Karlsruhe Institute of Technology

A Practical Analysis of Kernelization Techniques for the Maximum Cut Problem

Master's Thesis of

Damir Ferizovic

at the Department of Informatics
Institute of Theoretical Computer Science

Reviewer:	Prof. Dr. Peter Sanders
Second reviewer:	Prof. Dr. Dorothea Wagner
Advisors:	M.Sc. Demian Hespe
	M.Sc. Sebastian Lamm
	Dr. Matthias Mnich, University of Bonn
	Dr. rer. nat. Christian Schulz, University of Vienna
	Dr. Darren Strash, Hamilton College

1. October 2018 – 31. March 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, March 4, 2019

. . . . . .

(Damir Ferizovic)

Abstract

We examine the application of existing and new kernelization techniques for a well-known NP-hard problem, Max-Cut. Given an undirected graph, the task is to find a bipartition of the vertex set that maximizes the total weight of edges that have their endpoints in different partitions. We primarily focus on the unweighted case, but we also consider the signed and weighted versions to some extent.

Reduction rules are effective for solving many NP-hard problems in practice. They compute a smaller problem instance – called a kernel – with the imposed requirement that solving it also allows one to (quickly) compute a solution for the initially given instance.

Areas with interest in Max-Cut encompass both theory and practice. Practical fields where utilization exists include modeling of social networks [Har59], portfolio risk analysis [HLW02], network design [Bar96], VLSI (Very-Large Scale Integration) design and statistical physics [Bar+88], and image segmentation [SHK13]. Due to this, and the successful studies of kernelization in other hard problems [Abu+04] [Lam+17] [HSS18], we investigate the practical utility of kernelization for Max-Cut – a missing study until now.

We consider reduction rules from previous works and further introduce new rules. We show from a theoretical and practical viewpoint that kernelization performs best on graphs containing loosely connected subgraphs and that no more than six rules are needed to achieve a kernel size not larger than what is possible with the reduction rules from previous works.

Our results reveal that kernelization has a significant positive impact on a large collection of graphs, including both synthetic instances, and real-world graphs from VLSI design and image segmentation. In instances where kernelization succeeds in reducing the graphs, existing current state of the art Max-Cut solvers converge faster towards a maximum cut. In fact, in 9 of the 19 selected real-world instances, we compute a solution in seconds, whereas without kernelization 10 hours were not enough.

Zusammenfassung

Wir untersuchen die Anwendbarkeit von existierenden und neuen Techniken der Problemkern-Reduktion für das bekannte NP-schwere Problem, Max-Cut. Für einen gegebenen ungerichteten Graphen, besteht dabei die Aufgabe eine Bipartition der Knotenmenge zu finden, sodass das gesamte Gewicht der Kanten mit Endpunkten in verschiedenen Partitionen maximiert wird. Wir fokussieren uns primär auf den ungewichteten Fall, untersuchen aber auch bis zu einem gewissen Grad den vorzeichenbehafteten und gewichteten Fall.

In der Praxis sind Reduktionsregeln für das Lösen vieler NP-schwerer Probleme hilfreich. Sie liefern eine kleinere Probleminstanz – genannt "Kern" – mit der Anforderung, dass eine Lösung der reduzierten Instanz auch ein (schnelles) Lösen der initialen Instanz erlaubt.

Max-Cut ist sowohl für die Theorie als auch Praxis von Interesse. Praktische Anwendungen umfassen die Modellierung von sozialen Netzwerken [Har59], Risiko-Analyse von Portfolios [HLW02], Netzwerk-Design [Bar96], VLSI-Design (Very-Large Scale Integration) und statistische Physik [Bar+88], und Bildsegmentierung [SHK13]. Aufgrund des Erfolges der Problemkern-Reduktion auf andere schwierige Probleme [Abu+04] [Lam+17] [HSS18], untersuchen wir deren praktischen Nutzen für Max-Cut . Dies ist eine fehlende Studie bis zu diesem Zeitpunkt.

Dafür betrachten wir die Reduktionsregeln vorheriger Arbeiten und führen neue Regeln ein. Wir zeigen sowohl aus einer theoretischer als auch praktischer Sicht, dass die Problemkern-Reduktion am besten für Graphen mit schwach verbundenen Subgraphen anwendbar ist. Wir demonstrieren, dass bereits sechs Reduktionsregeln ausreichend sind um die Größe des Kerns mindestens so zu reduzieren, wie es mit bisherigen Regeln möglich ist.

Unsere Resultate zeigen, dass die Problemkern-Reduktion eine signifikante Auswirkung auf eine große Menge von Graphen hat. Hier eingeschlossen sind synthetische Instanzen und Graphen aus Anwendungsbereichen des VLSI-Designs und der Bildsegmentierung. Für Instanzen bei denen die Problemkern-Reduktion erfolgreich den Graphen reduziert, konvergieren existierende Max-Cut Löser schneller zu einem maximalen Schnitt. Tatsächlich berechnen wir in 9 von den 19 Instanzen einen maximalen Schnitt in wenigen Sekunden, während es ohne die Problemkern-Reduktion mehr als 10 Stunden dauert.

Contents

1 Introduction

1.1 Motivation

The goal of the (unweighted) Max-Cut problem is to partition the vertex set of a given graph $G = (V, E)$ into two sets S and $V \setminus S$ such that the total number of edges between those two sets is maximized (see Figure 1.1). The problem of determining the maximum cut of a graph is a famous problem in the area of computer science, particularly because it is a well-known NP-complete problem. Furthermore, the weighted version is one of Karp's 21 NP-complete problems [Kar72].

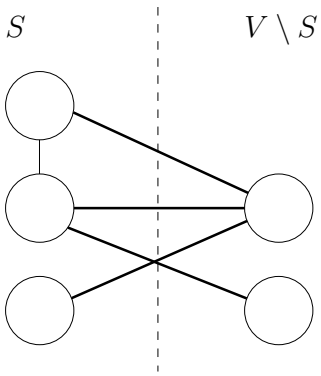

Figure 1.1: A bipartition $S, V \setminus S$ inducing a maximum cut of size 4 is shown.

While utilization for the signed and weighted version of the Max-Cut problem exists in multiple areas [Har59] [Bar82] [HLW02] [Chi+07] [Bar96] [Bar+88] [SHK13], the unweighted case acts as a proxy for these variants and presents a significant challenge for researchers.

Use cases for the Max-Cut problem with signed edges include modeling of social networks [Har59], statistical physics [Bar82], portfolio risk analysis [HLW02], and VLSI design [Chi+07]; while the version with weighted edges is used in network design [Bar96], VLSI design and statistical physics [Bar+88], and image segmentation [SHK13].

In VLSI design, Max-Cut is able to solve the *detailed routing* in the *Knock-Knee Model* [Len12]; also called *constrained via minimization problem (CVMP)*. The task is to minimize the amount of necessary holes (also called "vias") in a circuit board. We desire their amount to be small as they present a costly feature and negatively impact the product's quality. When limiting the maximum vertex degree to three, the resulting model has a *planar* structure (it is possible to draw it cross-free on a plane). In those cases, Max-Cut actually has a polynomial time solution [Had75]. However, interest in applications remain where the structure is not planar [XK88] [Bar+88].

Two essential elements of circuit boards are pins and wires, where each wire is required to connect two predetermined pins. Due to planarity not always being given, the problem of wires having to cross each other arises. To mitigate this, layering of the circuit boards is introduced and holes are created to allow traversal from one layer into another. The task of minimizing the amount of necessary holes arises, which is a Max-Cut problem instance. To achieve this, two numbers are computed for every pair of pins connected by a wire: The amount of required holes when the endpoints are on the same layer; and the amount when they are on different ones. See a more elaborate explanation on this within the work of Barahona et al. [Bar+88].

Kernelization is a powerful technique that has been utilized to improve the results of many NP-hard problems [Abu+04] [Lam+17] [HSS18]; however not much has been done in this direction for the Max-Cut problem.

To the best of our knowledge, nothing exists in the area of studying kernelization and Max-Cut with the sole goal of achieving small kernels in practice. Although, plenty of theoretical studies exist. Their focus was primarily set on providing bounds over the parameters of slightly altered Max-Cut problems – including (Signed) Max-Cut Above Edwards-Erdős [EM18] [Suc+17] [CJM15] [Cro+13] and Max-Cut Above Spanning Tree [MSZ18]. In the first one of these, for a given connected graph G, a parameter $k \in \mathbb{N}_0$ is chosen and we check for the existence of a cut with size $k + \frac{|E|}{2} + \frac{|V|-1}{4}$. The later part of the expression, $\frac{|E|}{2} + \frac{|V|-1}{4}$, denotes the Edwards-Erdős bound – one can always find a bipartition inducing a cut of that size. Similar applies for Max-Cut Above Spanning Tree where we consider a lower bound of $|V| - 1$ for the size of a cut.

Therefore, both of these outlined variations investigate the properties of cuts above proven lower bounds. One of such works by Etscheid and Mnich [EM18] is the primary source of inspiration for this work. They showed that efficient kernelization is possible with promising theoretical bounds on the size of the kernel. Meaning, that for a given k in (Signed) Max-Cut Above Edwards-Erdős, one can construct a kernel of size $O(k)$ in time $O(k \cdot (|V| + |E|))$.

1.2 Contribution

Our contributions are manifold on the utilization of kernelization for the Max-Cut problem. We thoroughly summarize all 22 found reduction rules from previous works, analyze the context in which they are used, and show how one can encompass 20 of those with a reduced set of four reduction rules. Alongside this, the reduction rules by Etscheid and Mnich [EM18] are examined in more detail. We do this due to their seemingly small kernel of size $O(k)$ for the problem (Signed) Max-Cut Above Edwards-Erdős and to show that the theory is relatively disconnected from achieving a small kernel in practice.

Across our whole work, several theoretical insights are also given. We outline the limitations of current kernelization techniques and the areas where further research may yield more reductions. To that goal, we give a comprehensive overview on how effective our kernelization is on very small subgraphs. We emphasize this by also providing the amount of kernelization that theoretically remains possible.

Key details of our implementation are given when a relevant concept is introduced. At the conclusion of our work, we provide an evaluation from different angles. We show that kernelization achieves a significant reduction on sparse graphs. We also present the benefits of permitting weighted edges in the final kernel. These benefits are particularly interesting in our evaluation of the current state of the art Max-Cut solvers, including: Localsolver [Ben+11] [Gar+14], MqLib [DGS15] (utilizing the heuristic by Burer, Monteiro, and Zhang [BMZ02]), and Biq Mac [RRW10]. Moreover, we show that kernelization oftentimes helps deliver a faster convergence towards a maximum cut. This also includes real-world instances where Max-Cut is used.

1.3 Thesis Structure

After having gone over the preliminaries in Chapter 2, we proceed with an introduction of previous research in Chapter 3. Here, an overview of relevant work that utilized kernelization is given – stemming mostly from the theoretical domain. Within this overview, we provide an exhaustive list of found reduction rules.

