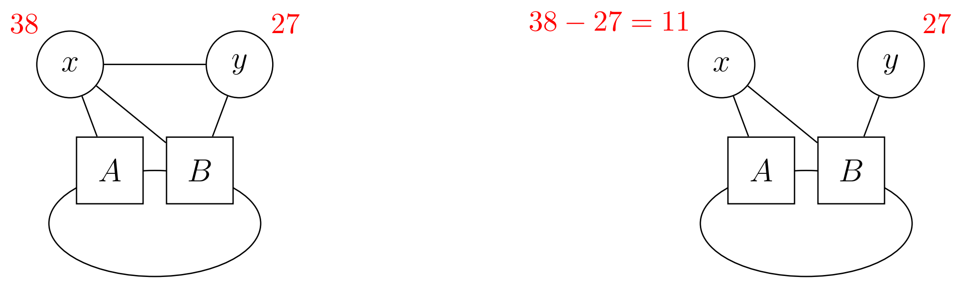
Reduction 20 (Domination Edge Deletion)
Given two vertices x and y such that x dominates y (i.e., , $N[x] \supseteq N[y]$) and $\omega(x) > \omega(y)$. Remove the edge $e(x, y)$ and decrease the weight of x by $\omega_{G'}(x) \leftarrow \omega_G(x) - \omega(y)$. We obtain MWIS$(G)$ $\mathcal{I}$ from a MWIS(G') $\mathcal{I}'$ as follows: If $\{x, y\} \subseteq \mathcal{I}'$ then $\mathcal{I} = \mathcal{I}' \setminus \{y\}$, otherwise $\mathcal{I} = \mathcal{I}'$. It holds that $\alpha_\omega(G) = \alpha_\omega(G')$.

4.4 Backward Reduction Rules

4.4.1 Weight Generation

Over the course of a run of this algorithm, it is always ensured that all vertex weights are positive integers. When generating weights for new vertices in backward reductions, they

Domination Edge Deletion



(a) G (before DOMINATION EDGE DELETION)

(b) G' (after DOMINATION EDGE DELETION)

Figure 4.1: (a) The closed neighborhood of y, $N[y] = \{x, y\} \cup B$, is a subset of the closed neighborhood of x, $N[x] = \{x, y\} \cup A \cup B$, and $\omega(x) > \omega(y)$ so that x is not outright removable; (b) The edge linking x and y has been removed, and the weight of x has been reduced. The inclusion of both x and y in an $MWIS(G')$ now represents the inclusion of just x in an $MWIS(G)$.

are typically selected randomly and uniformly from an interval $[a, b]$. Both a and b may be restricted by the weight of other vertices, and $a \geq 1$ must always hold.

However, there are often no restrictions on how large the upper bound b could be. It is preferable that the backward reduction rules behave similarly from graph to graph, even if the scale of weights is different between them. Therefore, we want to set b relative to the pre-existing weights in the graph.

Our initial experiments have shown that the arithmetic mean of the weights in the original graph is well suited as an anchor, relative to which we set b. For each reduction, there is a parameter which is a factor that controls how large relative to the anchor we select b for this reduction.

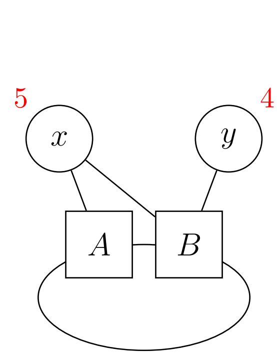
4.4.2 Backward V-Shape Mid Reductions

Two of the most important backward reductions that we are presenting are two variants of BACKWARD V-SHAPE MID. These can be applied when for two vertices x and y, all the vertices adjacent to y are also adjacent to x (except for self-loops).

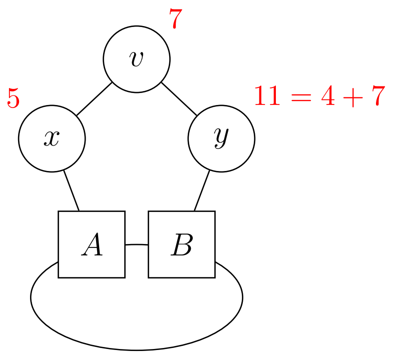
A notable effect of the reduction is that edges between x and $u \in N_G(y)$ become redundant and can be removed. Therefore, even though the new vertex v gets added to both $N_{G'}(x)$ and $N_{G'}(y)$, the degree of x can shrink by up to $d_G(y) - 1$, meaning the graph G' can contain fewer edges than G.

Backward Reduction 1 (Backward V-Shape Mid 1)
Required: *A pair of nonadjacent vertices* $\{x, y\} \subseteq V$, $e(x, y) \notin E$, *such that* $N(y) \subseteq N(x)$.

Backward V-Shape Mid 1



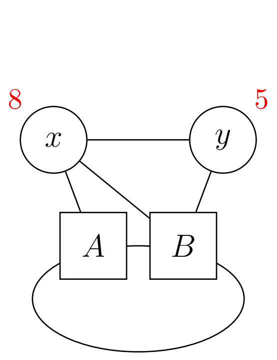
(a) G (before BACKWARD V-SHAPE MID 1)



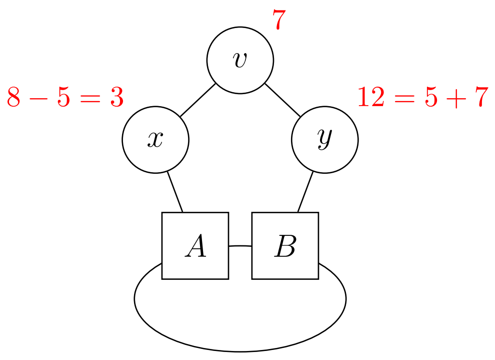
(b) G' (after BACKWARD V-SHAPE MID 1)

Figure 4.2: (a) The graph G contains two nonadjacent vertices x and y, where the neighborhood of y, which is $N(y) = B$, is a subset of $N(x) = A \cup B$; (b) A vertex v has been added and made adjacent to x and y, the weight of y has been increased, and the edges $e(x, b)$ for $b \in B$ became optional and were removed.

Backward V-Shape Mid 2



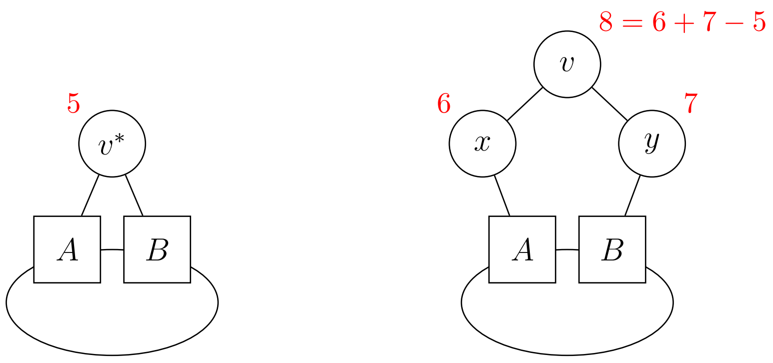
(a) G (before BACKWARD V-SHAPE MID 2)



(b) G' (after BACKWARD V-SHAPE MID 2)

Figure 4.3: (a) The graph G contains the edge $e(x, y)$ between x and y with $\omega(x) > \omega(y)$, and the closed neighborhood of y, which is $N[y] = B \cup \{x, y\}$, is a subset of $N[x] = A \cup B \cup \{x, y\}$; (b) The weight of x has been decreased by $\omega_G(y)$, the edge $e(x, y)$ was removed, a vertex v has been added and made adjacent to x and y, the weight of y has been increased by $\omega_{G'}(v)$, and the edges $e(x, b)$ for $b \in B$ became optional and were removed.

Backward V-Shape Max



(a) G (before BACKWARD V-SHAPE MAX) **(b)** G' (after BACKWARD V-SHAPE MAX)

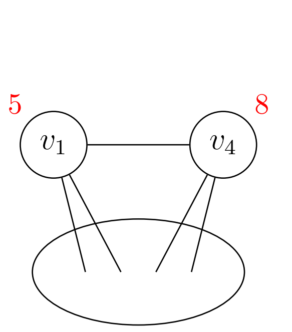
Figure 4.4: (a) The graph G contains some vertex v^*, in this example with neighbors $N(v^*) = A \cup B$; (b) The graph G' after replacing v^* with the v-shape structure $\{v, x, y\}$ and redistributing the neighbors of v^* to x and y.

Obtain G' as follows: Add a new vertex v and select $\omega_{G'}(v)$ such that $\omega_{G'}(v) > \omega_{G'}(x)$. Update $\omega_{G'}(y) \leftarrow \omega_G(y) + \omega_{G'}(v)$. Add the edges $e(v,x)$ and $e(v,y)$. For each $u \in N_G(x) \cap N(y) \setminus \{v\}$, the edge $e(u,x)$ is optional and can be removed. Then it holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v)$. All MWIS$(G')$ contain either $\{v\}$, $\{y\}$ or $\{x,y\}$. The solution set MWIS(G) $\mathcal{I}$ can be obtained from a MWIS(G') $\mathcal{I}'$ by removing v, if it is present: $\mathcal{I} = \mathcal{I}' \setminus \{v\}$.

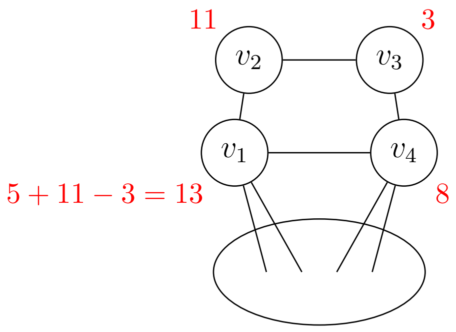
The second variant of BACKWARD V-SHAPE MID is a combination of DOMINATION EDGE DELETION (Reduction 20) and the first variant, allowing us to also apply the reduction to adjacent vertices x and y:

Backward Reduction 2 (Backward V-Shape Mid 2)
Required: *A pair of adjacent vertices $\{x, y\} \subseteq V$, $e(x,y) \in E$ with $\omega(x) > \omega(y)$, such that x dominates y (i.e., $N[x] \supseteq N[y]$).*
First, remove the edge $e(x,y)$ and update the weight of x with $\omega_{G'}(x) \leftarrow \omega_G(x) - \omega_G(y)$. Then, add a new vertex v with $\omega_{G'}(v) > \omega_{G'}(x)$ and update the weight of y with $\omega_{G'}(y) = \omega_G(y) + \omega_{G'}(v)$. Add the edges $e(v,x)$ and $e(v,y)$. For each $u \in N_G(x) \cap N(y) \setminus \{v\}$, the edge $e(u,x)$ is optional and can be removed. Then it holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v)$. From an MWIS$(G')$ $\mathcal{I}'$ we obtain MWIS(G) $\mathcal{I}$ as follows: If $\{x,y\} \subset \mathcal{I}'$, then $\mathcal{I} = \mathcal{I}' \setminus \{y\}$, otherwise $\mathcal{I} = \mathcal{I}' \setminus \{v\}$.

Backward Four-Cycle



(a) G (before BACKWARD 4-CYCLE)



(b) G' (after BACKWARD 4-CYCLE)

Figure 4.5: (a) the graph G contains two adjacent vertices v_1 and v_4. They are in no particular relation to each other regarding their weight; (b) the two vertices v_2 and v_3 have been added, and the weight of v_1 has increased appropriately. The weights of v_1, v_2 and v_3 are sorted $\omega(v_1) > \omega(v_2) > \omega(v_3)$. The four vertices now form a cycle.

4.4.3 Backward Rules Based on Alternative Sets

Backward Reduction 3 (Backward V-Shape Max)
Required: *A vertex $v^* \in V$.*
Obtain G' by modifying G as follows: Remove v^* from G and add three new vertices v, x, y. Select $\omega_{G'}(x)$ and $\omega_{G'}(y)$ with the restriction that $\min\{\omega_{G'}(x), \omega_{G'}(y)\} > \omega_G(v^*)$ and set $\omega_{G'}(v) \leftarrow \omega_{G'}(x) + \omega_{G'}(y) - \omega_{G'}(v^*)$. Add the edges $e(v, x)$ and $e(v, y)$. For each $u \in N_G(v^*)$, add at least one edge out of $e(u, x)$ and $e(u, y)$.

Then it holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v)$. An MWIS$(G')$ $\mathcal{I}'$ contains either v or both $\{x, y\}$. We obtain an MWIS(G) $\mathcal{I}$ as follows: If $\mathcal{I}'$ contains v, then $\mathcal{I} = \mathcal{I}' \setminus \{v\}$, otherwise $\mathcal{I} = (\mathcal{I}' \setminus \{x, y\}) \cup \{v^*\}$.

4.4.4 Expansion of a Single Edge

The following three backward reduction rules modify an edge $e \in E$, inflating it into a bigger structure.

Expansion of an Edge to a 4-Cycle. For an edge $e(v_1, v_4)$, the BACKWARD 4-CYCLE reduction preserves $\omega(v_4)$ while increasing $\omega(v_1)$.

Backward Reduction 4 (Backward 4-Cycle)
Required: *An edge $e(v_1, v_4) \in E$ with $\omega_G(v_1) \geq 2$.*
Add two new vertices v_2 and v_3 and select the weight of v_3 such that $\omega_{G'}(v_3) < \omega_G(v_1)$, then select the weight of v_2 such that $\omega_{G'}(v_2) > \omega_{G'}(v_3)$. Increase $\omega(v_1)$ with $\omega_{G'}(v_1) \leftarrow$

$\omega_G(v_1) + \omega_{G'}(v_2) - \omega_{G'}(v_3)$. Lastly, add the three edges $e(v_1, v_2)$, $e(v_2, v_3)$ and $e(v_3, v_4)$. Then it holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v_2)$ and $MWIS(G) = MWIS(G') \setminus \{v_2, v_3\}$.