Subsequently, we address our contributions until the end. First, we introduce newly developed reduction rules in Chapter 4. Provided here is also an overview of the interactions among our newly developed rules and those from previous works.

In Chapter 5 and 6, we give important aspects of our implementation. Chapter 5 outlines the algorithms we used to transform between different Max-Cut variations (e.g., how to transform a Signed Max-Cut problem into a Weighted Max-Cut). In Chapter 6, we describe the structure of our implementation and the utilized timestamping system to avoid rechecking the same structures for the same reduction.

Finally, we supply a practical evaluation of our contributions in Chapter 7. It begins with an assessment of our implementation of the reduction rules by Etscheid and Mnich [EM18]. Following that, we then utilize our full suite – including all our reduction rules – and compute the kernels for a large set of randomly generated graphs. These tests equip us with good tools to determine when kernelization performs well and when not. Moreover, this provides guidance in understanding our achieved results on a set of selected real-world instances. This is also where we evaluate the impact of kernelization on current state of the art Max-Cut solvers. Lastly, a total kernelizability analysis of very small subgraphs is provided. We do this to show where current kernelization performs best, where worst, and to give a direction where more research is able to yield further improvements.

2 Preliminaries

2.1 Graphs

In this work, we are primarily concerned with unweighted and undirected graphs, without multi-edges or self-loops. If not explicitly stated otherwise, we always refer to such graphs.

Let V be a set of elements $\{v_1, ..., v_n\}$, for $n \in \mathbb{N}$ – the *vertex set* – and let $E \subseteq V \times V$ be the *edge set*, with the condition $(a, b) \in E$ implies $(b, a) \in E$ (making it an undirected graph). Then, the ordered pair $G = (V, E)$ is considered an *undirected graph*. Moreover, an undirected graph in which every pair of vertices is connected by an edge is called a *complete graph*. *Weighted graph* instances are also being considered throughout our work. Meaning, the edge set of the graph is complemented with a function $w : E \to \mathbb{R}$. We represent such weighted graph instances with the notation (G, w).

To easier denote the vertices and edges of a graph G, $V(G)$ and $E(G)$ are used, respectively. Occasionally, we will also be interested in the edges between the vertices of different vertex sets $S_1, S_2 \subseteq V(G)$. We denote that set by $E_G(S_1, S_2)$.

For any undirected graph $G = (V, E)$, $\{a, b\} \in E$ denotes an undirected edge in G. Therefore, $\{a, b\} \in E$ implies $(a, b) \in E \wedge (b, a) \in E$. The *neighborhood* of a vertex v is given by $N_G(v) = \{w \in V \mid \{v, w\} \in E\}$. We also define the neighborhood of a vertex set $X \subseteq V$ as $N_G(X) = \bigcup_{v \in X} N_G(v) \setminus X$. The *degree* of a vertex v is then defined as $deg_G(v) := |N_G(v)|$.

When comparing two graphs G_1 and G_2, it is possible for them to have the same topology. This property is characterized through an *isomorphism* – a bijection ξ between $V(G_1)$ and $V(G_2)$ such that $\{u, v\} \in E(G_1)$ if and only if $\{\xi(u), \xi(v)\} \in E(G_2)$. When an isomorphism for G_1 and G_2 exists, the graphs are called *isomorphic*; otherwise, they are *non-isomorphic*. Observe that isomorphisms form an equivalence relation on graphs.

A *path* is any sequence of vertices $\langle v_1, \ldots, v_k \rangle$ such that it holds $\{v_i, v_{i+1}\} \in E(G)$, for $i = 1, \ldots, k-1$, and no vertex repeats in the sequence. We also use the notation $v_1 \ldots v_k$ to describe a path. In both cases, the *length* of the path is given by the value $k - 1$. An *induced ℓ-path* is a path of length ℓ and for each non-successive pair of vertices, there is no direct edge connection between them. A path $v_1 \ldots v_k$ for which only the vertices v_1 and v_k are the same ($v_1 = v_k$) is furthermore called a *cycle*. The size of a cycle is given by the value of k.

We call a graph $G = (V, E)$ *connected* if at least one path from v to w exists for any two vertices $v, w \in V$; and *disconnected* otherwise. Similarly, two vertices are called connected, if at least one path between them exists. If, in a connected graph $G = (V, E)$, there is a vertex $v \in V$ whose removal leads to the graph becoming disconnected, then that vertex is called an *articulation point* or *cut vertex*.

2.2 Graph Components

Let the following be given: A graph $G = (V, E)$, a vertex subset $V' \subseteq V$, and an edge subset $E' \subseteq E$, with $E' \subseteq V' \times V'$. We then define $G' = (V', E')$ as a *subgraph* of G.

When performing kernelization on a graph $G = (V, E)$, it is often crucial to search for specific subgraphs of a graph. We are primarily focused on *induced subgraphs*. For a vertex subset $S \subseteq V$, an induced subgraph is defined as $G[S] := (S, E[S])$, where we define the edge set as $E[S] := \{\{a, b\} \mid a, b \in S \wedge \{a, b\} \in E\}$. That is, an induced subgraph of G for a vertex subset $S \subseteq V$ is a graph that contains all edges between the vertices of S in G. An induced subgraph that is a complete graph is also referred to as a *clique*. Another term useful when working with cliques is K_p, which denotes a clique containing p vertices. A clique with 3 vertices is also called a *triangle*.

This now naturally inspires the definition of a *(connected) component* of G: A maximal vertex set S such that all vertices in $G[S]$ are connected with each other. Employed also is the adjective *"isolated"* to describe such components in a shorter manner (example: "isolated cycle").

Let $k \in \mathbb{N}$ be the smallest number of vertices whose removal disconnects a graph G with at least $k + 2$ vertices. The graph G is then called *k-connected*. The same terminology also applies to subgraphs/components. A 2-connected graph is also often referred to as *biconnected* and a maximal 2-connected subgraph as a *block*.

2.3 External/Internal Vertices

For a graph $G = (V, E)$, let $S \subseteq V$ be any vertex subset. The set of *external vertices* of $G[S]$ is defined as $C_{\text{ext}(G)}(S) = \{v \in S \mid \exists w \in V \setminus S, \{v, w\} \in E\}$. In other words, this is the set of vertices in S which contain at least one neighbor in G that is not in S. In similar fashion, $C_{\text{int}(G)}(S) = S \setminus C_{\text{ext}(G)}(S)$ defines the set of *internal vertices*. Also in use for a subgraph H of G is $C_{\text{int}(G)}(H) := C_{\text{int}(G)}(V(H))$ and $C_{\text{ext}(G)}(H) := C_{\text{ext}(G)}(V(H))$. See Figure 2.1. Beware that this definition of exterior/interior vertices slightly differentiates from the one within related works on this topic.

For $m \leq n$, we define $(n, m)_C$ as the set of subgraphs with n vertices of which the first m within the ordered vertex set $v_1, \ldots, v_n$ are external. Note that $|(n, m)_C| = 2^{\binom{n}{2}}$. This size is calculated by counting the number of graphs one can form with n vertices; which is done by counting all possible edge sets for n vertices. Since the definition of $(n, m)_C$ uniquely defines the external vertex set for each graph, the amount of external vertices does not partake on the size of the set.

2.4 Maximum Cut

The goal in the (Unweighted) Max-Cut problem for an undirected and unweighted graph $G = (V, E)$ is to determine a vertex set S such that $|E_G(S, V \setminus S)|$ is maximized. Occasionally, the notion $V_0 \cup V_1 = V$ is also used to denote a bipartition. While the bipartition $(S, V \setminus S)$ is a *maximum cut*, a *cut* is simply any bipartition $(H, V \setminus H)$ of the graph. The size of a maximum cut is then given by $\beta(G) := |E_G(S, V \setminus S)|$. Another way to

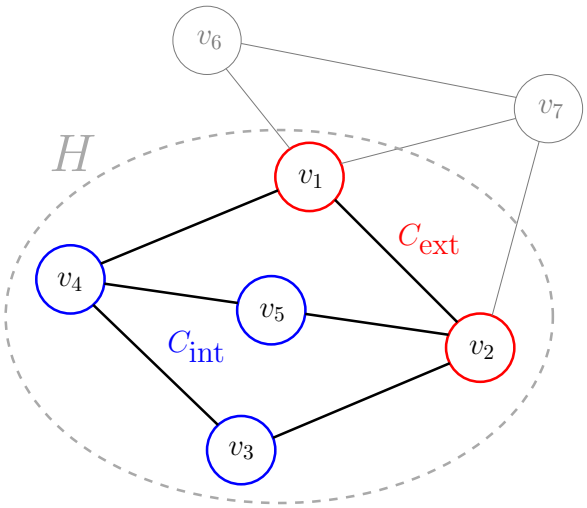

Figure 2.1: A subgraph H of some graph with vertices $\{v_1, v_2, v_3, v_4\}$ and edges $\{\{v_1, v_2\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_5\}\}$ is depicted. Vertices in red are external; those in blue, internal. Therefore, $C_{\text{ext}(G)}(H) = \{v_1, v_2\}$ and $C_{\text{int}(G)}(H) = \{v_3, v_4, v_5\}$. The membership $H \in (5, 2)_C$ holds.

define the Max-Cut problem is through a *2-coloring*: Color each vertex of G with one of two colors (denoted as "0" and "1") such that the number of edges connecting two different colors is of maximum size. In this case, the coloring is given by a function: $\delta : V \to \{0, 1\}$.

Let δ be a fixed 2-coloring of a vertex subset $X \subseteq V(G)$. We call the problem of finding the largest cut with respect to the given δ a Local Max-Cut problem and denote the size of a local maximum cut by $\beta_\delta(G)$.

We show through Lemma 1 that $\beta_\delta(G)$ is indeed a local optimum by providing a lower bound for the size of a maximum cut.

Lemma 1. *Let $G = (V, E)$ be a graph, $X \subseteq V$, and $\delta : X \to \{0, 1\}$. Then $\beta(G) \geq \beta_\delta(G)$ holds.*

Proof. If $\beta(G) < \beta_\delta(G)$, then $\beta(G)$ is not the size of a maximum cut. □

Such a local optimum may also be a (*global*) maximum cut, but only if at least one maximum cut of G exists that does have the same coloring for the selected vertex subset. This is summarized by Lemma 2.

Lemma 2. *Let $G = (V, E)$ be a graph, $X \subseteq V$, and $\delta : X \to \{0, 1\}$. If a maximum cut of G exists whose vertices in X are colored according to δ, then $\beta_\delta(G) = \beta(G)$.*

Proof. True by definition. The value $\beta_\delta(G)$ is the size of a maximum cut with X partitioned according to the 2-coloring δ. □

Knowledge about the Local Max-Cut problem motivates the importance of graph connectivity. We define $\mathcal{D}(G, X) := \{G[V(C) \cup X] : C \text{ connected component in } G[V \setminus X]\}$. See Figure 2.2 for an example. As will be evident through Lemma 3, this set helps us divide a Max-Cut problem into smaller subproblems.