Expansion of an Edge to a 3-Path. For an edge $e(v_1, v_4)$, the BACKWARD 3-PATH reduction also preserves $\omega(v_4)$ while increasing $\omega(v_1)$. Note that this backward reduction is the worst performing out of the ones presented here. In Section 5.2.3, we explain our process of eliminating rules that perform badly.

Backward Reduction 5 (Backward 3-Path)
Required: *An edge $e(v_1, v_4) \in E$ such that $\omega_G(v_1) > \omega_G(v_4) + 1$.*
Add two new vertices v_2 and v_3, and select $\omega_{G'}(v_3)$ such that $\omega_{G'}(v_4) < \omega_{G'}(v_3) < \omega_G(v_1)$, then select $\omega_{G'}(v_2)$ such that $\omega_{G'}(v_3) < \omega_{G'}(v_2)$. Update $\omega_{G'}(v_1) \leftarrow \omega_G(v_1) + \omega_{G'}(v_2) - \omega_{G'}(v_3)$, which increases the weight of v_1 by at least one. Now it holds that $\omega_{G'}(v_4) < \omega_{G'}(v_3) < \omega_{G'}(v_2) < \omega_{G'}(v_1)$. Remove $e(v_1, v_4)$ and add the edges $e(v_1, v_2)$, $e(v_2, v_3)$ and $e(v_3, v_4)$. Then it holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v_2)$ and $MWIS(G) = MWIS(G') \setminus \{v_2, v_3\}$.

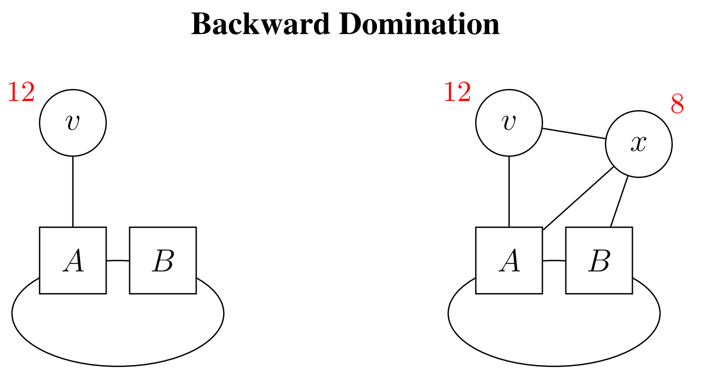
Backward Reduction 6 (Backward 4-Path)
Required: *A 2-path $(v_1, v_3, v_5) \in V^3$ with $d(v_3) = 2$, $e(v_1, v_5) \notin E$ and $\omega(v_3) < \min\{\omega(v_1), \omega(v_5)\}$.*
Add two new vertices v_2 and v_4 to G'. Select $\omega_{G'}(v_2)$ and $\omega_{G'}(v_4)$ such that $\min\{\omega_{G'}(v_2), \omega_{G'}(v_4)\} > \omega(v_3)$. Update $\omega_{G'}(v_1) \leftarrow \omega_G(v_1) + \omega_{G'}(v_2) - \omega(v_3)$ and $\omega_{G'}(v_5) \leftarrow \omega_G(v_5) + \omega_{G'}(v_4) - \omega(v_3)$. Remove the edges $e(v_1, v_3)$ and $e(v_3, v_5)$ and add the new edges $e(v_1, v_2)$, $e(v_2, v_3)$, $e(v_3, v_4)$ and $e(v_4, v_5)$.

Now it holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v_2) - \omega_{G'}(v_4) + \omega(v_3)$. From a solution set $MWIS(G')$, we obtain the solution $MWIS(G)$ as follows: If neither v_1 nor v_5 is in $MWIS(G')$, then $MWIS(G) = (MWIS(G') \setminus \{v_2, v_4\}) \cup \{v_3\}$. Otherwise, $MWIS(G) = MWIS(G') \setminus \{v_2, v_3, v_4\}$.

Backward Reduction 7 (Backward 5-Cycle)
Required: *A 2-path $(v_1, v_5, v_4) \in V^3$ with $d(v_5) = 2$, $e(v_1, v_4) \notin E$.*
Add the two new vertices v_2 and v_3 and select their weights freely. Add the edges $e(v_1, v_2)$, $e(v_2, v_3)$ and $e(v_3, v_4)$. Update the weights with $\omega_{G'}(v_1) \leftarrow \omega_G(v_1) + \omega_{G'}(v_2)$, $\omega_{G'}(v_4) \leftarrow \omega_G(v_4) + \omega_{G'}(v_3)$ and $\omega_{G'}(v_5) \leftarrow \omega_G(v_5) + \min\{\omega_{G'}(v_2), \omega_{G'}(v_3)\}$. Then it holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v_2) - \omega_{G'}(v_3)$ and $MWIS(G) = MWIS(G') \setminus \{v_2, v_3\}$.

4.4.5 Backward Reductions Introducing Removable Vertices

The following two backward reductions, which are similar, each introduce a single removable vertex. The advantage of adding removable vertices stems from the fact that we can generate arbitrary new edges from the new removable vertex to any other vertex, which may change how and in which order other reduction rules get applied.



(a) G (before BACKWARD DOMINATION)

(b) G' (after BACKWARD DOMINATION)

Figure 4.6: (a) The graph G contains some vertex v; (b) The graph G' now additionally contains the removable vertex x. We guarantee that $N[x] \supseteq N[v]$, but x additionally receives edges to vertices $b \in B$.

Backward Reduction 8 (Backward Domination)
Required: *A vertex $v \in V$ with $\omega(v) \geq 2$.*
Add a new vertex x with $\omega_{G'}(x) < \omega(v)$. Add an edge between x and each vertex $y \in N[v]$. Optionally add arbitrary edges from x to any other vertices.

The new vertex x now dominates v and is removable. Thus, it holds that $\alpha_\omega(G) = \alpha_\omega(G')$ and $MWIS(G) = MWIS(G')$.

The following backward reduction is similar, but may connect x with fewer neighbors of v. The key to ensuring that x is removable is making sure that for an independent set S containing x and all the non-neighbors of x in $N(v)$ (i.e., $S = \{x\} \cup (N_G(v) \setminus N_{G'}(x))$), it still holds that $\omega(S) < \omega(v)$.

Backward Reduction 9 (Backward Basic Single-Edge)
Required: *A vertex $v \in V$ with $\omega(v) \geq 2$.*
Add a new vertex x with $\omega_{G'}(x) < \omega(v)$. Continue adding edges between x and vertices $y \in N_G(v)$ until $\omega(N_G(v) \setminus N_{G'}(x)) < \omega(v) - \omega_{G'}(x)$, and add the edge $e(v, x)$. Optionally add edges between x and arbitrary other vertices.

The new vertex x is removable. Thus, it holds that $\alpha_\omega(G) = \alpha_\omega(G')$ and $MWIS(G) = MWIS(G')$.

Backward Triangle Reductions. The forward reduction TRIANGLE is a special case of ISOLATED WEIGHT TRANSFER (Reduction 15) for cliques of size 3.

The following reduction, BACKWARD TRIANGLE MID, can be seen as reversing one such triangle case. The structure we create contains a triangle with a new isolated vertex v, where the weight of the isolated vertex v is inbetween the weight of its newly created neighbor x and its pre-existing neighbor y (i.e., $\omega(x) < \omega(v) < \omega(y)$). This reduction combines aspects of different other expansion rules, as it both increases the weight of the old vertex y, while also adding a removable vertex x with arbitrary edges.

Backward Reduction 10 (Backward Triangle Mid)
Required: *Any vertex $y \in V$.*
Add two vertices v and x and add the edges $e(v, x)$, $e(v, y)$ and $e(x, y)$. Select the weights of v and x so that $\omega_{G'}(x) < \omega_{G'}(v)$ and increase the weight of y: $\omega_{G'}(y) \leftarrow \omega_G(y) + \omega_{G'}(v)$. While v stays a degree-2 vertex, optionally add edges between x and other vertices in G'. Then $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v)$ and $MWIS(G) = MWIS(G') \setminus \{v\}$.

4.4.6 Adding Simplicial Vertices

The following three backward reduction rules each add a single simplicial vertex v to the graph. The simplicial vertex is linked to a single vertex (Backward Reduction 11), to an edge (Backward Reduction 12) or to some maximal clique (Backward Reduction 13) as its only edges. In all cases, the weight of all vertices that are now adjacent to the new simplicial vertex v have their weight increased by $\omega_{G'}(v)$.

Single Vertex Weight Increase.

Backward Reduction 11 (Backward Degree-1 Fold)
Required: *A vertex $x \in V$.*
Generate a new vertex v with any weight and add the edge $e(v, x)$. Update $\omega_{G'}(x) \leftarrow \omega_G(x) + \omega_{G'}(v)$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v)$ and $MWIS(G) = MWIS(G') \setminus \{v\}$.

Adjacent Vertices Weight Increase. The following reduction results in a weight increase in *both* endpoints of an edge, which are extended by a new vertex to form a triangle.

Backward Reduction 12 (Backward Triangle Min)
Required: *An edge $e(x, y) \in E$.*
Add a new vertex v and the edges $e(v, x)$ and $e(v, y)$. Select some weight for v without restrictions and increase the weights of x and y by $\omega_{G'}(v)$. Then $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v)$ and $MWIS(G) = MWIS(G') \setminus \{v\}$.

Maximal Clique Weight Increase. For this reduction, we first select a starting vertex x^* and then perform a greedy search to find a maximal clique in $N[x^*]$.

Backward Reduction 13 (Backward Simplicial Weight Transfer)
Let $X \subset V$ be some maximal clique in V. Add a new vertex v and assign an unconstrained weight $w_{G'}(v)$ to it. For each $x \in X$, add the edge $e(v, x)$ and increase the weight of x by $\omega_{G'}(v)$. It holds that $\alpha_\omega(G) = \alpha_\omega(G') - \omega_{G'}(v)$ and $MWIS(G) = MWIS(G') \setminus \{v\}$.

4.4.7 Translation between Solutions

After solving the MWIS problem on a modified graph G' and obtaining $MWIS(G')$ $\mathcal{I}'$, we need to translate $\mathcal{I}'$ into a solution set for the original graph G, i.e., a $MWIS(G)$ $\mathcal{I}$. In general, we formulate the backward reduction rules in such a way that it is simple to obtain the actual solution $\mathcal{I}$ from the solution to the inflated graph $\mathcal{I}'$.

Strict Weight Relationships. For an example of how this is achieved, consider our definition of BACKWARD 3-PATH: Whereas the forward reduction 3-PATH (Reduction 11) only requires $\omega_G(v_4) \leq \omega_G(v_3) \leq \omega_G(v_2) \leq \omega_G(v_1)$, we ensure that the backward reduction results in the stricter weight relationships $\omega_{G'}(v_4) < \omega_{G'}(v_3) < \omega_{G'}(v_2) < \omega_{G'}(v_1)$.

The reason for this is as follows: Let $\mathcal{I}'$ be a $MWIS(G')$ and let $S^* = \mathcal{I}' \cap \{v_1, v_2, v_3, v_4\}$. Because none of the weights of $\{v_1, v_2, v_3, v_4\}$ are equal, we can deduce some information about the rest of $\mathcal{I}'$ from just S^*. For example, if $v_2 \in S^*$, then we know that at least one other neighbor of v_1 must also be in $\mathcal{I}'$. If this were not the case, $(\mathcal{I}' \setminus \{v_2\}) \cup \{v_1\}$ would be a heavier independent set in G' than $\mathcal{I}'$, which would be a contradiction.

This knowledge is useful, because it makes it easier to translate $\mathcal{I}'$, which might contain v_2, into a solution $MWIS(G)$ for the original graph G, where a vertex v_2 does not exist in the vertex set V_G: The vertices v_2 and v_3, if they are contained in $\mathcal{I}'$, can simply be deleted to obtain $\mathcal{I}$.

It would theoretically be possible to define a backward reduction rule to 3-PATH in such a way to allow equal weights for $\{v_1, v_2, v_3, v_4\}$. It should still be possible to find an equivalent solution, since the forward rule 3-PATH can apply to such cases. However, deductions such as in the previous example would no longer be possible, and the process of reconstructing the solution $MWIS(G)$ from an $MWIS(G')$ would be significantly complicated.

In general, we define all our backward rules in such a way that the weight of newly generated vertices v does not equal the weight of a pre-existing vertex among their neighbors $u \in N_{G'}(v)$. This avoids any issues with the translation between solution sets.

4.5 Decreasing Phase

Exhaustive Application. When making use of a reduction rule to decrease the size of a graph, we typically iterate over every $v \in V$ and check if the rule is applicable to v and its neighborhood.

An exception to this is the CWIS reduction, which is the only *global* reduction in our selection. If we have checked the entire graph and no change has been made, then we know that this reduction cannot apply to the graph in its current state. The reduction becomes disabled and will not be checked again until the graph has changed, either through a different forward reduction rule, or after a blowup-phase of the cyclic algorithm. When all of the forward reduction rules have been disabled for this reason, the resulting kernel of the graph is called *exhaustively reduced* with respect to our set of forward reduction rules.

Randomized Rule Order. In other works dealing with graph reduction rules, the rules are typically applied in a predefined order. To minimize running time or resulting kernel size, simple and effective rules are applied first [20][8].

In our algorithm, however, we instead opt for a partially randomized application of reduction rules. If a predefined order was used, it would potentially undermine some of the backward reduction rules. For example, if every reducing phase were to always begin with the NEIGHBORHOOD REMOVAL reduction, then this might negatively influence the effectiveness of a BACKWARD NEIGHBORHOOD REMOVAL rule in the increasing phase.