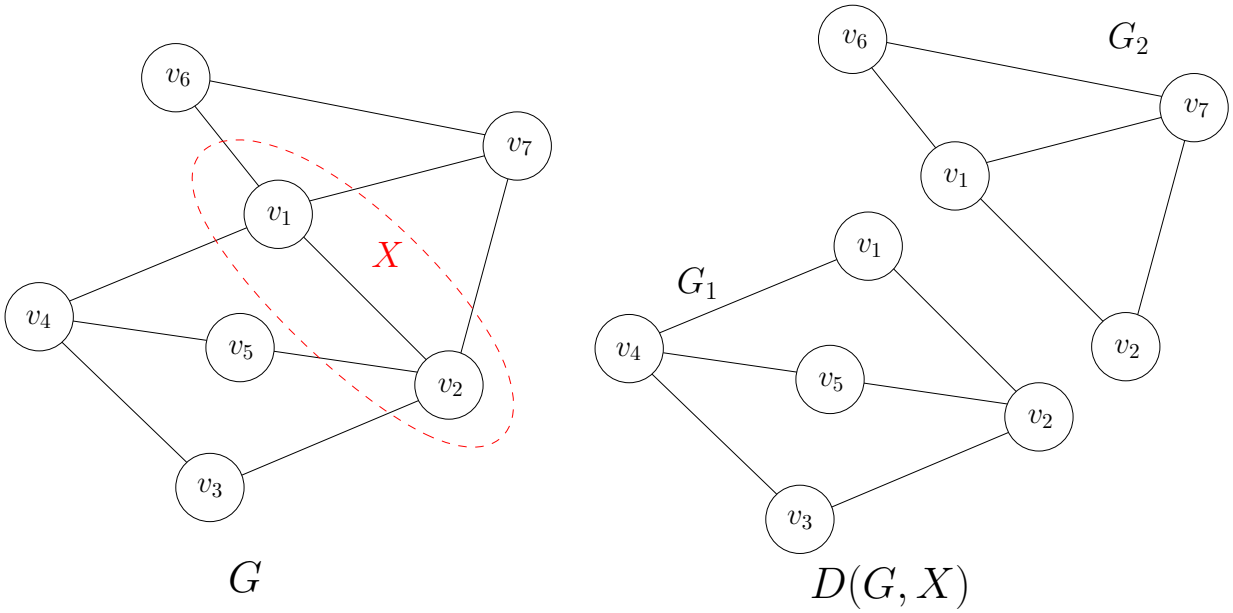

Figure 2.2: Exemplary presentation of the set $D(G, X)$ for a graph G. The set $D(G, X)$ consisting of two graphs, G_1 and G_2.

Lemma 3. *Let $G = (V, E)$ be a graph, $X \subseteq V$, and $\delta : X \to \{0, 1\}$. The followings holds:*

$$\beta_\delta(G) = \sum_{G' \in \mathcal{D}(G,X)} (\beta_\delta(G') - \beta_\delta(G[X])) + \beta_\delta(G[X]).$$

Proof. For any two different $G_1, G_2 \in \mathcal{D}(G, X)$ the relationship $E(G_1) \cap E(G_2) = E(X)$ and $V(G_1) \cap V(G_2) = X$ holds. Therefore, only the coloring of vertices in X is able to impact the values of $\beta_\delta(G_1)$ and $\beta_\delta(G_2)$ at the same time. As the vertices in X already have a fixed coloring, all connected graphs in $\mathcal{D}(G, X)$ may be independently optimized when determining the value of $\beta_\delta(G)$. Furthermore, when adding $\beta_\delta(G_1) + \beta_\delta(G_2)$, the value of $\beta_\delta(G[X])$ is counted twice. That is why we substract $\beta_\delta(G[X])$ for each element of $\mathcal{D}(G, X)$ and add it once at the end. □

Throughout this work, a few decision problems are shown. We proceed to define our first and most basic decision problem related to our work – the definition of the Max-Cut decision problem.

Definition 2.4.1. Let G be a graph and $k \in \mathbb{N}_0$, we denote an instance of the Max-Cut decision problem as $(G, k)_{\text{MC}}$. If the size of a maximum cut in G is k, then $(G, k)_{\text{MC}}$ is a "yes"-instance; otherwise, it is a "no"-instance.

2.4.1 Weighted Maximum Cut

A closely related subject to the Unweighted Max-Cut problem is the Weighted Max-Cut problem. As mentioned earlier, in the case of weighted graphs $G = (V, E)$, a weight function $w : V \to \mathbb{Z}$ is additionally provided – forming the weighted graph instance (G, w).

The Weighted Max-Cut problem then asks for a bipartition $S \subseteq V, V \setminus S$, such that the added total weight of all edges in $E_G(S, V \setminus S)$ is maximized. Meaning, we want to find an $S \subseteq V$ such that $\beta(G, w) := \sum_{e \in E_G(S, V \setminus S)} w(e)$ is maximized.

2.4.2 Signed Maximum Cut

In a Signed Max-Cut problem instance, each edge in the graph $G = (V, E)$ is given a label $l : E \to \{\text{“+”}, \text{“−”}\}$ – forming a *signed graph* (G, l). The task is then to find a vertex set bipartition $S \subseteq V, V \setminus S$, such that the number of edges satisfying the following constraints is maximized:

- Edges with sign “+” and endpoints in the same set of the bipartition,
- Edges with sign “−” and endpoints in different sets of the bipartition.

Or more formally, we want to find an $S \subseteq V$ such that the size of the cut as given by $\beta(G, l) := |E^-_{G,l}(S, V \setminus S)| + |E^+(G[S], l) \cup E^+(G[V \setminus S], l)|$ is maximized, where we define $E^c_{G,l}(S, V \setminus S) := \{e \in E_G(S, V \setminus S) \mid l(e) = c\}$ and $E^c(G, l) := \{e \in E(G) \mid l(e) = c\}$ for $c \in \{\text{“−”}, \text{“+”}\}$. Similarly, for the neighborhood of a vertex (set), we use the notations $N^c_{G,l}(v) := \{w \in V \mid \{v, w\} \in E^c(G, l)\}$ and $N^c_{G,l}(X) := \bigcup_{v \in X} N^c_{G,l}(v) \setminus X$. We call a triangle *positive* if its amount of “−” edges is even.

An Unweighted Max-Cut instance can be transformed into as a Signed Max-Cut problem by labeling all edges with “−”.

2.5 Fixed-Parameter Tractable (FPT)

FPT is the class of problems which are *fixed-parameter tractable* [DF12]. For a problem from this class, it is possible to limit the combinatorial explosion with a parameter k, which is used to parametrize an important property of the given input. It may be an upper bound for a cycle; for counting problems, it could be used as “how much above a lower bound does the solution lie in“; one could use it to characterize graph connectivity; etc.

In the next definition, we use Σ^* to denote the set of input instances for a decision problem.

Definition 2.5.1 (Fixed Parameter Tractability by Cygan et al. [Cyg+])**.** A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$ (called a *fixed-parameter algorithm*) and a computable function $f : \mathbb{N} \to \mathbb{N}$ such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^{O(1)}$. The complexity class containing all fixed-parameter tractable problems is called FPT.

The running time of fixed-parameter algorithms is characterized by the term $|(x, k)|^{O(1)}$ from the above definition. So, even though a problem might not admit a polynomial time solution in general, a *polynomial fixed-parameter algorithm* may exist. Same applies to *linear time fixed-parameter algorithm.*

2.6 Reduction Rules and Kernelization

Oftentimes, it is possible to quickly solve a Max-Cut decision problem $(G, k)_{\text{MC}}$ through a known solution for another instance $(G', k')_{\text{MC}}$. The act of determining such a related instance $(G', k')_{\text{MC}}$ (also named *kernel*) is called *kernelization*. The process of kernelization almost always involves multiple transformations of the initially given instance. Such transformations are also called *reductions*. We proceed to provide a formal description of what a reduction rule entails.

Definition 2.6.1 (Reduction Rule by Cygan et al. [Cyg+])**.** A *reduction rule* for a parametrized problem Q is a function $\phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance (I, k) of Q to an equivalent instance (I', k') of Q such that ϕ is computable in time polynomial in $|I|$ and k. We say that two instances of Q are *equivalent* if $(I, k) \in Q$ if and only if $(I', k') \in Q$.

It is straightforward to see how Definition 2.6.1 applies to the Max-Cut decision problem. Observe that for two equivalent "yes" instances $(G, \beta(G))$ and $(G', \beta(G'))$, the relationship $\beta(G) = \beta(G') + k$ must necessarily hold for some $k \in \mathbb{Z}$. Whenever a reduction rule is outlined from previous and our work, this parameter k is provided.

2.7 Vertex-Weighted Max-Cut

This is a more general version of the Max-Cut problem as it introduces an additional property that may contribute to the value we want to maximize. For each vertex $v \in V$ in a graph $G = (V, E)$, two weights are assigned, $w_0 : V \to \mathbb{R}, w_1 : V \to \mathbb{R}$.

Then, instead of only maximizing the number of edges between the sets of a bipartition $V_0 \cup V_1 = V$, we are also required to consider the sum $\sum_{v \in V_0} w_0(v) + \sum_{v \in V_1} w_1(v)$.

Definition 2.7.1. Given a graph $G = (V, E)$ and vertex weights $w_0 : V \to \mathbb{R}, w_1 : V \to \mathbb{R}$, the Vertex-Weighted Max-Cut problem is to compute a bipartition $V_0 \cup V_1 = V$ such that $|E_G(V_0, V_1)| + \sum_{v \in V_0} w_0(v) + \sum_{v \in V_1} w_1(v)$ is maximized.

2.8 Special (Sub-)graph Classes

In a later section, we will show that the Vertex-Weighted Max-Cut is solvable in polynomial time for certain graph classes. Two such graph classes are described in Definition 2.8.1 and Definition 2.8.2. Both are utilized in related research. See also Figure 2.3 and 2.4 for an example of both definitions, respectively.

Definition 2.8.1 (Clique Tree/Forest by Crowston, Jones, and Mnich [CJM15])**.** A *clique tree* is a connected graph whose blocks are cliques, where a clique is a complete subgraph of a graph. A *clique forest* is a graph whose connected components are clique trees.

Definition 2.8.2 (Clique-Cycle Forest by Madathil, Saurabh, and Zehavi [MSZ18])**.** The class of *clique-cycle forests* is defined as follows. A clique is a clique-cycle forest, and so is a cycle. The disjoint union of two clique-cycle forests is a clique-cycle forest. In addition, a graph formed from a clique-cycle forest by identifying two vertices, each from a different (connected) component, is also a clique-cycle forest.