Therefore, every time a new rule needs to be selected, it is picked uniformly at random from a set of reduction rules which are not currently disabled.

There is, however, an exception to this: The most expensive reductions, those being CRITICAL WEIGHTED INDEPENDENT SET, DECREASING EXTENDED STRUCTION and DECREASING EXTENDED REDUCED STRUCTION, are only applied after the other forward rules have been exhausted, and are checked in this order. Preliminary experiments have shown that treating this group of expensive reductions separately reduces the running time of each decreasing phase.

Effectively, this splits the decreasing phase into two sub-phases: First a set of simpler reduction rules gets applied in a randomized sequence, then a second set of more computationally expensive reduction rules gets applied in a fixed order. If the graph changes during the second sub-phase, the first sub-phase is restarted.

Combination of Rules. To minimize the running time of the decreasing phase, we decided to bundle some reduction rules together. When checking if a reduction rule is applicable and iterating over the vertices, the conditions for multiple reduction rules are tested in sequence. This allows us to avoid many duplicate checks for certain conditions.

The DECREASING LOW-DEGREE Rule combines the rules NEIGHBORHOOD REMOVAL, DEGREE-1 FOLD and TRIANGLE.

The COMPLICATED DEGREE-2 Rule combines V-SHAPE MAX, 3-PATH, 4-CYCLE, 4-PATH and 5-CYCLE rules. These all involve degree-2 vertices and have significant overlap between their conditions.

Another time-save can be found in the combination of TWO-VERTEX + TWIN. Both reductions operate on *2-hop neighbors*, which are pairs of vertices v and $x \in V$, $v \neq x$ such that $e(v, x) \notin E$, but v and x share at least one common neighbor. These reductions are combined, because the condition for the TWIN rule can also be checked for free as part of the calculations for the TWO-VERTEX reduction.

Decreasing Process. As long as at least one reduction rule is eligible, we follow the following process: We select the next reduction rule to apply. If at least one of them is eligible, we randomly pick an eligible rule from the first set of simpler reduction rules. This set consists of:

- DECREASING LOW-DEGREE

- COMPLICATED DEGREE-2
- TWO-VERTEX + TWIN
- OPTIONAL EDGE DELETION
- ISOLATED WEIGHT TRANSFER
- DECREASING LOW-DEGREE
- BASIC SINGLE-EDGE
- EXTENDED SINGLE-EDGE
- UNCONFINED

If none of these is eligible, we pick the first eligible reduction rule in the following list:

- CRITICAL WEIGHTED INDEPENDENT SET
- NON-INCREASING EXTENDED STRUCTION
- NON-INCREASING EXTENDED REDUCED STRUCTION

Whenever a rule is selected, we search the entire graph and apply it to every location where it can be applied. When it cannot be applied to a graph at all, it becomes temporarily ineligible for selection. Whenever a reduction is applied and the graph has changed, all reduction rules become eligible again. When no reduction rule is eligible anymore, this implies that none can be applied. We consider the graph an exhaustively reduced kernel and end the decreasing phase.

4.6 Increasing Phase

Termination. Every increasing phase starts with at least one vertex present in the graph and ends when the size n of the graph G reaches a specified target size n_{target}. In most cases, this is the only termination condition that is needed, since the graph size never decreases in this phase, and many rules which increase the graph size, like for example the BACKWARD DEGREE-1 FOLD rule, can always be applied. As an alternative condition, we end the increasing phase when we were unable to apply a backward rule 50 times during one increasing phase. This allows us to run variants of the algorithm with a limited set of backward reduction rules, which can only be applied in certain conditions.

Set of Rules in the Increasing Phase. The set of available backward reduction rules consists of nondecreasing variants of the two STRUCTION reduction rules, which had their definitions given in Section 4.2, and the 15 backward reduction rules which we introduced in Section 4.4. They are:

- NON-DECREASING EXTENDED STRUCTION

- Non-Decreasing Extended Reduced Struction
- Backward V-Shape Mid 1 & 2
- Backward 3-Path
- Backward 4-Path
- Backward 4-Cycle
- Backward 5-Cycle
- Backward Degree-1 Fold
- Backward V-Shape Max
- Backward Triangle Mid
- Backward Simplicial Weight Transfer
- Backward Triangle Min
- Backward Domination
- Backward Basic Single-Edge

Rule Application Step. An individual step in the increasing phase involves choosing a reduction rule uniformly at random out of the set of available rules.

After selecting a rule, we search for a suitable part of the graph to apply this rule to, according to the configuration of the individual backward rule. If such a part cannot be found with a reduction-specific search, a different rule is selected. When a suitable region to modify with the selected rule has been found, then the rule is applied just *once*, which concludes the step.

This is different from how we apply forward reduction rules in the increasing phase, where after we choose a forward reduction rule, we iterate over all vertices and apply this rule wherever it is possible to apply. This is because most backward reduction rules have low requirements, and could even be applied to the same region indefinitely. Therefore, changing the backward reduction rule after each individual rule application allows us to explore a greater variety of expanded graphs.

4.7 The Complete Algorithm

Putting all the elements presented in this chapter together, we obtain our final cyclic iteration routine.

We start by reading the original graph G_o from a file, and verifying that it is a simple undirected graph without self-loops. Optionally, randomized weights can be assigned in this step, although our experiments all use pre-assigned weights.

We proceed by following the steps of the decreasing phase for the initial reduction, and store the obtained kernel K as the currently best found solution K^*. Afterwards, the cyclic iteration begins: We fix a target size of the vertex set $n_{target} \leftarrow n+a$, where a is a parameter

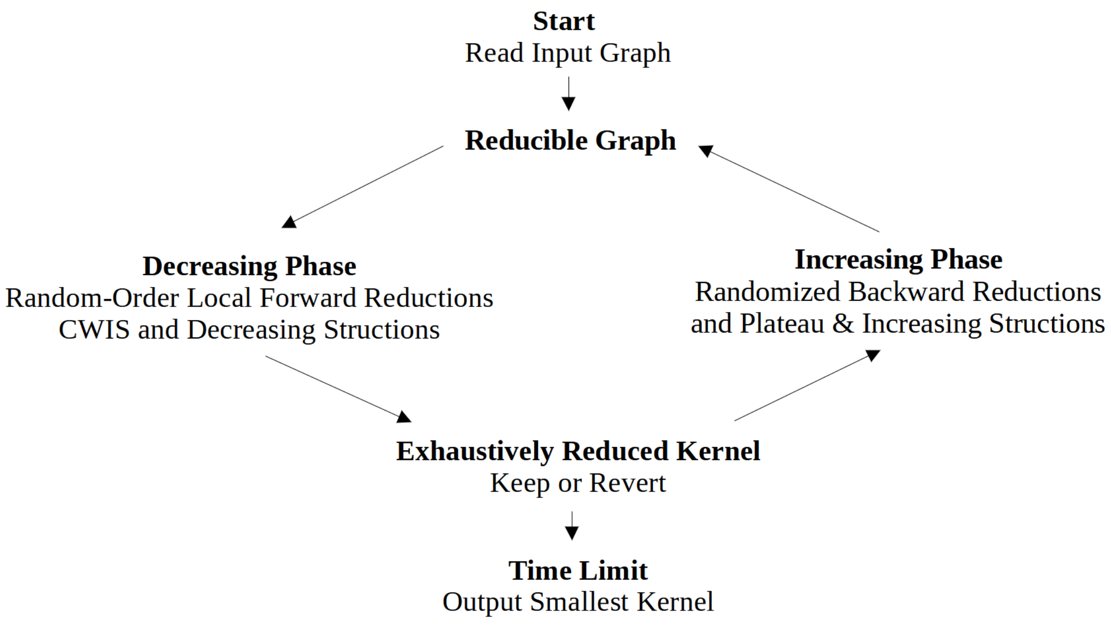

Start
Read Input Graph
Reducible Graph
Decreasing Phase
Random-Order Local Forward Reductions
CWIS and Decreasing Structions
Increasing Phase
Randomized Backward Reductions
and Plateau & Increasing Structions
Exhaustively Reduced Kernel
Keep or Revert
Time Limit
Output Smallest Kernel

Figure 4.7: Outline of the phase transitions in our algorithm

that controls the amount to which the graph gets expanded. Then we apply the increasing phase and then the decreasing phase as previously described. If the resulting kernel K' now contains fewer vertices than the previously found smallest kernel K^*, then we store the new kernel: $K^* \leftarrow K'$. In any case, the increasing phase of the next cycle starts with a copy of K^*. This means that if after the decreasing phase of a cycle, the resulting kernel is not smaller than the previous best (i.e., smallest) kernel, then it is discarded and the recursion continues starting with K^* another time.

When the time limit is met, the algorithm terminates and the current best solution K^* is returned. Additionally, a solution translation file is written, which can be used to obtain a solution for the original graph $\mathit{MWIS}(G_o)$ from a solution to the kernel $\mathit{MWIS}(K^*)$. If at any point during a decreasing phase, n becomes zero, then the algorithm terminates immediately, and the solution translation file will contain the solution directly.

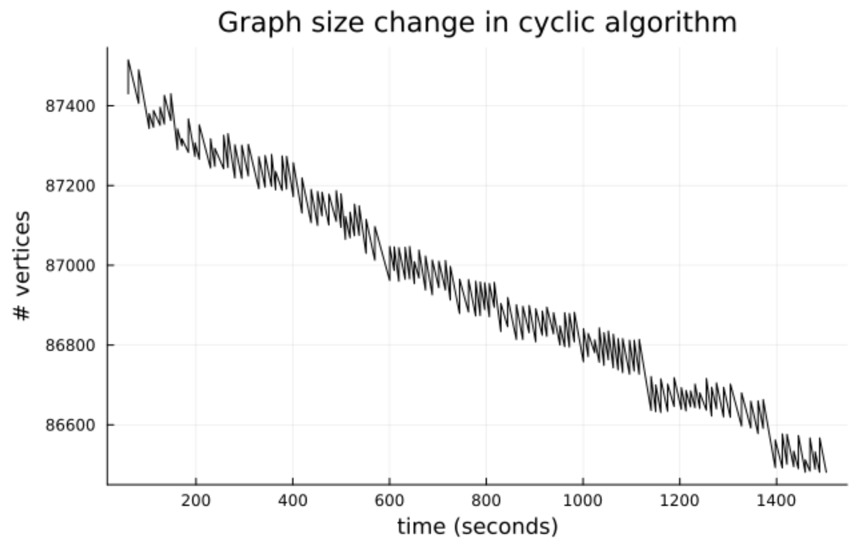


Figure 4.8: An example of applying our cyclic algorithm on the `fe_rotor-uniform` instance, showing the progress of n over 110 cycles in 1500 seconds. Because of the scale of n, the initial reduction from 99 617 to 87 430 vertices before the first increasing phase is not displayed here.

CHAPTER 5

Experimental Evaluation

5.1 Computing Environment

All experiments were performed on a machine equipped with a Intel Xeon Silver 4216 processor with 16 cores running at 2.10Ghz and 100 GB of RAM. It runs Ubuntu 20.04.2 with Linux kernel version 5.4.0-152-generic.

We are using GNU Parallel [19] to compute multiple instances in parallel. For instances where there is not enough RAM available to compute them in parallel with others, they are computed sequentially afterwards.

Compilation. All algorithms were compiled using GCC 9.4.0 at optimization level –O3. The C++ standard used for our cyclic algorithm was C++20, for all other algorithms used in the evaluation the C++11 standard is used.

5.2 Parameter Tuning

5.2.1 Preparing the Tuning Dataset

To adjust the parameters, we needed to get results from our algorithm on a variety of graphs, forming our tuning dataset. To avoid overtuning affecting our evaluation, we chose to not include any of the graphs that are later used in evaluating the algorithm's performance (see Section 5.3).

Instead, we chose a separate dataset from the *10th DIMACS Implementation Challenge* [1]. Initially, we considered 56 graphs from this collection. Since they are unweighted, we assigned new weights to the vertices of each graph in one of the following ways: Random weights from a discrete uniform distribution in the intervals between $[1, 50]$

and $[1, 1000]$, and random weights from a binomial distribution[1] with success probability 0.5 and sample sizes 25 and 500.

The resulting graphs with generated weights were then filtered down to the much smaller final training dataset. We decided on a time limit of 1000 seconds for each parameter tuning experiment, and therefore chose to exclude any instances which were either too simple (i.e., fully solvable in less than 1000 seconds) or too complex (i.e., either allowing for less than 50 cycles of our algorithm within the allotted time, or not being reactive to any of the used forward reduction rules) given the time limit. Most of the prepared instances were discarded here, as a majority was immediately solvable without making use of the cyclic algorithm, and several others completed less than 10 cycles of our cyclic algorithm within the time limit, or did not decrease in size through any of the forward reductions.

The final tuning dataset contains examples of graphs with randomly generated points (`delaunay` and `rgg`), two road network graphs (`osm`), a circuit simulation graph (`G3_circuit`)[3], an internet connectivity graph (`caidaRouterLevel`) and a numerical simulation graph (`NACA0015`). In total, the final tuning dataset contains 20 instances.

Table B.1 lists information about the 20 selected instances and their method of weight generation.

5.2.2 Backward Rule Tuning

Keeping Optional Edges in Backward V-Shape Mid. In both versions of our BACKWARD V-SHAPE MID reductions (Reductions 1 and 2), any edge between x and a vertex $u \in N_{G'}(x) \cap N_{G'}(y) \setminus \{v\}$ is redundant and can optionally be removed.

We implemented a probability parameter p for both BACKWARD V-SHAPE MID reductions which controls how often we remove this optional edge. Every time one of the two backward reductions is applied, for each optional edge $e = e(x, u)$, we choose independently with probability p if the edge e should be kept.

We performed experiments to determine a good value for this parameter p, the results of which can be seen in Figure 5.1.

Intuitively, one might assume that fewer edges (and therefore a more sparse problem instance) would always be better. Perhaps surprisingly, removing optional edges in $100\ \%$ of cases turned out to be the worst option out of the ones we tested, even worse than never removing optional edges. The average size of the reduced graphs was smallest when p was set to 0.4.

Struction Maximum Increase. The decreasing phase makes use of the INCREASING variants of the EXTENDED STRUCTION and EXTENDED REDUCED STRUCTION reductions. For both, we need to restrict the number of vertices that we allow n to increase by in

[1]Since a binomial distribution can return a result of 0 and we only allow weights greater than 0, we changed any randomly generated 0-weight to 1.

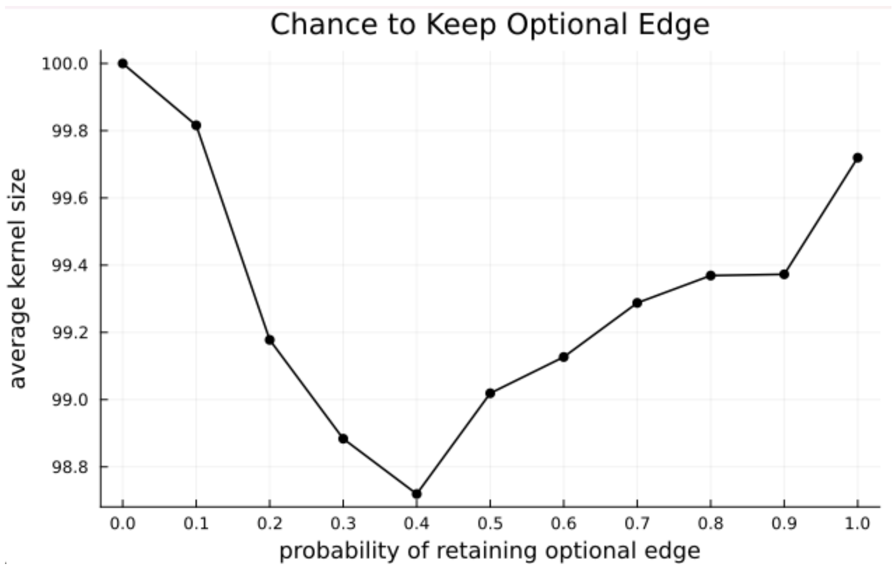


Figure 5.1: Plot showing the average size of remaining kernels of tuning dataset instances when running our algorithm for 1000 seconds for different probabilities p to keep an optional edge in BACKWARD V-SHAPE MID 1 and BACKWARD V-SHAPE MID 2.

a single modification. Our experiments indicate that we achieve the best results when we only allow small increases.

As can be seen in Table 5.1, the best limit for the increase in number of vertices is 4 for INCREASING EXTENDED STRUCTION. For INCREASING EXTENDED REDUCED STRUCTION, allowing an increase of up to just 1 is the most effective setting.

Increasing Phase: Target Size. After preliminary experimentation with a target size determined by multiplication with a factor, we instead decided to set the target size n_{target} of the increasing phase of our algorithm with an absolute offset value a (i.e., $n_{\text{target}} = n + a$). In other words, the increasing phase ends when the size of the graph is at least a greater than at the start of the increasing phase.

As can be seen in Table 5.2, the average size of the remaining kernels has been the smallest when we increase by $a = 85$ vertices, in experiments using our tuning dataset.

5.2.3 Rejecting Underperforming Backward Reduction Rules

Following our parameter tuning efforts for improving the effectiveness of individual expansion rules, we now compare their effectiveness against each other.

Not all of the 15 expansion rules are ultimately advantageous to have in the algorithm. In early experiments, it became clear that the algorithm performs better when some backward reduction rules are disabled.

Struction Maximum Increase Comparison	
$n_{G'} - n_G$ range	mean $\lvert K\rvert/\lvert K_{\text{Baseline}}\rvert$
EXTENDED STRUCTION	
$[0,1]$	100.0 % (Baseline)
$[0,2]$	98.93%
$[0,3]$	98.72 %
$[0,4]$	**98.52** %
$[0,5]$	99.46 %
$[0,6]$	100.39 %
EXTENDED REDUCED STRUCTION	
$[0,0]$	100.96 %
$[0,1]$	**100.0** % (Baseline)
$[0,2]$	100.74 %
$[0,3]$	102.39 %
$[0,4]$	104.16 %
$[0,5]$	106.32 %

Table 5.1: Arithmetic mean of the remaining number of vertices in a graph after running the algorithm for 1000 seconds, with a permitted increase in number of vertices in the respective range. A lower percentage value means the setting is better.

Increasing Phase: Target Increase of n	
n target increase	mean $\lvert K\rvert/\lvert K_{\text{Baseline}}\rvert$
$a = 50$	100.0 % (Baseline)
$a = 60$	99.46 %
$a = 65$	99.50 %
$a = 70$	99.17 %
$a = 75$	98.33 %
$a = 80$	98.33 %
$a = 85$	**97.04** %
$a = 90$	97.79 %
$a = 100$	97.85 %

Table 5.2: Arithmetic mean of the remaining graph size after running the algorithm for 1000 seconds, with the increasing phase increasing the graph size n by at least a vertices.

To measure the impact of individual expansion rules, we test 15 versions of our algorithm, each disabling a single expansion rule, as well as a 16th baseline version with all rules enabled. The results from this experiment are given in Table 5.3. As can be seen towards the bottom of the table, there are several expansion rules where the algorithm performs better when the respective rule is omitted from the algorithm. From this, we learn which rules are expendable.

Exclusion of a Single Expansion Rule	
Omitted Expansion Rule	Avg. remaining size
Nondecreasing Extended Struction	**105.33** %
Nondecreasing Extended Reduced Struction	100.95 %
Backward V-Shape Mid 1	100.54 %
Backward V-Shape Mid 2	100.51 %
Backward Degree-1 Fold	100.23 %
Backward 4-Cycle	100.12 %
Backward Domination	100.06 %
Backward Triangle Min	99.99 %
Backward Simplicial Weight Transfer	99.97 %
Backward Triangle Mid	99.94 %
Backward 5-Cycle	99.82 %
Backward 4-Path	99.55 %
Backward V-Shape Max	99.54 %
Backward Basic Single-Edge	98.88 %
Backward 3-Path	98.5 %
Baseline (none omitted)	100.0 %

Table 5.3: Comparison of variants which disable individual expansion rules. The performance of each variant, measured in the arithmetic mean size of the remaining kernels, is compared to the baseline, which has all 15 rules enabled. The experiment was repeated with five different seeds. A lower reported value means an expansion rule is worse, as it means the algorithm performs better without this rule.

5.3 Comparison with State-of-the-Art Solvers

Finally, after completing our tuning experiments and adjusting the algorithm accordingly by setting parameters and excluding underperforming expansion rules, we apply our finalized algorithm to the evaluation dataset of 207 MWIS problem instances. We try to determine if our algorithm successfully makes instances easier to solve by examining the result of state-of-the-art solvers on unprocessed and pre-processed instances.

Evaluation Dataset. For our evaluation dataset, we use the same instances used by Gellner et al. [7] and Großmann et al. [8]. The set, 207 instances in total, contains graphs in multiple different categories.

The graphs stem from five different sources: 34 instances are social networks from the Stanford Large Network Dataset Repository (SNAP) [14], 148 instances are real-world conflict graphs derived from OpenStreetMap (OSM) [16], 6 instances with weights related to population data are taken from the SuiteSparse Matrix Collection (SSMC) [4], 14 instances are dual graphs of triangle meshes (MESH) [17] and finally, 5 instances are 3D meshes that were derived from simulations using the finite element method (FE) [18].

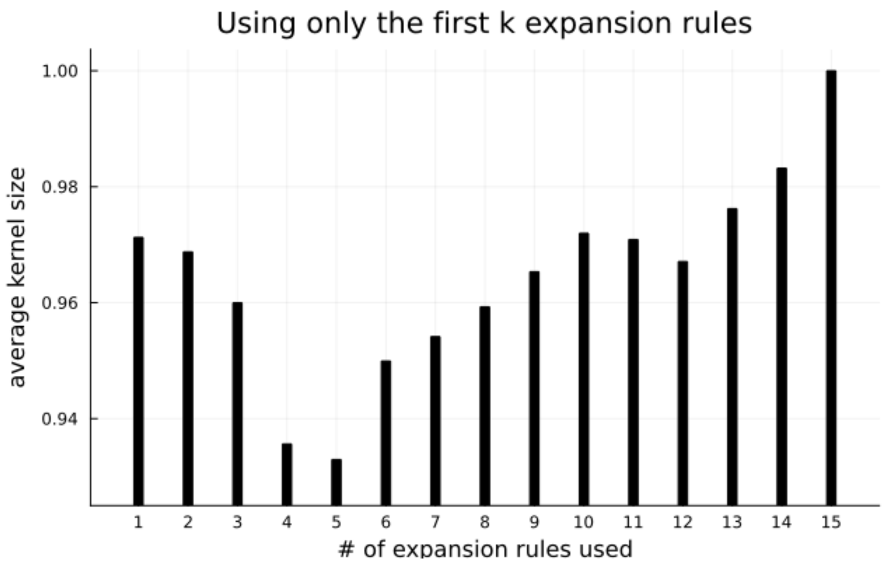


Figure 5.2: Plot depicting the average remaining graph size after applying our algorithm using the first k rules, using the order shown in Table 5.3. An alternative presentation of the data is given in Table 5.4.

Using only the first k Expansion Rules		
Selected rules	Avg. rem. n	Rank
Only Nondecr. Ext. Struction	97.13 %	11
First 2 (+ N. Ext. Redu. Struction)	96.88 %	9
First 3 (+ B. V-Shape Mid 1)	96.0 %	6
First 4 (+ B. V-Shape Mid 2)	93.56 %	2
First 5 (+ B. Degree-1 Fold)	**93.29** %	**1**
First 6 (+ B. 4-Cycle)	94.99 %	3
First 7 (+ B. Domination)	95.41 %	4
First 8 (+ B. Triangle Min)	95.93 %	5
First 9 (+ B. Simpl. Weight Transf.)	96.54 %	7
First 10 (+ B. Triangle Mid)	97.2 %	12
First 11 (+ B. 5-Cycle)	97.09 %	10
First 12 (+ B. 4-Path)	96.71 %	8
First 13 (+ B. V-Shape Max)	97.62 %	13
First 14 (+ B. Basic Single-Edge)	98.32 %	14
All 15 (+ B. 3-Path)	100.0 %	15

Table 5.4: Using the order of expansion rules from Table 5.3, these are the results after applying a version of our algorithm using the first k expansion rules to the parameter tuning dataset for 1000 seconds. The percentage given is the average size of the remaining kernel after termination of the algorithm, compared to the worst performing variant which uses all 15 expansion rules. A lower value means this set of expansion rules is better.

Some details about the instances included in this dataset can be found in Appendix C.

Comparison Setup. To evaluate its performance, we treat our algorithm as a pre-processing step of an instance G before attempting to solve the returned reduced graph instance G' with one of several state-of-the-art MWIS solvers. We report the weight $\alpha_\omega(G)$ of an optimal solution $MWIS(G)$, if found, or otherwise the weight of the heaviest independent set in G that can be found within the time limit.

As the time limit for computing a solution on each instance, we set 4 500 seconds (75 minutes). To be able to compare our work with a state-of-the-art solver, we first apply this solver to the original instance G with the full 4 500 seconds as the time limit.

For the pre-processing step, we execute our algorithm on the original instance G with a time limit of 1 500 seconds. Unless an optimal solution is found, this step does not end early. Then, we also apply the solver to the resulting reduced instance G' with its time limit set to the remaining time ($\lessapprox$ 3 000 seconds).

Examining the results from these different applications of the solver, we can evaluate if it is worthwhile to spend part of the time budget on our pre-processing algorithm.

Two Configurations. As described in Section 5.2.3, we found that using a small subset of only 5 expansion rules, three of which are our own contribution, produced the best results on the smaller parameter tuning dataset. While we expect this configuration to perform the best on the evaluation dataset as well, we do not want to fully reject all other presented backward rules yet. Therefore, we additionally compare against a configuration using three more expansion rules in this evaluation. The two configurations are denoted as the $\#R5$ and the $\#R8$ configurations, respectively, for their number of expansion rules.

In the increasing phase of our algorithm in the $\#R5$ configuration, the expansion rules NONDECREASING EXTENDED STRUCTION, NONDECREASING EXTENDED REDUCED STRUCTION, BACKWARD V-SHAPE MID 1, BACKWARD V-SHAPE MID 2 and BACKWARD DEGREE-1 FOLD are used.

In the $\#R8$ configuration, we make use of the BACKWARD 4-CYCLE, BACKWARD DOMINATION and BACKWARD TRIANGLE MIN expansion rules in addition to the five used in the $\#R5$ configuration. These are the three next best performing backward rules according to our previous experiments (cf. Table 5.3).

5.3.1 Intermediary Results after Pre-Processing

Our algorithm is restricted to only perform data reductions, and does not include a *branch-and-bound* component. Therefore, when an instance is *solved* by our pre-processing algorithm, that means the number of vertices in the kernel became zero exclusively through the application of reduction rules (of both directions).