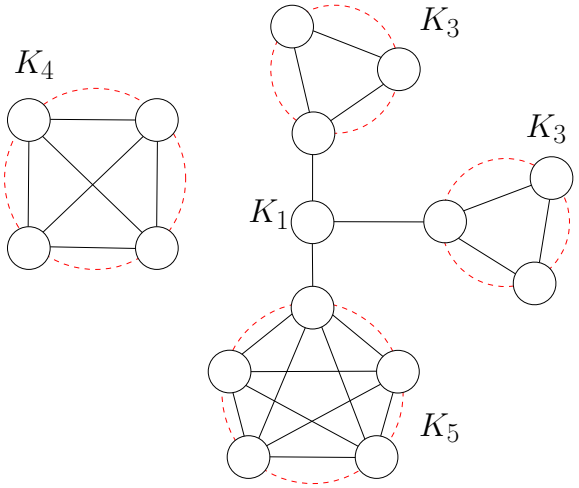

Figure 2.3: A clique forest consisting of two trees.

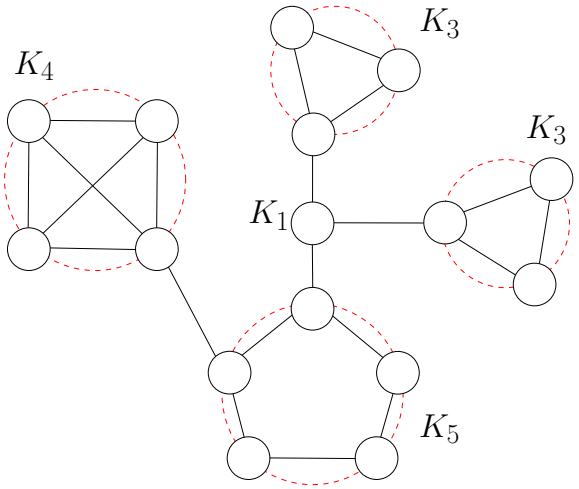

Figure 2.4: A clique-cycle forest consisting of a single tree.

3 Related Work

Several studies have been made in the direction of providing fixed-parameter algorithms for the Max-Cut problem [EM18] [Cro+13] [MSZ18] [CJM15]. Among these, a fair amount of kernelization rules have been introduced with the goal of effectively reducing Max-Cut instances [EM18] [Cro+13] [MSZ18] [Pri05] [Suc+17] [CJM15]. However, to the best of our knowledge, we are the first in this domain that conducted an analysis from a practical viewpoint. Meaning, all research related to fixed-parameter algorithms has been focused on theoretical properties, with the same observation also applying to all found kernelization rules. Relevant examples of these theoretical results follow.

3.1 Theoretical Results

For bipartite graphs, many NP-hard problems actually have polynomial time solutions. This includes Independent Set, Coloring, and Vertex Cover [Kőn31] [BM+76]. The same is true for the Max-Cut problem where all edges are in the maximum cut.

This observation motivated a fixed-parameter algorithm for Max-Cut. A graph is bipartite if and only if the graph has no odd cycles. Panolan and Rai [PR12] have shown that if one bounds the length of the longest odd cycle by parameter k, it is possible to develop a fixed-parameter algorithm that determines if the size of a maximum cut is above a given lower bound $t \in \mathbb{N}_0$ in time $4^k n^{O(1)}$.

Another way to go about creating a fixed-parameter algorithm is by utilizing known lower bounds for the size of a maximum cut. Two such bounds are the Edwards-Erdős bound by Edwards [Edw73] [Edw75] and the spanning tree bound (see Section 3.4 and 3.5, respectively). In those problem instances, a value $k \in \mathbb{N}_0$ is given. Suppose the instance has lower bound l. The algorithm then must decide if a cut of size $k + l$ exists.

Neither is superior for all graphs. As outlined by Madathil, Saurabh, and Zehavi [MSZ18], the spanning tree bound performs better than the Edwards-Erdős bound when inequality $|V| - 1 > \frac{|E|}{2} + \frac{|V|-1}{4}$ holds. Or, equivalently; when the average degree of the graph is smaller than 3. Meaning, the spanning tree bound is better suited for sparse graphs, whereas the Edwards-Erdős bound is more suited for dense graphs.

The first work which shows that the unweighted Max-Cut is fixed-parameter tractable above the Edwards-Erdős bound stems from Crowston, Jones, and Mnich [CJM15]. They have developed an algorithm that determines in time $2^{O(k)} \cdot n^4$ if a cut of size $l + k$ exists, where l is the lower bound by Edwards-Erdős. Moreover, achieved by them was a kernel of size $O(k^5)$. This was then enhanced by Crowston et al. [Cro+13] to include the Signed Max-Cut and the resulting kernel's size was decreased to $O(k^3)$. Lastly, further improvements were introduced by Etscheid and Mnich [EM18]. Runtime performance of $8^k \cdot O(m)$ was achieved alongside a kernel of size $O(k)$ – also for signed graphs.

3.2 Practical Results

Within the domain of Max-Cut, a multitude of practical approaches exist to determine a maximum cut or the largest possible cut for a collection of allocated resources. Recently, a group of heuristic solutions from previous works was evaluated by Dunning, Gupta, and Silberholz [DGS15]. They provide a thorough evaluation framework named MqLib [18b] and they show that one of the best heuristics to determine a large cut was developed by Burer, Monteiro, and Zhang [BMZ02]. However, it has no mechanisms to determine when an achieved cut is also a maximum one. Their evaluation is performed over a set of over 3000 weighted graph instances, specifically selected for the Max-Cut problem.

Another approach to solve the Max-Cut problem is available from the commercial domain by a software called Localsolver [Ben+11] [Gar+14]. While they also utilize a heuristic approach, they additionally implement a mechanism to determine when an achieved cut is a maximum cut as well. Notable also is that the Max-Cut problem is just one of many problems it is able to handle.

A solver specially developed to determine an exact maximum cut also exists. It is referred to as the Biq Mac Solver and was developed by Rendl, Rinaldi, and Wiegele [RRW10].

3.3 Vertex-Weighted Max-Cut by Etscheid and Mnich [EM18]

Before continuing to discuss the existing kernelization techniques, a solution by Etscheid and Mnich [EM18] for the Vertex-Weighted Max-Cut problem on Clique Forests is introduced. We later utilize this to briefly evaluate the potential for a new Max-Cut solver.

Consider an isolated clique tree within any given clique forest with vertex weights. To solve the Vertex-Weighted Max-Cut, we incrementally process all leaves and remove them. Let C be any such clique leaf with $|V(C)| \geq 2$ and let $r \in V(C)$ be the vertex (an articulation point) that connects C with the remainder of the tree. Further assume that all vertices $v \in V(C)$ of that clique have arbitrary values assigned to $w_0(v)$, $w_1(v)$ for the Vertex-Weighted Max-Cut problem. If C is a single clique, and does not have a vertex r as outlined above, pick any vertex for that role.

The goal now lies in updating the value of $w_0(r)$ and $w_1(r)$ such that the vertex-weighted maximum cut of C is fully described by them. That is, the value $w_0(r)$ stores the size of a maximum cut of C when r is assigned the "0" color, and $w_1(r)$ does the same for color "1".

Next task is to compute the values of $w_0(r)$ and $w_1(r)$. Without loss of generality, let r be colored with "0" ($\delta(r) = 0$) and for $x \in V(C) \setminus \{r\}$ let the values of $w_0(x)$ and $w_1(x)$ be updated accordingly – if r has an edge incident to $v \in V$, then increase $w_1(v)$ by one. Case $\delta(r) = 1$ is then symmetric to the case $\delta(r) = 0$.

We now focus on the remaining vertices of C, $X = V(C) \setminus \{r\}$. The challenge is to determine a 2-coloring of all vertices in X.

Define $\Delta(v) := w_0(v) - w_1(v)$. We know that each vertex has to be colored with either "0" or "1". Ignoring the edges, the best vertices to color with "0" are those with large $\Delta(v)$. If we color vertex v with "0", the size of the cut increases by $w_0(v)$; and if not, by $w_1(v)$. Meaning, in total, the cut increases (or possibly decreases!) by $\Delta(v)$ if vertex v is colored by "0" instead of "1". A natural next step is to sort the vertices $v \in X$ in decreasing order

of $\Delta(v)$ by utilizing counting sort. Let that order be denoted by $P = (v_1, \ldots, v_{|X|})$. All that has to be done now is to find an optimal $p \in \{0, \ldots, |X|\}$ such that vertices $(v_1, \ldots, v_p)$ are colored with "0", and vertices $(v_{p+1}, \ldots, v_{|X|})$ are colored with "1". That is, the following has to be solved:

$$\max_{p \in \{0,\ldots,|X|\}} \left(\sum_{i=1,\ldots,p} \Delta(v_i) - \sum_{i=p+1,\ldots,|X|} \Delta(v_i) + \left| E\left(\{v_1, \ldots, v_p\}, \{v_{p+1}, \ldots, v_{|X|}\} \right) \right| \right)$$

The term $|E(\{v_1, \ldots, v_p\}, \{v_{p+1}, \ldots, v_{|X|}\})|$ is added to account for the edges. Due to the regarded component being a clique, its value equals to $p \cdot (|X| - p)$. This also means that it does not matter which vertices are colored by what label, but only the amount. If we had used any other permutation than P, the amount of edges being added to a cut's size is the same for a fixed $p \in \{0, \ldots, |X|\}$. Meaning, by solving the above optimization problem in time $O(|X|)$, the Vertex-Weighted Max-Cut is also solved for the clique X.

Note that counting sort would imply a $O(|V|)$ time complexity. Although, one can bound the values of $\Delta(v)$ in range $[-|X|, |X|]$ to achieve a total time of $O(|X|)$ for each X. If, for any vertex $v \in X$, the value of $\Delta(v)$ is not in range $[-|X|, |X|]$, then the color of that vertex can be determined right away as its edges cannot compensate for the assignment of another color.

In the end, once a vertex-weighted maximum cut of X is computed for $\delta(r) = 0$, update the value of $w_0(r)$ with it, and erase all of X (after also computing $w_1(r)$). The vertex r now represents the whole component for the subsequent processing of the tree's remaining parts.

3.4 Edwards-Erdős Bound

For a connected graph, the Edwards-Erdős bound is defined as follows:

$$EE(G) = \frac{|E(G)|}{2} + \frac{|V(G)| - 1}{4}.$$

This bound stems from within the works of Edwards [Edw73] [Edw75]. Note that Erdős previously came up with a worse lower bound of size $\frac{|E(G)|}{2}$ [Erd65] which was then improved by Edwards. As shown by Madathil, Saurabh, and Zehavi [MSZ18], the Edwards-Erdős bound works significantly better for dense graphs compared to the bound given by the spanning tree.