Simple Instances. Out of the 207 instances in the evaluation dataset, 178 could be fully solved within the allotted 1 500 seconds by our pre-processing algorithm alone. Out of those, 141 were simple enough that they could already be solved in the initial decreasing phase of our algorithm, i.e., they did not require any cycles of our cyclic algorithm. Since the $\#R5$ and $\#R8$ configurations only differ in the increasing phase of our cyclic algorithm, these 141 instances are not meaningful when comparing the two configurations.

Solved by Cyclic Iteration. For the 34 instances which were solved during pre-processing, but *not* already during the initial decreasing phase (and instead after multiple cycles of our cyclic algorithm), we can compare the two configurations: The mean time taken to solve these instances was 79.9 seconds for $\#R5$ and 83.3 seconds for $\#R8$, meaning the configuration with fewer expansion rules calculated the optimal solution slightly faster. The solution was found after a mean number of cycles of only 317 in the $\#R5$ configuration, whereas the $\#R8$ configuration required an average of 408 cycles to arrive at the same solution. This indicates that adding more expansion rules beyond the best five hinders the chance of improvement in each individual cycle slightly.

Complex Instances. Conversely, there are 29 instances which were not solved during pre-processing. Five of these are so complex that the initial decreasing phase consumed all of the allotted 1 500 seconds, and no increasing phase was performed. These five are `osm_district-of-columbia-AM3`, `osm_hawaii-AM3`, `osm_kentucky-AM3`, `osm_rhode-island-AM3` and `snap_soc-pokec-relationships-uniform`. Again, comparison between the two configurations $\#R5$ and $\#R8$ is not meaningful on these five instances.

Unsolved after Cyclic Iteration. The 29 instances that were *not* solved by our pre-processing are 2 out of 5 `fe` instances, 22 out of 148 `osm` instances, 4 out of 34 `snap` instances, 1 out of 6 `ssmc` instances and none of the 14 `mesh` instances.

Using the $\#R5$ configuration, the arithmetic mean size of remaining kernels was 36 123, whereas with the $\#R8$ configuration, there were $4\ \%$ more vertices remaining with a mean of 37 553 vertices.

5.3.2 Experiments Using Pre-Processed Instances

In the following experiments, several state-of-the-art solvers for the MWIS problem were used to find solutions for the remaining kernels after pre-processing.

If spending some time on our pre-processing routine makes it possible to subsequently find a better solution on the kernel in the remaining time, compared to the solution obtained from spending the entire time budget on only using the solver, then this demonstrates some effectiveness of our algorithm.

Comparison with KaMIS Branch-and-Reduce

First, we perform a comparison using the `weighted_branch_reduce` routine from the KaMIS project [13].

Table 5.5 shows the results of this experiment. The left section contains results from using the branch-and-reduce solver alone with a time limit of 4500 seconds; in the center and right sections, the instance was pre-processed using $\#R5$ or $\#R8$ respectively for 1 500 seconds first. Only instances where the best calculated weight differs between the three different strategies are displayed. Rows where the best found weight is optimal have a gray background.

Naturally, the majority of calculated independent set weights are optimal in all categories. The KaMIS branch-and-reduce routine on its own manages to fully solve 176 of the 207 instances in 4 500 seconds. By combining the solver with our algorithm, this number is boosted to 187 instances.

Our pre-processing is particularly effective in improving the final branch-and-reduce result for the `fe`, `snap` and `ssmc` instances, as using our algorithm never results in a worse result. With the `osm` instances, which tend to have a higher density of edges, the result are more mixed. The worst result is for the `osm_rhode-island-AM3` instance, where the best found weight for an independent set after $\#R5$ is 1 492 weight units less than the result without pre-processing. However, note that this is one of the five instances where our algorithm could not perform even one full cycle of its routine within the 1 500 second time limit.

On average, the independent set weight found when one of our pre-processing configurations is applied is 4 000 and 3 920 weight units higher.

Comparison with HILS

Our next experimental comparison was performed with the HILS algorithm by Nogueira et al. [15]. As can be seen in Table 5.6, HILS on its own only solves 150 of the instances within the time limit, the lowest amount of instances solved among our experiments. However, we see great synergy when combining HILS with our pre-processing step, as we not only calculate the weights faster compared to our KaMIS branch-and-reduce experiment and to the runs of HILS alone, we also obtain on average higher weights. The number of fully solved instances rises to 189 out of 207 for both configurations, and $\#R5$ again performs slightly better than $\#R8$. A combination with $\#R5$ produced a higher weight compared to HILS alone in 54 instances, whereas HILS alone, thanks to the higher time limit, found a better weight for the instances `fe_rotor`, `osm_hawaii-AM3` and `osm_rhode-island-AM3`.

Comparison with m^{2}wis

Our next experimental comparison is with m^{2}wis, a memetic algorithm presented by Großmann et al. in [8].

graph	branch-and-reduce		#R5 + branch&reduce		#R8 + branch&reduce	
fe	t	ω	t	ω	t	ω
body	4 500.01	1 628 014	4 049.76	1 680 071	1 500.04	**1 680 182**
pwt	4 500.05	1 131 723	4 499.29	**1 166 800**	4 500.02	1 166 187
rotor	4 500.63	2 493 138	4 498.82	2 503 886	4 500.98	**2 507 775**
sphere	4 500.01	598 942	79.76	**617 816**	22.52	**617 816**
osm	t	ω	t	ω	t	ω
alabama-3	4 500.01	185 707	4 499.03	**185 729**	4 500.39	185 700
dc-2	4 500.01	196 081	4 502.09	208 036	4 499.63	**208 204**
dc-3	4 501.56	204 656	4 571.04	206 882	4 556.63	**206 908**
greenland-3	4 500.02	13 804	4 528.91	**13 827**	4 526.24	13 773
hawaii-3	4 502.86	**132 801**	4 396.44	**132 801**	4 405.87	**132 801**
idaho-3	4 500.08	**77 010**	4 519.35	**77 010**	4 510.9	**77 010**
kansas-3	4 500.12	87 923	4 497.85	**87 959**	4 499.06	87 949
kentucky-3	4 505.42	**100 311**	4 009.04	100 310	4 152.21	100 310
massachusetts-3	4 502.24	145 652	4 499.61	**145 855**	4 500.54	145 754
north-carolina-3	4 500.01	49 563	4 502.18	**49 720**	4 500.58	49 492
oregon-3	4 501.34	**174 966**	4 528.02	174 842	4 513.05	174 312
rhode-island-3	4 500.1	**196 062**	4 881.92	194 570	4 764.98	195 187
vermont-3	4 502.36	**62 687**	4 497.68	62 685	4 506.31	**62 687**
virginia-3	4 501.15	**308 242**	4 497.08	307 421	4 499.95	308 215
washington-3	4 500.02	308 453	4 493.52	308 461	4 515.71	**309 231**
snap	t	ω	t	ω	t	ω
as-skitter-u.	4 500.6	124 146 100	4 498.11	**124 154 165**	4 501.39	**124 154 165**
loc-gowalla_e.	4 500.07	12 276 798	1 572.31	**12 276 929**	1 599.71	**12 276 929**
soc-LiveJ.1-u.	4 502.89	284 009 147	4 499.23	**284 036 176**	4 499.98	284 036 062
soc-pokec-r.-u.	4 775.08	82 769 044	4 634.27	**82 824 299**	4 504.27	82 803 152
web-BerkStan-u.	4 500.18	43 885 391	294.17	**43 907 482**	186.97	**43 907 482**
web-Stanford-u.	4 500.1	17 792 824	16.37	**17 792 930**	16.56	**17 792 930**
ssmc	t	ω	t	ω	t	ω
ca2010	4 500.23	16 571 662	4 499.96	**16 869 550**	4 499.97	**16 869 550**
fl2010	4 500.15	8 632 731	998.05	**8 743 506**	1 500.66	**8 743 506**
ga2010	4 500.08	4 639 769	37.72	**4 644 417**	30.77	**4 644 417**
il2010	4 500.14	5 846 040	526.58	**5 998 539**	415.54	**5 998 539**
nh2010	4 500.03	581 732	0.59	**588 996**	0.56	**588 996**
ri2010	4 500.01	445 978	3.32	**459 275**	0.95	**459 275**

overall	branch-and-reduce	#R5 + branch&reduce	#R8 + branch&reduce
mean time	807.2	602.84	593.48
mean weight	7 389 985	**7 393 985**	7 393 905
# optimal sol.	176/207	**187/207**	186/207

Table 5.5: Comparison with the branch-and-reduce routine of KaMIS. ω columns show the weight of the heaviest found independent set, t gives the time in seconds.

Starting with this experiment, we unfortunately ran into a problem with our experimental setup, where certain instances with a large number of edges could not be computed in memory, as our computing machine is only equipped with 100 GB of main memory. In the experiment with m^{2}wis and its variant, m^{2}wis + s, this only affected the `osm_kentucky-AM3` instance. Failed instances are listed in the tables, but rows where one experiment failed are not counted for the arithmetic means reported at the bottom of each table.

As can be seen in Table 5.7, m^{2}wis alone outperforms the configurations using our algorithm in terms of speed. However, the quality of solutions found within 4500 seconds is again higher when using our pre-processing. Using the $\#R5$ configuration, we find independent sets with, on average, 1412 more weight units compared to only using m^{2}wis. With $\#R8$ we also see higher weight independent sets being found, but only by 537 units in the arithmetic mean.

Comparison with CyclicFast

Another experiment we performed was for comparison with the CyclicFast algorithm, presented by Gellner et al. in [7]. This algorithm also makes use of cyclic reduction and expansion phases to shrink a graph, embedded in a branch-and-reduce framework. It makes use of the EXTENDED STRUCTION reductions [7], which we also use in a similar way.

Unfortunately, for 14 of the 207 instances in the evaluation dataset, we were unable to compute the results for this algorithm, making the comparison more difficult.

Still, as we can see in Table 5.8 in the results for the remaining 193 instances, CyclicFast was vastly faster in computing its solutions with an arithmetic mean computation time of just 4.45 seconds. This is compared to the mean of 146 and 151 seconds for the computation including our algorithm. This is largely because our algorithm never terminates before its time limit of 1 500 seconds, unless the graph gets shrunk to zero vertices.

In these same 193 instances, the experiments using our pre-processing algorithm found independent sets with, on average, 4 577 and 4 486 more weight units per graph for $\#R5$ and $\#R8$, respectively.

Comparison with Combination of m^{2}wis and CyclicFast

Our last comparison is with the solver m^{2}wis + s, an algorithm combining regular m^{2}wis with an initial reduction using CyclicFast [8].

This comparison shows the smallest difference in performance, as can be seen in Table 5.9. With or without our pre-processing, the exact same set of instances get solved optimally. Our pre-processing again raises the mean time to return a solution. In terms of weight, we obtain a slightly higher arithmetic mean of 1 121 more units across all instances with $\#R5$, and almost no increase in average weight with just 51 more units when using $\#R8$.

5.3.3 Comparison Disabling our Contributions

The focus of our work was on developing the new backward reduction rules. To make use of them, we included them in a cyclic reduction algorithm. Various design decisions were made for this cyclic algorithm, including the selection of implemented forward reduction rules and the randomized order of applying them. Our comparisons in the preceding sections indicate that our algorithm as a whole is worthwhile to apply as a preprocessing step before other MWIS solvers.

In this section, we also want to demonstrate that the effectiveness of our algorithm stems from the newly developed backward reduction rules, in particular the three included in the $\#R5$ configuration, and not just from other incidental decisions that were made in the implementation of our algorithm. To this end, we compare the preprocessing performance of our $\#R5$ configuration to a version of our algorithm with our main contributions disabled. We call this configuration $\#R2$, because it uses only the two expansion rules Non-Decreasing Extended Struction and Nondecreasing Extended Reduced Struction in the increasing phase, which are previously known expansion rules. Additionally, the Optional Edge Deletion reductions in the decreasing phase are disabled in $\#R2$.

Using both configurations for preprocessing of the same 207 instances as before, again using a time limit of 1 500 seconds, $\#R2$ only manages to fully reduce 166 instances, compared to the 183 of $\#R5$. The arithmetic mean size of kernels obtained using $\#R2$ is 4 891.9, compared to 4 831.1 using $\#R5$.

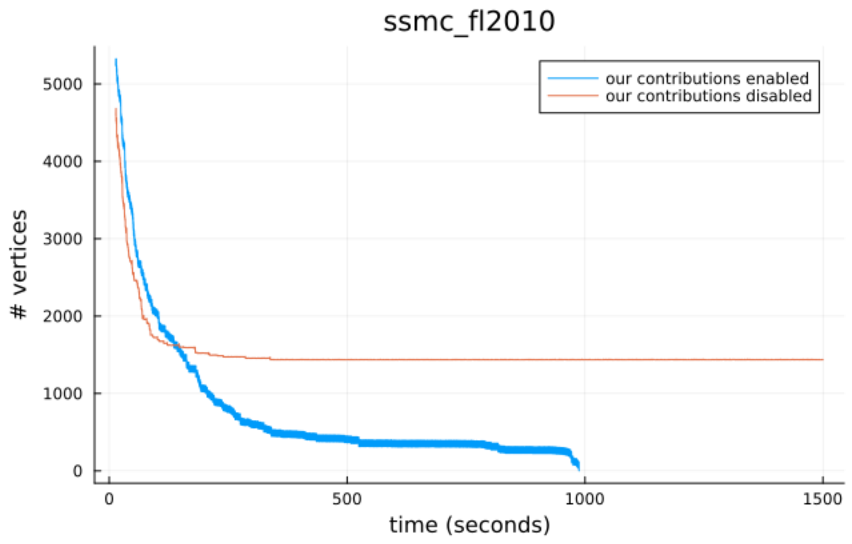


Figure 5.3: The configuration not using new backward reduction rules initially manages to shrink the graph quickly, finding a kernel with 1 435 vertices after 339 seconds, but is unable to make another breakthrough afterwards. When also using the three best new backward reduction rules, fully solving the graph in under 1 000 seconds is possible.