A linear time algorithm that computes a cut satisfying the Edwards-Erdős bound for any given graph also exists and was provided by Van Ngoc and Tuza [VT93]. Moreover, these results have been extensively used in researching fixed-parameter tractability of Max-Cut [MR99] [MRS09] [GY10] [Cro+14] [EM18]. Of particular interest among these works is the Max-Cut Above Edwards-Erdős (Max-Cut AEE) problem. It asks the following: For a given $k \in \mathbb{N}_0$, does a Max-Cut of size $EE(G) + \frac{k}{4}$ exist? We denote an instance of this problem by $(G, k)_{\text{AEE}}$.

The Edwards-Erdős bound also represented the starting point of our work. New research by Etscheid and Mnich [EM18] has shown that the problem Max-Cut AEE has a linear time fixed-parameter algorithm, $8^k \cdot O(|E|)$, with a linear sized kernel $O(k)$.

From this point on, for a graph $G = (V, E)$, let $S \subseteq V$ be any vertex set such that $G - S$ is a clique forest. We also call S the *marked vertex set*. All found kernelization rules for Max-Cut AEE require this set and each of them utilize the relationship between S and $G - S$ in their requirements and effects. In a later section, a way to generalize these rules to function independently of G and $G - S$ is shown, making them directly applicable on the whole graph. This furthermore means that the changes these rules inflict on a parameter k of a $(G, k)_{\text{AEE}}$ instance are transformed into a direct change on the size of a maximum cut.

Following is a short introduction of an algorithm that computes the vertex set S, as defined by Etscheid and Mnich [EM18]. It works in $O(k \cdot (|V| + |E|))$ time and provides the bound $|S| \leq 3k$. After that, the reduction rules for Max-Cut Above Edwards-Erdős are stated.

3.4.1 Algorithm for Clique Forest Computation

To achieve the computation of an marked vertex set $S \subseteq V(G)$ such that $G - S$ is a clique forest, a "special" set of one-way reduction rules were introduced by Crowston et al. [Cro+13] – unrelated to the reduction rules as described by Definition 2.4.1. These rules are used by Etscheid and Mnich [EM18] to transform a Max-Cut AEE instance $(G, k)_{\text{AEE}}$ to $(G', k')_{\text{AEE}}$ such that if we know that $(G', k')_{\text{AEE}}$ is satisfiable, then so is $(G, k)_{\text{AEE}}$. Beware, if $(G', k')_{\text{AEE}}$ is not satisfiable, then no implications on $(G, k)_{\text{AEE}}$ exist.

Following are the one-way reduction rules that one can apply on a connected graph G. Note that whenever C is used, a clique without *positive triangles* is being referred to.

One-Way Rule 1. ([Cro+13], Rule 3) Let C be a connected component of $G - v$ for some vertex $v \in V(G)$. If there exist $a, b \in V(C)$ such that $G - \{a, b\}$ is connected and there is an edge $\{a, v\}$ but no edge $\{b, v\}$, then add a, b to S and delete them from G, and set $k' = k - 2$.

One-Way Rule 2. ([Cro+13], Rule 5) If there is a vertex $v \in V(G)$ such that $G - v$ has a connected component C such that $G[V(C) \cup \{v\}]$ is a clique, then delete C. If $|V(C)|$ is odd, set $k' = k - 1$; otherwise set $k' = k$.

One-Way Rule 3. ([Cro+13], Rule 6) If $a, b, c \in V(G)$ induce a 2-path (a, b, c) such that $G - \{a, b, c\}$ is a connected graph, then add a, b, c to S and delete them from G, and set $k' = k - 1$.

One-Way Rule 4. ([Cro+13], Rule 7) Let $v, b \in V(G)$ be such that $\{v, b\} \notin E(G)$ and C, Y are the only connected components of $G - \{v, b\}$. If $G[V(C) \cup \{v\}]$ and $G[V(C) \cup \{b\}]$ are cliques, then add v, b to S and delete them from G, delete C, and set $k' = k - 1$.

Only these 4 rules are utilized in case of the unweighted version of the Max-Cut problem. The other one-way reduction rules within the same paper are used for the more general problem – Signed Max-Cut. While we do utilize signed reduction rules to some extend, we only evaluate the marked vertex set on unweighted cases. This was sufficient for us to develop the results we want to present with this work.

After the exhaustive application of the given rules, the resulting instance $(G', k')_{\text{AEE}}$ is either a satisfiable instance or the graph G' contains no edges. In this later case, the algorithm provides the marked vertex set $S = V(G')$, which can be further utilized within the reduction rules. From all kernelization rules that are listed by us in the next section, only two are used by Etscheid and Mnich [EM18]: Reduction Rule 5 and Reduction Rule 9.

Note that a naive implementation of the above one-way reduction rules does not lead to a $O(k \cdot (|V| + |E|))$ time complexity. To achieve linear time, an algorithm was developed by Etscheid and Mnich [EM18] that handles all these rules in a specific manner. It works by iteratively removing leaf blocks from the given graph G.

Therefore, let X be a leaf block of G and r the vertex through which X is connected with $G - X$, if it exists. If such r does not exist, pick any vertex (this case only applies when the whole graph is a single block). With this, Lemma 3 by Etscheid and Mnich [EM18] states that there is always a rule applicable on X and that it can be determined and applied in $O(|E(X)|)$ time:

- If X is a clique, then One-Way Rule 2 is applicable on X and r. Process next leaf block.
- If $X - \{r\}$ is a clique. Apply One-Way Rule 1. For v select r, for a chose any vertex in $X - \{r\}$ that is adjacent to r, and for b pick any vertex that is not adjacent to r. Such vertices must exist as X is a 2-connected component, but not a whole clique. Further, $X - \{r\}$ cannot be a single-vertex clique as X is 2-connected. Process next leaf block.
- At this point it is known that neither X nor $X - \{r\}$ is a clique. If $N_X(r) = \{x, y\}$ (the vertex r has exactly two neighbors in X) and $X - \{r, x\}, X - \{r, y\}$ are cliques, then Rule 4 is applicable with v as x and b as y.
- Under the condition that none of the previous steps worked, we find an induced 2-path. We refer the reader to the work by Etscheid and Mnich [EM18] for this step. It essentially utilizes the information gained about X and r to find any induced 2-path in linear time. The challenge in this part lies in achieving that runtime.

In case the one-way rules do not yield a "yes"-instance, another interesting theoretical bound on the computation of S is given: $|S| \leq 3k$. In total, the runtime performance of this algorithm can be summarized by $O(k \cdot (|V| + |E|))$.

In the upcoming list of reduction rules, we use the definition of a $\triangle_{\leq 2}$-block C in $G - S$. We define this as a triangle that fulfills $|C_{\text{ext}(G-S)}(C)| \leq 2$ and $N_G(C_{\text{int}(G-S)}(C)) \cap S = \emptyset$.

3.4.2 Reduction Rules

A list of found reduction rules for Max-Cut AEE follows under the stated circumstances from the previous sections. For each rule, a parameter k' is provided. The decision problem $(G, k)_{\text{AEE}}$ is then a "yes" instance if and only if $(G', k')_{\text{AEE}}$ is a "yes" instance, where G' is derived from applying a set of changes on G.

Some of the upcoming reduction rules are also applicable on signed graphs. Whenever that is the case, we explicitly state it. Note that in case of signed graphs, we assume that all

components in $G - S$ only contain negatives edges. If that is not the case, apply Corollary 3 by Crowston et al. [Cro+13] to achieve such an equivalent instance.

Reduction Rule 1. ([Suc+17], Rule A) Let G be a connected graph, and let v be a vertex of degree 1 in G. Then delete vertex v and set $k' = k - 1$.

Reduction Rule 2. ([CJM15], Rule 5) Apply if there exists a vertex $x \in V(G - S)$ and a set of vertices $X \subseteq V(G - S)$ such that $|X| > 1$, $G[X \cup \{x\}]$ is a clique, $G[X]$ is a connected component of $G - (S \cup \{x\})$, and no vertex in X is adjacent to any vertex in S. Remove all vertices in X, and set $k' = k - 1$ if $|X|$ is odd, otherwise $k' = k$.

Reduction Rule 3. ([CJM15], Rule 6) Apply if $s \in S, x \in V(G - S)$, and $X \subseteq V(G - S)$ exist such that $G[X \cup \{x\}]$ is a clique, $G[X \cup \{s\}]$ is a clique, $G[X]$ is a connected component of $G - (S \cup \{x\})$, and s is the only vertex in S adjacent to X. Remove all but one vertex of X, and set $k' = k - 1$ if $|X|$ is even, otherwise $k' = k$.

Reduction Rule 4. ([CJM15], Rule 7) Let X, Y be the vertex sets of two blocks in $G - S$ such that the sizes $|X|$ and $|Y|$ are odd, $\{z\} = X \cap Y, x \in X, y \in Y$, vertices in $\{x, z\}$ are the only vertices in X adjacent to a vertex in $G - X$, and the vertices in $\{y, z\}$ are the only vertices in Y adjacent to a vertex in $G - Y$. Remove all vertices in $(X \cup Y) \setminus \{x, y, z\}$ and add new vertices u, v alongside all the necessary edges for $\{x, y, z, u, v\}$ to form a clique.

Reduction Rule 5. ([Cro+13], Rule 8) Let (G, l) be a signed graph and C a block in $G - S$. If there is a $X \subseteq C_{\text{int}(G-S)}(C)$ such that $|X| > \frac{|V(C)|+|N_G(X)\cap S|}{2} \geq 1$, $N^+_{G,l}(x) \cap S = N^+_{G,l}(X) \cap S$ and $N^-_{G,l}(x) \cap S = N^-_{G,l}(X) \cap S$ for all $x \in X$, then delete two arbitrary vertices $x_1, x_2 \in X$ and set $k' = k$.

Reduction Rule 6. ([Cro+13], Rule 9) Let C be a block in $G - S$. If $|V(C)|$ is even and there exists a $X \subseteq C_{\text{int}(G-S)}(C)$ such that $|X| = \frac{|V(C)|}{2}$ and $N_G(X) \cap S = \emptyset$, then delete a vertex $x \in X$ and set $k' = k - 1$.

Reduction Rule 7. ([Cro+13], Rule 10) Let (G, l) be a signed graph and C be a block in $G - S$ with $V(C) = \{x, y, u\}$ such that $N_G(u) = \{x, y\}$. If the edge $\{x, y\}$ is a bridge in graph $G - \{u\}$, then delete C, add a new vertex z, positive edges $\{\{z, v\} : v \in N^+_{G-u,l}(\{x, y\})\}$, negative edges $\{\{z, v\} : v \in N^-_{G-u,l}(\{x, y\})\}$, and set $k' = k$. Otherwise, delete u and the edge $\{x, y\}$ and set $k' = k - 1$.

Reduction Rule 8. ([Cro+13], Rule 11) Let G be a signed graph and T a connected component of $G - S$ only adjacent to a vertex $s \in S$. Form a Vertex-Weighted Max-Cut instance on T by defining $w_0(x) = 1$ if $x \in N^+_{G,l}(s) \cap T$ is true ($w_0(x) = 0$ otherwise) and $w_1(y) = 1$ if $y \in N^-_{G,l}(s) \cap T$ is true ($w_1(y) = 0$ otherwise). Then solve the expression $\beta(G[V(T) \cup \{s\}]) = EE(G[V(T) \cup \{s\}]) + \frac{k}{4} = \frac{p}{4}$, delete T, and set $k' = k - p$.

Reduction Rule 9. ([EM18], Rule 9) Let C_1, C_2 be $\triangle_{\leq 2}$-blocks in $G - S$ which share a common vertex v. Make a block out of $V(C_1) \cup V(C_2)$ by adding all edges from the set $\{\{u, w\} \mid u \in V(C_1) \setminus \{v\}, w \in V(C_2) \setminus \{v\}\}$ to G. Set $k' = k$.

3.5 Spanning Tree Bound

Another approach is based on utilizing the spanning forest of a graph, as done by Madathil, Saurabh, and Zehavi [MSZ18]. For a given $k \in \mathbb{N}_0$, a Max-Cut of size $|V| - 1 + k$ is being searched for. This decision problem is denoted as Max-Cut AST (Max-Cut Above Spanning Tree), or shorter: $(G, k)_{\text{AST}}$. For sparse graphs, this bound is better than the one we have introduced in the previous section – the Edwards-Erdős bound. As mentioned, this is true when the average degree of the given graph is smaller than 3.

From here on, until the end of this section, let $S \subseteq V$ be any vertex set such that $G - S$ is a clique-cycle forest.

In the upcoming list of reduction rules, we use the definition of a $\triangle_{=2}$-block C in $G - S$. We define this as a triangle with $|C_{\text{ext}(G-S)}(C)| = 2$ and $N_G(C_{\text{int}(G-S)}(C)) \cap S = \emptyset$. Similarly, a $\pentagon$-block C in $G - S$ is a cycle of length 5 that satisfies $C_{\text{ext}(G-S)}(C) = \{x, y\}, \{x, y\} \notin E$, and $N_G(C_{\text{int}(G-S)}(C)) \cap S = \emptyset$.

3.5.1 Reduction Rules

A list of found reduction rules for Max-Cut AST follows under the stated circumstances from the previous section. In similar fashion like for the reduction rules for Max-Cut Above Edwards-Erdős, a parameter k' is provided by each rule. It holds that $(G, k)_{\text{AST}}$ is a "yes" instance if and only if $(G', k')_{\text{AST}}$ is a "yes" instance, where G' is derived from applying a set of changes on G.

Reduction Rule 10. ([MSZ18], Rule 6) Let C be an isolated odd cycle, K_2, or K_1 in $G-S$. Let there exist $s \in S$ such that s has either exactly one neighbor or exactly two neighbors in $V(C)$; and no other vertex of S has a neighbor in $V(C)$. Then delete $V(C)$ and set $k' = k-1$ if C is an odd cycle such that $|N(s) \cap V(C)| = 2$. Otherwise, $k' = k$.

Reduction Rule 11. ([MSZ18], Rule 7) Let C be a leaf-block of $G - S$ such that C is a clique and $N_G(C_{\text{int}(G-S)}(C)) \cap S = \emptyset$ holds. Then delete the vertices $C_{\text{int}(G-S)}(C)$ and set the parameter $k' = k - \frac{|C_{\text{int}(G-S)}(C)|^2}{4} - \frac{|C_{\text{int}(G-S)}(C)|}{2} + t$. Use $t = \frac{1}{4}$ if $|C_{\text{int}(G-S)}(S)|$ is odd; $t = 0$ otherwise.

Reduction Rule 12. ([MSZ18], Rule 8) Let C be a leaf-block of $G - S$ such that C is a cycle and $N_G(C_{\text{int}(G-S)}(C)) \cap S = \emptyset$. Then delete $C_{\text{int}(G-S)}(C)$ and set $k' = k - 1$ if C is an even cycle, and leave $k' = k$ if C is an odd cycle.

Reduction Rule 13. ([MSZ18], Rule 9) Let T be a connected component of $G - S$ such that $G[V(T)]$ is a path in $G - S$. Let $u, v \in V(T)$ be such that $deg_{G-S}(u) = deg_{G-S}(v) = 1$ holds. Also, there exists $s \in S$ such that $N_G(u) \cap S = N_G(v) \cap S = \{s\}$ and $N_G(x) \cap S = \emptyset$ for every $x \in V(T) \setminus \{u, v\}$. That is, $G[V(T) \cup \{s\}]$ is a cycle and no vertex in S except s has a neighbor in $V(T)$. Delete $V(T)$ and set $k' = k - 1$ if $|V(T)|$ is odd, and leave $k' = k$ otherwise.

Reduction Rule 14. ([MSZ18], Rule 10) Let $a'abcc'$ be an induced 4-path in $G - S$ such that $deg_G(a) = deg_G(b) = deg_G(c) = 2$ (Note that the degree restrictions imply that none of a, b, or c has a neighbor in S). Delete vertices a, b, c. Add new vertex v and new edges $\{a', v\}$ and $\{v, c'\}$. Set $k' = k$.

Reduction Rule 15. ([MSZ18], Rule 11) Apply when Rule 14 is no longer applicable. Let C be a $\pentagon$-block of $G-S$, $V(C) = \{u, a, b, v, c\}$, $E(C) = \{\{u, a\}, \{a, b\}, \{b, v\}, \{v, c\}, \{c, u\}\}$, and $C_{\text{ext}(G-S)}(C) = \{u, v\}$. Then delete vertices a and b, add edge $\{u, v\}$, and set $k' = k$.

Reduction Rule 16. ([MSZ18], Rule 12) Apply when Rule 14 is no longer applicable. Let A, B be two $\triangle_{=2}$-blocks of $G - S$. Let the vertices $a \in C_{\text{ext}(G-S)}(A)$ and $b \in C_{\text{ext}(G-S)}(B)$ be such that $deg_G(a) = deg_G(b) = 3$ holds. Let P be the unique path in $G - S$ of length at most three from a to b such that $deg_G(z) = 2$ for every internal vertex z of P. Then remove all edges in P, add edges $\{\{a, w\} \mid w \in N_G(b)\}$, and remove vertex b.

Reduction Rule 17. ([MSZ18], Rule 13) Apply to two $\triangle_{=2}$-blocks A, B that share a common vertex v with $N_G(v) \subseteq V(A) \cup V(B)$. Let $V(A) = \{a, a', v\}$ and $V(B) = \{b, b', v\}$ be such that $C_{\text{int}(G-S)}(A) = \{a\}$ and $C_{\text{int}(G-S)}(B) = \{b\}$. Then remove vertices a, b, v and introduce a new vertex v'. Add new edges $\{a', v'\}, \{b', v'\}, \{a', b'\}$. Set $k' = k$.

3.6 Other Rules

Some previous research has also studied reduction rules independent from any subgraphs (e.g., clique forests). Most of them are fairly simplistic and focus on very narrow cases. We have found such rules within the work by Prieto [Pri05] and mainly provide them for completeness. Also notice that Reduction Rule 21 is a generalization of Reduction Rule 20.

Reduction Rule 18. ([Pri05], Rule 1) For a graph $G = (V, E)$, let $u \in V$ be a vertex of degree 0. The problem $(G, k)_{\text{MC}}$ is a "yes"-instance if and only if $(G', k)_{\text{MC}}$ is a "yes"-instance, with G' being the graph resulting from the removal of vertex u from G.

Reduction Rule 19. ([Pri05], Rule 2) For a graph $G = (V, E)$, let $u \in V$ be a vertex of degree 1. The problem $(G, k)_{\text{MC}}$ is a "yes"-instance if and only if $(G', k - 1)_{\text{MC}}$ is a "yes"-instance, with G' being the graph resulting from the removal of vertex u from G.

Reduction Rule 20. ([Pri05], Rule 3) For a graph $G = (V, E)$, let $u, v, w \in V$ be such that $N_G(u) = \{v, w\}$ and $N_G(v) = \{u, w\}$ holds. The problem $(G, k)_{\text{MC}}$ is a "yes"-instance if and only if $(G', k - 2)_{\text{MC}}$ is a "yes"-instance, with G' being the graph resulting from the removal of vertices u and v from G.

Reduction Rule 21. ([Pri05], Rule 4) For a graph $G = (V, E)$, let u be a vertex of degree 2 in V and let $N_G(u) = \{x, y\}$ with $\{x, y\} \in E$. The problem $(G, k)_{\text{MC}}$ is a "yes"-instance if and only if $(G', k-2)_{\text{MC}}$ is a "yes"-instance, with G' being the graph resulting from the removal of edges $\{u, x\}$, $\{u, y\}$, and $\{x, y\}$ from G.

Reduction Rule 22. ([Pri05], Rule 5) For a graph $G = (V, E)$, let x, y, and z be three consecutive vertices of degree 2 in V. Let $N_G(x) = \{u, y\}$, $N_G(y) = \{x, z\}$, $N_G(z) = \{y, v\}$. The problem $(G, k)_{\text{MC}}$ is a "yes"-instance if and only if $(G', k-2)_{\text{MC}}$ is a "yes"-instance, with G' being the graph resulting from G after the removal of vertices $\{y, z\}$ and addition of edge $\{x, v\}$.

4 Reduction Rules

In essence, kernelization is a preprocessing technique characterized by replacing a larger input instance with a smaller one. The single condition for the replacing instance is that its solution is able yield a solution for the initial problem in polynomial time. In case of the Max-Cut problem, most kernelization rules from related works [CJM15] [EM18] [MSZ18] are actually not defined on the general graphs. For example, rules by Etscheid and Mnich [EM18] are defined on clique forest subgraphs. Moreover, some rules are straightforward and easy to understand, while others are more complicated and generalize others.

In the next section, we first provide a theoretical insight on how to efficiently determine the internal vertices of a clique. We use this in a later section to reduce the amount of candidates for the application of a reduction rule. This theoretical insight is then succeeded by an introduction of all newly developed reduction rules. For all such rules, we prove their correctness and outline implementation details where more elaboration is necessary. After that, we also generalize Reduction Rule 5 from previous works. This specifically includes the removal of the restriction that requires the identification of a clique forest subgraph. Direct application on the initially given graph is instead made possible. Finally, we also explain the existing inclusions among all mentioned reduction rules in our work. We show that only a few reduction rules are crucial to achieve all reducibility.

4.1 Determining Internal Vertices of a Clique

An observation follows that is used across our work to easily determine which vertices are internal and which ones are external in a clique.