We can see this demonstrated in Figure 5.3. The $\#R5$ configuration initially finds improvements slightly slower compared to $\#R2$, but $\#R5$ has more options to expand the graph which allow the algorithm to explore more reduction paths and ultimately, to fully solve the instance.

5.4 Discussion

Comparison between Configurations. As seen in the previous sections, our $\#R5$ configuration, using only the two NONDECREASING EXTENDED STRUCTION rules, the two BACKWARD V-SHAPE MID rules, and BACKWARD DEGREE-1 FOLD outperformed the $\#R8$ configuration with three more expansion rules in every comparison, as higher weight solutions could be found. At the same time, the time taken to compute these solutions was similar in every case.

This confirms our findings from the experiments on the smaller parameter tuning dataset, and indicates that only a small number of backward reduction rules are effective and efficient on a level comparable to the previously known EXTENDED STRUCTION-based rules.

Improvements in Solution Quality. In our experimental evaluation, we found that solution quality was rarely worse when time was spent on our pre-processing, specifically using the $\#R5$ configuration. The cases where pre-processing using our algorithm is clearly not worth it are mainly those where the graph instance is so complex that the pre-processing algorithm cannot reach the cyclic phases beyond the initial reduction within the allowed time limit. Overall, our algorithm was shown to lead to higher independent set weights very consistently.

In fact, if we consider the arithmetic mean of the weight values computed across the dataset, the variants using pre-processing by our $\#R5$ algorithm resulted in the highest solution quality in every single comparison.

Particularly Complex Instances and the Time Limit. As previously stated in Section 5.3.1, our pre-processing algorithm was unable to complete a full cycle consisting of an increasing and a decreasing phase after initial reduction, or was unable to complete the initial reduction, in the case of five graph instances. It is possible that our algorithm would have shown an effect if the time limit was long enough to reach the iterative phases, but in our experiments, these instances generally showed the weakest results. For backward rules to have an effect on these instances within a short time limit like the one we set, they would have to be intermixed with forward reduction rules in the decreasing phase, which is an idea we did not explore in this thesis.

Time Efficiency. In terms of time taken to compute a solution, our pre-processing improved the time taken compared to KaMIS branch-and-reduce and to HILS, but the combination of pre-processing and application of the solver took longer compared to the computations of m^{2}wis, m^{2}wis + s and CyclicFast. This is partially because our algorithm keeps performing iterations of the cyclic routine until the time limit is met, unless it manages to reduce the graph down to zero vertices and thereby solve it.

For an example of where instead terminating early would likely be faster, consider the instance `ssmc_ca2010`. Over 1 500 seconds, our algorithm continually shrinks the graph down, with only 710 vertices remaining after the time limit. This kernel can then be quickly solve optimally by one of the state-of-the-art solvers, for example after just 0.75 seconds using the m^{2}wis algorithm.

However, as can be seen in Figure 5.4, the rate of improvement slows down substantially after initial great improvements. After 677 seconds, there are already less than 1 000 vertices remaining, but improvements after this time are much rarer.

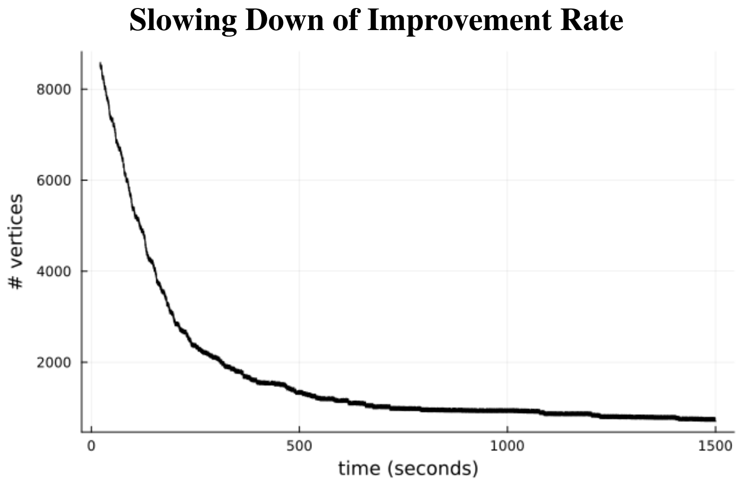


Figure 5.4: Applying our cyclic algorithm to `ssmc_ca2010`, we manage to greatly shrink the graph size in the first 200 seconds using decreasing and increasing reduction rules, but the rate of improvement slows down significantly afterwards.

With a method for detecting when the rate of improvement becomes too slow, it would be possible to stop the cyclic algorithm early and continue with a different solver, which would likely improve the efficiency of the computation.

graph	HILS		#R5 + HILS		#R8 + HILS	
fe	t	ω	t	ω	t	ω
body	2 072.64	1 678 286	1 051.2	**1 680 182**	1 507.45	**1 680 182**
ocean	4 500.01	7 038 618	8.51	**7 248 581**	8.14	**7 248 581**
pwt	1 098.66	1 174 694	1 850.36	**1 178 087**	1 862.23	1 177 979
rotor	4 500.0	**2 652 075**	4 499.61	2 650 137	4 500.0	2 651 218
sphere	311.99	616 516	80.11	**617 816**	22.95	**617 816**
mesh	t	ω	t	ω	t	ω
blob	286.22	854 822	0.49	**855 547**	0.48	**855 547**
buddha	4 500.06	56 098 924	12.49	**57 555 880**	15.56	**57 555 880**
bunny	2 643.6	3 680 724	0.8	**3 686 960**	0.75	**3 686 960**
cow	63.72	269 452	0.45	**269 543**	0.37	**269 543**
dragon	4 500.0	7 946 511	1.15	**7 956 530**	1.06	**7 956 530**
ecat	4 500.02	36 098 983	8.66	**36 650 298**	9.49	**36 650 298**
osm	t	ω	t	ω	t	ω
alabama-3	384.17	**185 744**	1 530.77	**185 744**	1 542.76	**185 744**
dc-2	1 197.05	209 127	1 828.42	**209 132**	1 813.93	**209 132**
dc-3	4 500.01	227 618	4 499.25	**227 669**	4 500.0	227 550
greenland-3	1 359.01	14 011	2 503.83	**14 012**	2 505.14	14 011
hawaii-3	4 500.0	**141 047**	4 499.92	141 033	4 500.01	141 015
kansas-3	854.7	**87 976**	2 026.44	**87 976**	1 998.57	**87 976**
kentucky-3	4 500.04	100 507	4 499.45	100 502	4 500.02	**100 510**
rhode-island-3	4 171.61	**201 771**	4 499.84	201 751	4 500.0	201 743
vermont-3	959.7	63 304	1 942.16	**63 312**	1 954.88	**63 312**
snap	t	ω	t	ω	t	ω
as-skitter-u.	4 500.14	123 241 756	1 542.06	**124 157 729**	1 542.1	**124 157 729**
ca-CondMat-u.	1 337.73	1 147 948	0.54	**1 147 950**	0.45	**1 147 950**
com-amazon	4 500.02	19 265 906	2.16	**19 271 031**	2.17	**19 271 031**
loc-gowalla_e.	4 500.01	12 274 663	1 509.18	**12 276 929**	1 509.67	**12 276 929**
roadNet-CA-u.	4 500.06	108 406 930	13.74	**111 360 828**	17.08	**111 360 828**
soc-LiveJ.1-u.	4 500.17	278 906 042	1 537.17	**284 036 239**	1 543.71	**284 036 239**
soc-pokec-r.-u.	4 500.26	82 977 745	4 499.56	**83 625 383**	4 500.05	83 530 465
web-BerkStan-u.	4 500.05	43 696 500	294.59	**43 907 482**	187.38	**43 907 482**
ssmc	t	ω	t	ω	t	ω
ca2010	4 500.04	16 644 409	1 513.38	**16 869 550**	1 516.21	**16 869 550**
fl2010	4 500.03	8 696 096	997.49	**8 743 506**	1 507.59	**8 743 506**
ga2010	4 500.04	4 640 587	38.1	**4 644 417**	31.18	**4 644 417**
overall	HILS		#R5 + HILS		#R8 + HILS	
mean time	908.22		357.88		364.18	
mean weight	7 321 920		**7 398 853**		7 398 399	
# optimal sol.	150/207		**189/207**		**189/207**	

Table 5.6: An excerpt of the experiment results comparing the performance of our algorithm with the HILS algorithm.

graph	m^2wis		#R5 + m^2wis		#R8 + m^2wis	
fe	t	ω	t	ω	t	ω
body	591.19	1 679 750	1 050.76	**1 680 182**	1 500.69	**1 680 182**
pwt	289.45	1 172 482	2 085.24	**1 177 844**	1 801.38	1 177 827
rotor	191.83	2 637 201	1 749.17	2 639 995	2 257.6	**2 641 414**
sphere	457.86	616 144	79.76	**617 816**	22.52	**617 816**
mesh	t	ω	t	ω	t	ω
dragon	6.88	7 956 510	0.74	**7 956 530**	0.7	**7 956 530**
dragonsub	403.73	32 213 118	6.49	**32 213 898**	6.73	**32 213 898**
ecat	387.01	36 649 883	8.2	**36 650 298**	9.1	**36 650 298**
turtle	135.25	14 262 961	1.95	**14 263 005**	1.87	**14 263 005**
osm	t	ω	t	ω	t	ω
dc-2	100.2	209 131	1 547.26	**209 132**	1 541.11	**209 132**
dc-3	1 818.61	227 540	2 563.8	**227 626**	2 899.51	227 605
greenland-3	258.24	**14 012**	1 713.35	14 011	1 769.43	14 011
hawaii-3	3 759.16	140 992	3 622.8	**141 004**	3 924.84	140 969
idaho-3	255.52	77 144	1 783.88	**77 145**	1 707.86	**77 145**
kentucky-3	4 241.59	97 212	-		-	
massachusetts-3	41.94	**145 866**	1 584.86	**145 866**	1 544.17	**145 866**
oregon-3	171.78	**175 078**	1 700.08	**175 078**	1 706.9	**175 078**
rhode-island-3	1 245.24	**201 771**	2 928.53	201 712	2 344.4	**201 771**
vermont-3	49.6	**63 312**	1 551.73	**63 312**	1 563.12	**63 312**
virginia-3	73.14	**308 305**	1 570.87	**308 305**	1 540.14	**308 305**
washington-3	823.2	**314 288**	1 720.66	**314 288**	1 621.26	**314 288**
snap	t	ω	t	ω	t	ω
as-skitter-u.	235.84	124 157 715	1 552.99	**124 157 729**	1 549.6	**124 157 729**
roadNet-CA-u.	14.14	111 360 162	13.34	**111 360 828**	16.69	**111 360 828**
roadNet-PA-u.	636.36	61 731 270	6.49	**61 731 589**	7.93	**61 731 589**
roadNet-TX-u.	10.81	78 599 294	9.86	**78 599 946**	11.35	**78 599 946**
soc-LiveJ.1-u.	263.16	284 036 126	1 517.52	**284 036 239**	1 515.52	**284 036 239**
soc-pokec-r.-u.	4 039.28	80 690 587	4 547.09	**80 915 286**	4 514.02	80 733 686
web-BerkStan-u.	556.08	43 906 971	294.17	**43 907 482**	186.97	**43 907 482**
ssmc	t	ω	t	ω	t	ω
ca2010	358.28	16 840 602	1 501.71	**16 869 550**	1 501.81	16 869 468
fl2010	97.06	8 737 274	997.1	**8 743 506**	1 500.63	**8 743 506**
ga2010	11.57	4 644 209	37.72	**4 644 417**	30.77	**4 644 417**
il2010	331.15	5 982 542	526.58	**5 998 539**	415.54	**5 998 539**
overall	m^2wis		#R5 + m^2wis		#R8 + m^2wis	
mean time	89.86		275.63		281.27	
mean weight	7 419 663		**7 421 075**		7 420 200	
# optimal sol.	172/207		**189/206**		188/206	

Table 5.7: Comparison with m^2wis. Due to limited memory, a heavy independent set for `osm_kentucky-AM3` could not be computed.

graph	CyclicFast		#R5 + CyclicFast		#R8 + CyclicFast	
fe	t	ω	t	ω	t	ω
body	1.63	1 680 108	1 051.65	**1 680 182**	1 500.04	**1 680 182**
ocean	13.34	6 594 370	8.1	**7 248 581**	7.77	**7 248 581**
pwt	30.33	1 112 613	1 516.85	1 178 583	1 598.75	**1 178 735**
rotor	27.37	**2 539 291**	1 529.05	2 532 233	1 524.24	2 534 475
osm	t	ω	t	ω	t	ω
dc-2	-		-		1 502.53	209 132
dc-3	-		2 305.06	149 648	2 292.35	149 648
greenland-3	-		-		-	
hawaii-3	-		2 199.71	123 884	2 193.9	123 884
idaho-3	-		2 865.49	77 145	2 885.53	77 145
kentucky-3	-		2 104.11	100 475	2 102.26	100 465
massachusetts-3	-		1 501.24	145 866	1 501.56	145 866
north-carolina-3	-		1 501.05	**49 720**	2 574.99	**49 720**
oregon-3	-		-		-	
rhode-island-3	-		-		-	
vermont-3	-		1 586.48	63 312	1 561.06	63 312
virginia-3	-		1 511.65	308 209	1 508.16	308 166
washington-3	3.4	285 549	-		-	
snap	t	ω	t	ω	t	ω
as-skitter-u.	282.67	124 137 365	1 739.65	124 138 232	1 739.36	**124 138 439**
soc-LiveJ.1-u.	-		1 507.06	284 036 216	1 505.24	284 036 216
soc-pokec-r.-u.	460.26	76 880 511	1 978.95	**77 049 923**	1 880.44	77 029 787
overall	CyclicFast		#R5 + CyclicFast		#R8 + CyclicFast	
mean time	4.45		146.16		151.53	
mean weight	6 414 004		**6 418 581**		6 418 490	
# optimal sol.	187/194		**189/203**		**189/202**	