Theorem 4. *Let X be a clique in a graph G. It holds that $v \in C_{\text{int}(G)}(X)$ if and only if $\forall w \in V(X) : deg_G(v) \leq deg_G(w)$.*

Proof. " $\Longrightarrow$ ": Since X is a clique, for each $v \in C_{\text{int}(G)}(X)$ and $w \in V(X)$, the relationships $deg_G(v) = |V(X)| - 1$ and $deg_G(w) \geq |V(X)| - 1$ hold. This then naturally implies $deg_G(v) \leq deg_G(w)$ for the same vertices.

" $\Longleftarrow$ ": Let any two vertices $v, w \in V(X)$ exist such that $deg_G(w) < deg_G(v)$ holds. Due to $deg_G(w) \geq |V(X)| - 1$, is $deg_G(v) > |V(X)| - 1$ and, therefore, is v an external vertex. □

The purpose of this theorem is found within the context of developing efficient algorithms later on. To this end, we introduce Algorithm 2 with running time $O(deg(v))$. Notice that Algorithm 2 does not require the considered subgraph $G[S]$ to be a clique. It holds that a vertex $v \in S$ is internal in $G[S]$ if and only if both of the conditions IsCliqueInternal(G, S, v) and IsClique(G, S) are true. See Algorithm 1 for our implementation of procedure IsClique.

This observation is useful because one can check if a vertex is internal first and then verify the clique property – calculation of this later attribute requires significantly more time. Furthermore, observe that an internal vertex of a clique is able to induce it wholly with $N_G(v) \cup \{v\}$.

Algorithm 1 Determines if subgraph $G[S]$ is a clique for a given subset $S \subseteq V(G)$.

```
1: function IsClique(G : Graph, S ⊆ V(G))                    ▷ Time O(|S|²)
2:     for all v ∈ S do
3:         for all w ∈ S do
4:             if ¬ AreAdjacent(G,v,w) then
5:                 return FALSE
       return TRUE
```

Algorithm 2 Determines if $v \in S$ is an internal vertex of clique $G[S]$.

```
1: function IsCliqueInternal(G : Graph, S ⊆ V(G), v ∈ S )    ▷ Time O(deg(v))
2:     for all w ∈ N_G(v) do
3:         if deg_G(w) < deg_G(v) then
4:             return FALSE
5:     return TRUE
```

4.2 New Reduction Rules

4.2.1 New Rule: Removal of Cliques of Size n with $|C_{\text{ext}(G)}| \leq \lceil \frac{n}{2} \rceil$

As will be evident in the evaluation chapter, the reducibility of induced subgraphs is significantly influenced by the number of external vertices it contains. This observation led to a new reduction rule for cliques. The idea is to classify all cliques in a graph instance according to their amount of external vertices. The rule can then wholly remove all cliques that have at most half of its vertices as external. In the next steps, we state a lemma followed by the reduction rule. The sole purpose of the lemma is to help us construct a proof for the reduction rule.

Lemma 5. *Let $q, a, b \in \mathbb{N}_0$ and $a + b = q$ hold. The product $a \cdot b$ is maximized if and only if $|a - b|$ is minimized (for even q that is $|a - b| = 0$, and for odd $|a - b| = 1$).*

Proof. Let $q \geq 0$. Define $p(a_1, b_1) = a_1 \cdot b_1$ for any integers $a_1, b_1 \geq 0$ with $a_1 + b_1 = q$. Consider increasing the value of one parameter by 1 and decreasing the other by the same amount. Without loss of generality, let $a_2 = a_1 - 1$ and $b_2 = b_1 + 1$. The solution induced by these new parameters then equals to $p(a_2, b_2) = p(a_1 - 1, b_1 + 1) = a_1 b_1 + a_1 - b_1 - 1$. Trivially, $a_2 + b_2 = q$ holds.

From $p(a_2, b_2) > p(a_1, b_1) \iff a_1 b_1 + a_1 - b_1 - 1 > a_1 \cdot b_1 \iff a_1 > b_1 + 1$ we then infer that, while $|a_1 - b_1| > 1$, one can always find integers a_2, b_2 such that $p(a_2, b_2) > p(a_1, b_1)$ is satisfied.

□

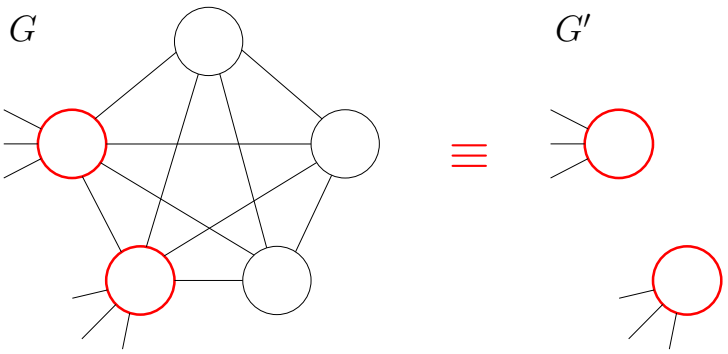

Figure 4.1: Application of Reduction Rule 23 on a clique. Vertices marked in red are external. The equality $\beta(G) = \beta(G') + \beta(K_5) = \beta(G') + 6$ holds.

Reduction Rule 23. Let G be a graph and $G[S]$, $S \subseteq V(G)$, a clique. If $|C_{\text{ext}(G)}(S)| \leq \left\lceil \frac{|S|}{2} \right\rceil$, then $\beta(G) = \beta(G') + \beta(K_{|S|})$ holds for graph $G' = (V \setminus C_{\text{int}(G)}(S), E \setminus E(G[S]))$.

Proof. Let $G = (V, E)$ be a graph and for the vertex subset $S \subseteq V$ let $G[S]$ be an induced subgraph that satisfies the conditions of Reduction Rule 23. Meaning, subgraph $G[S]$ is a clique, $n := |S|$, and $m := |C_{\text{ext}(G)}(S)| \leq \lceil \frac{n}{2} \rceil$. As stated in the definition of the Max-Cut problem, we search for a bipartition V_0, V_1 such that we maximize the number of edges between V_0 and V_1.

Consider any given 2-coloring of the vertices in $C_{\text{ext}(G)}(S)$. Put the vertices marked with color "0" in V_0 and those marked with "1" in V_1, then the assignment of $n - m$ vertices to a partition remains.

When solving the Max-Cut problem for $G[S]$ with a given coloring of $C_{\text{ext}(G)}(S)$, all vertices in $S \setminus C_{\text{ext}(G)}(S) = C_{\text{int}(G)}(S)$ can be treated "equivalently". Meaning, all of their properties are the same: $N_G(v) \cup \{v\} = N_G(w) \cup \{w\}, \forall v, w \in C_{\text{int}(G)}(S)$. No matter in what order we chose to insert them into V_0 or V_1, only the actual amount that we add to each is of importance.

Since $G[S]$ is a clique and every vertex is connected to every other one by an edge, the size of a maximum cut for $G[S]$ is given by maximizing $|V_0| \cdot |V_1|$. Using Lemma 5, we know that this is accomplished by minimizing $||V_0| - |V_1||$. It is therefore sufficient to place the vertices from $C_{\text{int}(G)}(S)$ in such a way that balances the size of V_0 and V_1. This balancing is satisfiable through the condition $m \leq \lceil \frac{n}{2} \rceil$: No matter how many vertices from $C_{\text{ext}(G)}(S)$ are placed in either V_0 or V_1, $|C_{\text{int}(G)}(S)| = |S| - |C_{\text{ext}(G)}(S)| = n - m \geq n - \frac{n}{2}$ are enough to balance V_0 and V_1.

This implies that, regardless on how we partition the vertices in $C_{\text{ext}(G)}(S)$, the size of the maximum cut of $G[S]$ remains the same. □

See Figure 4.1 for an example where Reduction Rule 23 is applied. We now proceed to show that it is possible to find all candidates for Reduction Rule 23 in polynomial time.

Theorem 6. *For a graph $G = (V, E)$, all applications of Reduction Rule 23 can be found in $O(|V| \cdot deg_{max}^2)$ time, where $deg_{max} := \max_{v \in V(G)} deg_G(v)$.*

Proof. Consider a clique $G[S], S \subseteq V$, of G that satisfies the conditions in Reduction Rule 23. Meaning, for $n := |S|$ and $m := |C_{\text{ext}(G)}(S)|$, $m \leq \lceil \frac{n}{2} \rceil$ holds. Also observe that any internal vertex u of a clique is able to induce all vertices of the clique through $\{u\} \cup N_G(u)$.

Observe that $I_{clique}(G) := \{G[S] \mid S = \{u\} \cup N_G(u)$ and IsClique(G, S), $u \in V(G)\}$ contains exactly those cliques in G with at least one internal vertex. This is also a superset of the candidates for Reduction Rule 23, which expects at least one internal vertex for every candidate. To determine if a member of $I_{clique}(G)$ allows the application of Reduction Rule 23 is then reduced to counting how many internal/external vertices it possesses. As we can calculate $I_{clique}(G)$ in polynomial time, we can also determine the members of $I_{clique}(G)$ on which Reduction Rule 23 is applicable in polynomial time.

Let us analyze Algorithm 3 to prove that it is possible to determine all candidates in $O(|V| \cdot deg_{max}^2)$ time. The correctness of the given algorithm follows from the observations in the previous paragraph. Consider the body of the main for-loop. Only a few statements are possibly in $\Omega(deg_G(v))$. One of them is the IsClique call on line 13. Although, its time performance actually lies in $O(|N_G(v)|^2) = O(deg_{max}^2)$ because $|S| = |N_G(v)| + 1$ holds and it is possible to determine if S forms a clique in time $O(|S|^2)$. Line 17 can be done in $O(|S|)$ by checking the size of $N_G(v)$ for every vertex $v \in S$. As S is a clique, a vertex $v \in S$ is internal if and only if $deg_G(v) = |S| - 1$ holds. □

Algorithm 3 Find all candidates for Reduction Rule 23.

```
1:  function IsClique(G : Graph, S ⊆ V(G))
2:      for all v ∈ S do
3:          for all w ∈ S do
4:              if ¬ AreAdjacent(G,v,w) then
5:                  return false
        return true
6:
7:  function FindAllRule23Candidates(G = (V, E) : Graph)
8:      candidates ← ∅
9:      for all v ∈ V do
10:         if visited[v] then
11:             continue
12:         S ← {v} ⋃ N_G(v)                                  ▷ Time O(|S|)
13:         if ¬IsCliqueInternal(G, S, v) ∨ ¬IsClique(G, S) then
14:             continue
15:         for all s ∈ S do
16:             visited[s] ← true
17:         if |C_ext(G)(S)| ≤ ⌈|S|/2⌉ then                   ▷ Time O(|S|)
18:             candidates.Append(v)
19:     return candidates
```

Note: In Algorithm 3, we use IsCliqueInternal on line 13 solely for practical performance purposes. It may preemptively determine that a vertex cannot be internal in a clique. This allows us to avoid the expensive IsClique call.