Table 5.8: Result of the experiment with CyclicFast. Due to limited memory available in our setup, 14 instances have incomplete results. The reported mean time and weight only take such rows into account where all results are present.

graph	m^2wis + s		#R5, then m^2wis + s		#R8, then m^2wis + s	
fe	t	ω	t	ω	t	ω
body	61.17	**1 680 182**	1 050.76	**1 680 182**	1 504.46	**1 680 182**
pwt	65.04	1 178 365	1 802.01	**1 178 734**	1 681.28	1 178 567
rotor	1 775.11	2 623 267	3 985.96	**2 641 428**	3 434.06	2 641 414
osm	t	ω	t	ω	t	ω
dc-2	249.1	**209 132**	1 631.46	**209 132**	1 569.98	**209 132**
dc-3	3 089.39	**227 645**	4 079.54	227 599	2 140.1	209 719
greenland-3	2 000.76	**14 012**	1 678.85	14 011	1 626.98	14 011
hawaii-3	4 610.74	**140 979**	3 015.61	130 191	2 998.27	130 191
idaho-3	428.62	77 144	1 925.6	**77 145**	2 927.39	**77 145**
kentucky-3	4 239.33	97 778	-		-	
massachusetts-3	125.0	**145 866**	1 623.02	**145 866**	1 623.33	**145 866**
oregon-3	478.97	**175 078**	1 851.34	**175 078**	2 843.17	**175 078**
rhode-island-3	2 500.92	**201 771**	4 058.66	201 712	3 777.84	201 758
vermont-3	1 308.92	**63 312**	2 751.75	**63 312**	3 242.23	**63 312**
virginia-3	1 455.0	**308 305**	1 696.78	**308 305**	1 633.51	**308 305**
washington-3	1 545.75	**314 288**	2 924.94	**314 288**	2 822.01	**314 288**
snap	t	ω	t	ω	t	ω
as-skitter-u.	2 442.79	124 157 715	2 691.91	**124 157 729**	2 934.24	**124 157 729**
soc-LiveJ.1-u.	805.21	**284 036 239**	2 113.32	**284 036 239**	1 690.36	**284 036 239**
soc-pokec-r.-u.	4 586.55	80 695 686	6 468.52	**80 919 005**	5 608.25	80 716 632
overall	m^2wis + s		#R5, then m^2wis + s		#R8, then m^2wis + s	
mean time	136.05		328.26		324.77	
mean weight	7 419 931		**7 421 052**		7 419 982	
# optimal sol.	**189/207**		**189/206**		**189/206**	

Table 5.9: Experimental results from comparison with m^2wis + s. For all 189 instances where the optimal weight is known, it was computed by all three variants.

CHAPTER 6

Conclusion

6.1 Conclusion

This thesis set out to explore the idea of applying backward reduction rules for the MWIS problem to graph instances, in order to increase their size. Ideally, this would allow a wider range of forward reduction rules to be applicable to the expanded instance, which would then ultimately make it possible to reduce the graph instance down to an even smaller size than before.

To this end, we developed 14 backward reduction rules which reverse the effects of previously known forward reductions. Additionally, we presented another rule, BACKWARD V-SHAPE MID 2, which exploits a previously unexplored graph reduction which allows us to remove an edge from the graph in certain conditions.

We then integrated these reduction rules into a cyclic algorithm with randomized increasing and decreasing phases. Through experiments, we adjusted their parameters to improve their effectiveness. Then we filtered the set of backward reduction rules to identify which rules resulted in the highest solution quality. We discovered that using a small selection of backward rules, the two BACKWARD V-SHAPE MID reductions and BACKWARD DEGREE-1, increased the effectiveness of our routine the most. Accordingly, we limited the backward reduction rules that are used to this selection and finalized the algorithm.

Using this algorithm as a pre-processing step, we compared the results of applying various state-of-the-art solvers on both pre-processed and unmodified graph instances in a large dataset. Through this experimental comparison, we were able to successfully demonstrate that our pre-processing algorithm enables us to find, on average, higher quality solutions compared to those that can be found without our pre-processing.

In this thesis, we were able to show that most of the presented backward reduction rules cannot compete with the previously known NON-DECREASING EXTENDED STRUCTION and NON-DECREASING EXTENDED REDUCED STRUCTION in terms of effectiveness.

On the other hand, we were able to show that a cyclic algorithm using these two

STRUCTION-based reductions can be improved by a small number of backward reduction rules, primarily the two variants of BACKWARD V-SHAPE MID. We can recommend utilizing these backward rules in future work related to increasing transformations for the MWIS problem.

6.2 Future Work

Integration into Branch-and-Reduce Algorithm. The algorithm we developed is limited to shrinking instances in a preprocessing step. In practice, many graph instances cannot be solved optimally through reduction rules alone. The standard approach for solving the MWIS problem is to integrate reduction techniques, as presented here, into a *branch-and-reduce* algorithm [7]. A next step could be to more tightly integrate backward reduction rules in such a branch-and-reduce context. How a cyclic increase-decrease routine would best work in the middle of a branch-and-reduce algorithm is an interesting open question for future work.

Relax Separation of Phases. A weakness we encountered in our experiments was that the cyclic reduction routine becomes useless if the initial decreasing phase takes too long, since we do not even reach the cyclic recursion within the time limit. The decreasing and increasing phases are completely separate in our algorithm. One might consider applying a limited number of some expansion rules like BACKWARD V-SHAPE MID 2 during what would normally be the decreasing phase, before complex and time-consuming rules like CWIS and EXTENDED STRUCTION. It is possible that this way, graphs could be shrunk further using only simple and cheap reduction rules, before the more expensive reduction rules are applied. It remains a question for future work if this would be a viable strategy.

APPENDIX A

Implementation Details

Notes on Data Structure

Adjacency between vertices is stored through adjacency lists[1]. In the adjacency list of vertex v, every $u \in N(v)$ will appear in unsorted order. This means that generally, checking if two vertices $u, v \in V$ are adjacent takes $\mathcal{O}(\min(d(u), d(v)))$ time.

Caching the Neighborhood Weight. For a vertex $v \in V$, we frequently want to know the weight of its neighborhood, $\omega(N(v))$. This value is directly used for some reductions like NEIGHBORHOOD REMOVAL or EXTENDED SINGLE-EDGE, or while computing other values in reductions like TWO-VERTEX. We decided to store a variable containing $N(v)$ for every v, updating it whenever relevant vertices get added, deleted, or change their weight.

Timestamps. To avoid duplicate work during the decreasing phase, we store a *time-stamp* value t_v for each $v \in V$, which represents the last time v or $N(v)$ has been changed (i.e., , the last time $\omega(v)$ was modified, an edge including v was added or removed, or the weight of a neighbor $x \in N(v)$ was modified). For each forward reduction, we also store the timestamp for the most recent time it was checked if the reduction is applicable. These timestamp values are then used to quickly rule out vertices for specific reductions, instead of repeatedly checking the same conditions for a part of the graph that has not changed since the last time these checks were made.

[1]Using `std::vector` in C++.

APPENDIX B

Parameter Tuning Dataset

Parameter Tuning Dataset			
Filename	$\|V\|$	$\|E\|$	gen. distribution
delaunay_n11	2 048	6 127	$B(500, 0.5)$
delaunay_n12	4 096	12 264	$B(500, 0.5)$
delaunay_n13	8 192	24 547	$B(500, 0.5)$
delaunay_n13	8 192	24 547	$B(25, 0.5)$
delaunay_n14	16 384	49 122	$B(25, 0.5)$
delaunay_n15	32 768	98 274	$B(25, 0.5)$
delaunay_n16	65 536	196 575	$B(25, 0.5)$
delaunay_n17	131 072	393 176	$B(25, 0.5)$
belgium.osm	1 441 295	1 549 970	$B(500, 0.5)$
netherlands.osm	2 216 688	2 441 238	$B(25, 0.5)$
G3_circuit	1 585 478	3 037 674	$\mathcal{U}\{1, 1000\}$
rgg_n_2_15	32 768	160 240	$B(25, 0.5)$
rgg_n_2_16	65 536	342 127	$B(25, 0.5)$
rgg_n_2_17	131 072	728 753	$B(25, 0.5)$
rgg_n_2_18	262 144	1 547 283	$B(500, 0.5)$
caidaRouterLevel	192 244	609 066	$B(25, 0.5)$
caidaRouterLevel	192 244	609 066	$B(500, 0.5)$
caidaRouterLevel	192 244	609 066	$\mathcal{U}\{1, 50\}$
caidaRouterLevel	192 244	609 066	$\mathcal{U}\{1, 1000\}$
NACA0015	1 039 183	3 114 818	$\mathcal{U}\{1, 50\}$

Table B.1: The graphs that were picked for the tuning dataset, alongside the method(s) with which their weights were generated. Weights generated from a discrete uniform distribution in the interval $[a, b]$ are indicated with $\mathcal{U}\{a, b\}$. Weights generated from the binomial distribution with n trials and success probability 0.5 are indicated with $B(n, 0.5)$.

APPENDIX C

Evaluation Dataset

The following table contains information about the 207 graphs that were used for the experimental evaluation in Section 5.3. The table lists $|V|$ and $|E|$ (i.e., the sizes of the vertex set and the edge set), as well as the maximum degree $d_{\max}$ in each graph.

Additionally, the last column lists the weight of a *MWIS* for this graph, denoted by α_ω. For 18 of the 207 graphs, we were unable to determine this value. This set of particularly complex graphs consists of 3 `fe` instances, 12 `osm` instances, and 3 `snap` instances.

Evaluation Dataset Summary				
Graph	$\|V\|$	$\|E\|$	$d_{\max}$	α_ω
fe_body-uniform	45 087	163 734	28	-
fe_ocean-uniform	143 437	409 593	6	7 248 581
fe_pwt-uniform	36 519	144 794	15	-
fe_rotor-uniform	99 617	662 431	125	-
fe_sphere-uniform	16 386	49 152	6	617 816
mesh_blob-uniform	16 068	24 102	3	855 547
mesh_buddha-uniform	1 087 716	1 631 574	3	57 555 880
mesh_bunny-uniform	68 790	103 017	3	3 686 960
mesh_cow-uniform	5 036	7 366	3	269 543
mesh_dragon-uniform	150 000	225 000	3	7 956 530
mesh_dragonsub-uniform	600 000	900 000	3	32 213 898
mesh_ecat-uniform	684 496	1 026 744	3	36 650 298
mesh_face-uniform	22 871	34 054	3	1 219 418
mesh_fandisk-uniform	8 634	12 818	3	463 288
mesh_feline-uniform	41 262	61 893	3	2 207 219
mesh_gameguy-uniform	42 623	63 850	3	2 325 878
mesh_gargoyle-uniform	20 000	30 000	3	1 059 559
mesh_turtle-uniform	267 534	401 178	3	14 263 005

Continuation of Table C.1				
Graph	$\lvert V\rvert$	$\lvert E\rvert$	$d_{\max}$	α_ω
mesh_venus-uniform	5 672	8 508	3	305 749
osm_alabama-AM1	320	581	19	167 588
osm_alabama-AM2	1 164	19 386	126	174 309
osm_alabama-AM3	3 504	309 664	643	185 744
osm_alaska-AM1	31	31	4	20 266
osm_alaska-AM2	54	156	15	20 900
osm_alaska-AM3	86	475	29	22 325
osm_arkansas-AM1	26	19	3	17 702
osm_arkansas-AM2	55	233	21	20 771
osm_arkansas-AM3	103	1 376	59	20 935
osm_california-AM1	77	130	13	46 537
osm_california-AM2	231	3 074	68	47 153
osm_california-AM3	587	27 536	253	49 365
osm_canada-AM1	189	240	8	78 466
osm_canada-AM2	449	2 947	44	81 799
osm_canada-AM3	943	20 241	146	86 018
osm_colorado-AM1	128	232	10	50 507
osm_colorado-AM2	283	2 026	36	52 172
osm_colorado-AM3	538	8 365	92	54 741
osm_connecticut-AM1	87	96	6	55 131
osm_connecticut-AM2	211	975	23	56 058
osm_connecticut-AM3	367	3 769	62	57 650
osm_delaware-AM1	2	1	1	1 060
osm_delaware-AM2	3	3	2	1 060
osm_delaware-AM3	5	9	4	1 060
osm_district-of-columbia-AM1	2 500	24 651	137	196 475
osm_district-of-columbia-AM2	13 597	1 609 795	1 126	-
osm_district-of-columbia-AM3	46 221	27 729 137	5 940	-
osm_florida-AM1	475	1 277	19	225 655
osm_florida-AM2	1 254	16 936	106	230 595
osm_florida-AM3	2 985	154 043	356	237 333
osm_georgia-AM1	294	434	12	205 068
osm_georgia-AM2	746	7 753	95	216 346
osm_georgia-AM3	1 680	74 126	384	222 652
osm_greenland-AM1	77	341	30	9 328
osm_greenland-AM2	686	50 218	358	10 718
osm_greenland-AM3	4 986	3 652 361	3 354	-
osm_hawaii-AM1	411	1 423	30	113 792
osm_hawaii-AM2	2 875	265 158	584	125 284
osm_hawaii-AM3	28 006	49 444 921	10 313	-

Continuation of Table C.1				
Graph	$\lvert V\rvert$	$\lvert E\rvert$	$d_{\max}$	α_ω
osm_idaho-AM1	136	208	15	70 623
osm_idaho-AM2	552	35 221	296	73 554
osm_idaho-AM3	4 064	3 924 080	3 332	-
osm_illinois-AM1	113	202	15	54 678
osm_illinois-AM2	261	2 138	49	55 496
osm_indiana-AM1	2	1	1	1 146
osm_indiana-AM2	2	1	1	1 146
osm_indiana-AM3	4	6	3	1 146
osm_iowa-AM1	90	164	12	47 907
osm_iowa-AM2	155	954	38	47 984
osm_kansas-AM1	190	400	19	84 449
osm_kansas-AM2	602	16 474	197	85 942
osm_kansas-AM3	2 732	806 912	1 404	87 976
osm_kentucky-AM1	381	2 402	49	91 897
osm_kentucky-AM2	2 453	643 428	1 222	97 397
osm_kentucky-AM3	19 095	59 533 630	14 928	-
osm_louisiana-AM1	157	181	6	51 446
osm_louisiana-AM2	436	3 111	46	55 127
osm_louisiana-AM3	1 162	37 077	224	60 024
osm_maine-AM1	38	29	3	24 921
osm_maine-AM2	81	243	14	26 208
osm_maine-AM3	143	850	26	26 734
osm_maryland-AM1	104	216	18	43 930
osm_maryland-AM2	316	4 715	89	45 300
osm_maryland-AM3	1 018	95 415	476	45 496
osm_massachusetts-AM1	413	1 089	33	136 695
osm_massachusetts-AM2	1 339	35 449	236	140 095
osm_massachusetts-AM3	3 703	551 491	1 188	-
osm_mexico-AM1	175	358	17	90 599
osm_mexico-AM2	516	9 411	95	94 834
osm_mexico-AM3	1 096	47 131	273	97 663
osm_michigan-AM1	133	112	6	51 076
osm_michigan-AM2	241	750	21	51 928
osm_michigan-AM3	376	2 459	45	52 674
osm_minnesota-AM1	86	136	10	28 692
osm_minnesota-AM2	253	2 580	58	30 251
osm_minnesota-AM3	683	34 188	274	32 787
osm_mississippi-AM1	74	60	4	32 273
osm_mississippi-AM2	151	366	16	33 187
osm_mississippi-AM3	242	1 116	23	33 318

Continuation of Table C.1				
Graph	$\|V\|$	$\|E\|$	$d_{\max}$	α_{ω}
osm_missouri-AM1	10	6	2	7 928
osm_missouri-AM2	13	12	3	7 928
osm_missouri-AM3	17	24	5	7 928
osm_montana-AM1	109	194	12	55 348
osm_montana-AM2	307	5 154	108	56 068
osm_montana-AM3	837	69 293	418	59 822
osm_nebraska-AM1	40	46	7	24 345
osm_nebraska-AM2	93	734	42	26 680
osm_nebraska-AM3	145	2 168	74	27 214
osm_nevada-AM1	89	93	7	45 761
osm_nevada-AM2	242	1 531	40	47 068
osm_nevada-AM3	569	15 016	166	52 036
osm_new-hampshire-AM1	195	302	10	108 186
osm_new-hampshire-AM2	514	3 369	39	110 621
osm_new-hampshire-AM3	1 107	18 021	95	116 060
osm_new-jersey-AM1	4	6	3	256
osm_new-jersey-AM2	4	6	3	256
osm_new-jersey-AM3	4	6	3	256
osm_new-mexico-AM1	3	3	2	182
osm_new-mexico-AM2	3	3	2	182
osm_new-mexico-AM3	3	3	2	182
osm_new-york-AM1	42	118	13	13 187
osm_new-york-AM2	224	6 399	112	14 330
osm_new-york-AM3	837	88 728	511	16 268
osm_north-carolina-AM1	93	150	11	45 254
osm_north-carolina-AM2	398	10 116	140	46 896
osm_north-carolina-AM3	1 557	236 739	709	49 720
osm_ohio-AM1	78	96	8	50 964
osm_ohio-AM2	211	1 815	49	51 289
osm_ohio-AM3	482	11 376	137	52 634
osm_oregon-AM1	381	996	36	161 298
osm_oregon-AM2	1 325	57 517	356	165 047
osm_oregon-AM3	5 588	2 912 701	2 906	-
osm_pennsylvania-AM1	193	276	10	133 914
osm_pennsylvania-AM2	521	3 812	43	138 413
osm_pennsylvania-AM3	1 148	26 464	173	143 870
osm_puerto-rico-AM1	60	63	7	29 802
osm_puerto-rico-AM2	165	1 285	45	32 921
osm_puerto-rico-AM3	494	26 926	246	33 590
osm_rhode-island-AM1	455	1 973	49	171 224

Continuation of Table C.1				
Graph	$\|V\|$	$\|E\|$	$d_{\max}$	α_ω
osm_rhode-island-AM2	2 866	295 488	692	184 596
osm_rhode-island-AM3	15 124	12 622 219	5 930	-
osm_south-carolina-AM1	75	69	6	50 033
osm_south-carolina-AM2	165	713	28	51 446
osm_south-carolina-AM3	317	4 508	89	52 087
osm_tennessee-AM1	49	39	4	29 569
osm_tennessee-AM2	100	418	25	31 567
osm_tennessee-AM3	212	3 215	73	32 276
osm_utah-AM1	230	309	11	87 856
osm_utah-AM2	589	4 692	67	95 087
osm_utah-AM3	1 339	42 872	268	98 847
osm_vermont-AM1	128	418	28	55 884
osm_vermont-AM2	766	37 607	272	59 310
osm_vermont-AM3	3 436	1 136 164	1 608	-
osm_virginia-AM1	570	1 480	25	280 936
osm_virginia-AM2	2 279	60 040	174	295 867
osm_virginia-AM3	6 185	665 903	775	-
osm_washington-AM1	713	2 316	32	296 653
osm_washington-AM2	3 025	152 449	401	305 619
osm_washington-AM3	10 022	2 346 213	1 986	-
osm_west-virginia-AM1	65	150	13	42 868
osm_west-virginia-AM2	317	8 328	124	45 923
osm_west-virginia-AM3	1 185	125 620	561	47 927
osm_wisconsin-AM1	54	51	5	44 608
osm_wisconsin-AM2	89	219	14	44 651
osm_wisconsin-AM3	136	588	24	47 904
osm_wyoming-AM1	7	11	4	4 568
osm_wyoming-AM2	8	16	5	4 568
osm_wyoming-AM3	12	42	9	4 568
snap_as-skitter-uniform	1 696 415	11 095 298	35 455	-
snap_ca-AstroPh-uniform	18 772	198 050	504	797 510
snap_ca-CondMat-uniform	23 133	93 439	279	1 147 950
snap_ca-GrQc-uniform	5 242	14 484	81	286 489
snap_ca-HepPh-uniform	12 008	118 489	491	581 039
snap_ca-HepTh-uniform	9 877	25 973	65	562 004
snap_com-amazon	334 863	925 869	549	19 271 031
snap_com-youtube	1 134 890	2 987 624	28 754	90 295 294
snap_email-Enron-uniform	36 692	183 831	1 383	2 464 935
snap_email-EuAll-uniform	265 214	364 481	7 636	25 286 322
snap_loc-gowalla_edges	196 591	950 327	14 730	12 276 929

Continuation of Table C.1				
Graph	$\lvert V\rvert$	$\lvert E\rvert$	$d_{\max}$	α_ω
snap_p2p-Gnutella04-uniform	10 876	39 994	103	679 111
snap_p2p-Gnutella05-uniform	8 846	31 839	88	554 943
snap_p2p-Gnutella06-uniform	8 717	31 525	115	548 612
snap_p2p-Gnutella08-uniform	6 301	20 777	97	434 577
snap_p2p-Gnutella09-uniform	8 114	26 013	102	568 439
snap_p2p-Gnutella24-uniform	26 518	65 369	355	1 984 567
snap_p2p-Gnutella25-uniform	22 687	54 705	66	1 701 967
snap_p2p-Gnutella30-uniform	36 682	88 328	55	2 787 907
snap_p2p-Gnutella31-uniform	62 586	147 892	95	4 776 986
snap_roadNet-CA-uniform	1 965 206	2 766 607	12	111 360 828
snap_roadNet-PA-uniform	1 088 092	1 541 898	9	61 731 589
snap_roadNet-TX-uniform	1 379 917	1 921 660	12	78 599 946
snap_soc-Epinions1-uniform	75 879	405 740	3 044	5 690 970
snap_soc-LiveJournal1-uniform	4 847 571	42 851 237	20 333	-
snap_soc-Slashdot0811-uniform	77 360	469 180	2 539	5 660 899
snap_soc-Slashdot0902-uniform	82 168	504 230	2 552	5 971 849
snap_soc-pokec-relationships-u.	1 632 803	22 301 964	14 854	-
snap_web-BerkStan-uniform	685 230	6 649 470	84 230	43 907 482
snap_web-Google-uniform	875 713	4 322 051	6 332	56 326 504
snap_web-NotreDame-uniform	325 729	1 090 108	10 721	26 016 941
snap_web-Stanford-uniform	281 903	1 992 636	38 625	17 792 930
snap_wiki-Talk-uniform	2 394 385	4 659 565	100 029	235 837 346
snap_wiki-Vote-uniform	7 115	100 762	1 065	500 079
ssmc_ca2010	710 145	1 744 683	141	16 869 550
ssmc_fl2010	484 481	1 173 147	177	8 743 506
ssmc_ga2010	291 086	709 028	85	4 644 417
ssmc_il2010	451 554	1 082 232	90	5 998 539
ssmc_nh2010	48 837	117 275	74	588 996
ssmc_ri2010	25 181	62 875	44	459 275

Table C.1: Sizes of the vertex and edge sets, maximum degree, and, if known, the weight of an *MWIS* for all 207 graph instances in the evaluation dataset.

Abstract (German)

Diese Arbeit beschäftigt sich mit dem Maximum Weight Independent Set-Problem auf Graphen mit gewichteten Knoten. *Data Reduction*-Methoden werden typischerweise eingesetzt, um die Größe der Probleminstanz zu verringern. Die Frage, die in dieser Arbeit geklärt wird, ist ob man durch Modifikationen, die die Problemgröße stattdessen erhöhen, den Graph am Ende auf eine noch kleinere Größe reduzieren kann. Dafür entwickelten wir mehrere sogenannte *Backward Reduction Rules*, welche in einem zyklischen Algorithmus Anwendung finden. Durch experimentelle Evaluation konnten wir feststellen, dass diese Methode, angewandt als *Preprocessing*, tatsächlich effektiv ist, um Lösungen von höherer Qualität zu finden.

Bibliography

[1] David A. Bader, Andrea Kappes, Henning Meyerhenke, Peter Sanders, Christian Schulz, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. In Reda Alhajj and Jon G. Rokne, editors, *Encyclopedia of Social Network Analysis and Mining, 2nd Edition*. Springer, 2018.

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

[3] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

[4] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

[5] Ch. Ebenegger, P.L. Hammer, and D. de Werra. Pseudo-boolean functions and stability of graphs. In R.E. Burkard, R.A. Cuninghame-Green, and U. Zimmermann, editors, *Algebraic and Combinatorial Methods in Operations Research*, volume 95 of *North-Holland Mathematics Studies*, pages 83–97. North-Holland, 1984.

[6] Aleksander Figiel, Vincent Froese, André Nichterlein, and Rolf Niedermeier. There and back again: On applying data reduction rules by undoing others. *arXiv preprint arXiv:2206.14698*, 2022.

[7] Alexander Gellner, Sebastian Lamm, Christian Schulz, Darren Strash, and Bogdán Zaválnij. Boosting data reduction for the maximum weight independent set problem using increasing transformations. In Martin Farach-Colton and Sabine Storandt, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*, pages 128–142. SIAM, 2021.

[8] Ernestine Großmann, Sebastian Lamm, Christian Schulz, and Darren Strash. Finding near-optimal weight independent sets at scale. In Sara Silva and Luís Paquete, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, pages 293–302. ACM, 2023.

[9] Jiewei Gu, Weiguo Zheng, Yuzheng Cai, and Peng Peng. Towards computing a near-maximum weighted independent set on massive graphs. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 467–477. ACM, 2021.

[10] Sen Huang, Mingyu Xiao, and Xiaoyu Chen. Exact algorithms for maximum weighted independent set on sparse graphs. *CoRR*, abs/2108.12840, 2021.

[11] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[12] 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.

[13] Sebastian Lamm, Christian Schulz, Darren Strash, Robert Williger, and Huashuo Zhang. Exactly solving the maximum weight independent set problem on large real-world graphs. In Stephen G. Kobourov and Henning Meyerhenke, editors, *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*, pages 144–158. SIAM, 2019.

[14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[15] Bruno Nogueira, Rian GS Pinheiro, and Anand Subramanian. A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optimization Letters*, 12:567–583, 2018.

[16] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[17] Pedro V. Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Trans. Graph.*, 27(5):144, 2008.

[18] Alan J. Soper, Chris Walshaw, and Mark Cross. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *J. Glob. Optim.*, 29(2):225–241, 2004.

[19] Ole Tange. GNU parallel: The command-line power tool. *login Usenix Mag.*, 36(1), 2011.

[20] 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.

[21] 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.

[22] Weiguo Zheng, Jiewei Gu, Peng Peng, and Jeffrey Xu Yu. Efficient weighted independent set computation over large graphs. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1970–1973. IEEE, 2020.