4.2.2 New Rule: Reduction of Induced 3-Paths

This is a small but extensive rule to reduce paths while still maintaining that the edges remain unweighted. As is going to be evident with Reduction Rule 28, the only way to further shorten the remaining paths is to permit weighted edges in the kernel.

Reduction Rule 24. Let $a'abb'$ be an induced 3-path in a graph G with $N_G(a) = \{a', b\}$ and $N_G(b) = \{a, b'\}$. Construct G' from G by adding a new edge $\{a', b'\}$ and deleting the vertices a and b. Then $\beta(G) = \beta(G') + 2$ holds.

Proof. Let $S = \{a', a, b, b'\}$. Two cases exist: Vertices a', b' are in the same partition of a maximum cut ($\delta_1(a') = 0$ and $\delta_1(b') = 0$), and a', b' are in different partitions ($\delta_2(a') = 0$ and $\delta_2(b') = 1$).

Consider the first case. If only one or both of a and b are in the opposing partition of a' and b', then the cut increases by 2. If both a and b are in the same partition as a' and b' then the size of the cut stays the same. Therefore, $\beta_{\delta_1}(G[S]) = 2$ holds in total.

In the second case, by coloring a with "1" and b with "0" we get that $\beta_{\delta_2}(G[S]) = 3$ holds. A larger increase is not possible as all edges of $G[S]$ are being accounted for.

Due to $\beta_{\delta_1}(G[S]) = \beta_{\delta_1}(G'[S]) + 2$ and $\beta_{\delta_2}(G[S]) = \beta_{\delta_2}(G'[S]) + 2$ being true, regardless of in which partitions a' and b' are placed in, the change by this reduction rule always implies the same constant difference between $\beta(G)$ and $\beta(G')$. We deduce that $\beta(G) = \beta(G') + 2$ holds. □

See Figure 4.2 for a visual representation of the reduction rule.

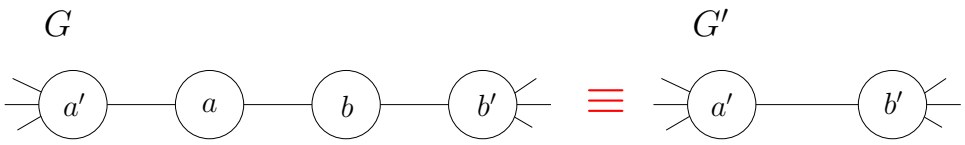

Figure 4.2: Application of Reduction Rule 24. The equality $\beta(G) = \beta(G') + 2$ holds.

4.2.3 New Rule: Clique Amount Increase

As seen with Reduction Rule 23, it is possible to remove a large variety of cliques in the graph. Within a later section, another rule reliant on cliques is introduced (see Reduction Rule $5^+{}_{w=1}$). sDue to this presence of reduction rules that need cliques, we have further expanded onto this by developing a rule that is able to form additional cliques in the graph.

Reduction Rule 25. Let G be a graph and let $G[S], S \subseteq V(G)$, be a subgraph in G such that it can be transformed into a clique with the addition of a single edge e' between two internal vertices. Define $G' = G + e'$. If the value of $|S|$ is odd or $|C_{\text{int}(G)}(S)| > 2$, then $\beta(G) = \beta(G')$ holds.

Proof. Our goal is to show that for all 2-colorings δ of $C_{\text{ext}(G)}(S)$, a maximum cut of G exists such that the endpoints of e' are in the same partition. To do so, consider any 2-coloring δ of $C_{\text{ext}(G)}(S)$. We show that $\beta_\delta(G[S])$ remains the same – whether e' exists or not. As this is being shown for any 2-coloring δ and no edges in $G[C_{\text{ext}(G)}(S)]$ are changed, Lemma 2 and 3 imply that $\beta(G)$ then also remains the same.

Let δ be any 2-coloring of $C_{\text{ext}(G)}(S)$. We proceed to create a bipartition V_0, V_1 that describes a maximum cut. Meaning, we maximize $E(V_0, V_1)$.

Initially, set $V_c = \{x \in C_{\text{ext}(G)}(S) \mid \delta(x) = c\}$ and define $z_c := |V_c|$, both for $c \in \{0, 1\}$. Without loss of generality, assume $z_0 \leq z_1$. Next, insert both endpoints of e' into V_0. Notice that the value of $E(V_1, V_2)$ is now equal to $|V_1| \cdot |V_2|$. This product always describes a maximum cut when maximized – one cannot insert any more edges between V_0 and V_1. We distinguish between two cases. First case is $z_0 < z_1$ and second is $z_0 = z_1$.

Consider the first case. The addition of both endpoints of e' into V_0 contributes towards minimizing $||V_0| - |V_1||$ – which is the goal when desiring to maximize $|V_0| \cdot |V_1|$ (see Lemma 5). As all other vertices of $C_{\text{int}(G)}(S)$ can be freely distributed, a way to achieve the maximum cut without the edge e' exists. We accomplish that by distributing the vertices of $C_{\text{int}(G)}(S)$ in a way that minimizes $||V_0| - |V_1||$. This can be done by greedily taking the vertices from $C_{\text{int}(G)}(S)$ and inserting them into the less populated partition.

Next, the case $z_0 = z_1$ is considered. As both endpoints of e' are inserted into V_0, the state $|V_0| = |V_1| + 2$ arises. To balance this, at least one more internal vertex is required. Such a vertex necessarily exists when $|S|$ is odd or $|C_{\text{int}(G)}(S)| > 2$. Therefore, one can analogously proceed as in the first case by greedily distributing each vertex of $C_{\text{int}(G)}(S)$ into the less populated partition. □

See Figure 4.3 for a visual representation of the reduction rule.

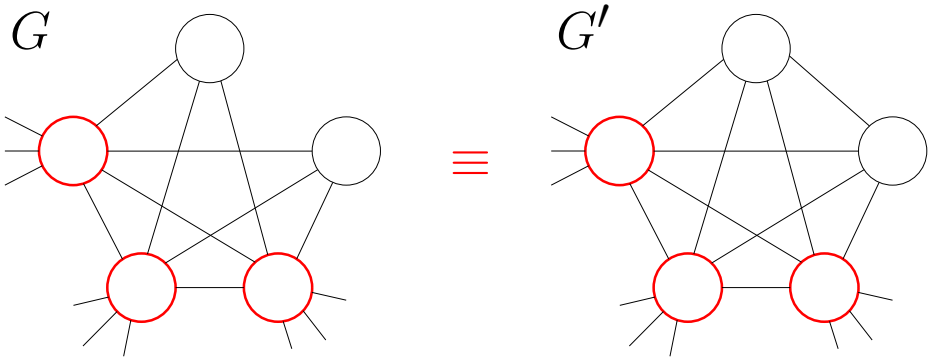

Figure 4.3: Application of Reduction Rule 25 on a subgraph. Vertices marked in red are external. The equality $\beta(G) = \beta(G')$ holds.

4.2.4 New Rule: Removal of an Edge in Cliques

As it is possible that some cliques are irreducible by any rules known to us, we also want to apply a reversed application of Reduction Rule 25. This does not reduce the vertex set nor,

as our experiments indicate, cause further possible applications of other rules. However, it is able to counteract unnecessary additions of edges by Reduction Rule 25 and possibly remove certain edges from the graph.

Reduction Rule 26. Let G be a graph and let $G[S]$, $S \subseteq V(G)$, be a clique in G. If the value of $|V|$ is odd or $C_{\text{int}(G)}(S) > 2$, an edge between two vertices of $C_{\text{int}(G)}(S)$ is removable. That is, $\beta(G) = \beta(G')$ for $G' = (V, E \setminus \{e\})$, $e \in E(G[C_{\text{int}(G)}(S)])$ holds.

Proof. Follows from proof of Reduction Rule 25. □

4.2.5 New Rule: Common Clique in Adjacency

The following reduction rule is closely related to the upcoming generalization of Reduction Rule 5 (see Section 4.3). It is able to further reduce the case where $|X| = |N_G(X)|$ is true for a clique $G[X]$ of G. In comparison, the generalization of Reduction Rule 5 is able to handle the case $|X| > |N_G(X)|$. Due to the degree by which these rules are similar, they are also merged together in our implementation, as the techniques to handle both are the same.

Reduction Rule 27. Let X be the vertex set of any clique in a graph G and let the conditions $|X| = |N_G(X)| \geq 1$ and $N_G(X) = N_G(x) \setminus X$ for all $x \in X$ be true. Create G' from G by removing an arbitrary vertex of X. The equality $\beta(G) = \beta(G') + |X|$ then holds.

Proof. Let $S := X \cup N_G(X)$ and observe that $C_{\text{ext}(G)}(S) \subseteq N_G(X)$ – the removal of the vertices in $N_G(X)$ disconnects X from the remainder of the graph. We proceed to show a way to determine a maximum cut of $G[S]$ for all possible colorings of $N_G(X)$. This is then used as an intermediate step to prove the validity of the reduction rule.

Let δ be any 2-coloring of the vertices in $N_G(X)$. Set $V_c = \{x \in C_{\text{ext}(G)}(S) \mid \delta(x) = c\}$ and define $z_c := |V_c|$, both for $c \in \{0, 1\}$. We then distribute the vertices of X among V_0 and V_1 in order to create a bipartition of all vertices in S. Furthermore, we want to do this in a way that maximizes $E(V_0, V_1)$. Notice that every vertex in X is connected to all other vertices in S. We can therefore express the size of a cut with $p(c_0, c_1) = c_0z_1 + c_1z_0 + c_0c_1 + |E(V_0, V_1)|$, where c_0 and c_1 denote the amount of vertices within X that we want to insert into partition V_0 and V_1, respectively. This can be equivalently expressed as the following function $p(c_0, c_1) = (z_0 + c_0) \cdot (z_1 + c_1) - z_0z_1 + |E(V_0, V_1)|$. Observe that the crux of the problem is then to maximize the expression $(z_0+c_0)\cdot(z_1+c_1)$ as all other terms are constant. Finally, use Lemma 5 to see that $(z_0 + c_0) \cdot (z_1 + c_1)$ is maximized when $||z_0 + c_0| - |z_1 + c_1||$ is minimized.

Due to $|X| = |N_G(X)|$, it is always possible to distribute the vertices of X in a way that achieves $|z_0 + c_0| = |z_1 + c_1| = |X|$, which then maximizes $p(c_0, c_1)$. Let us accordingly distribute the vertices of X among V_0 and V_1. Subsequently, no matter from which partition we remove a single vertex of X, the size of a maximum cut of G will change by $-|X|$. To show this, without loss of generality, let that one removed vertex be from the partition V_0. Then $|X| + |N_G(X)|$ is odd and $||z_0 + (c_0 - 1)| - |z_1 + c_1|| = 1$. Meaning, the value of $p(c_0 - 1, c_1) = p(c_0, c_1) - |X|$ describes the size of a maximum cut after that one vertex is removed. See Lemma 5 again in support of this last step. □

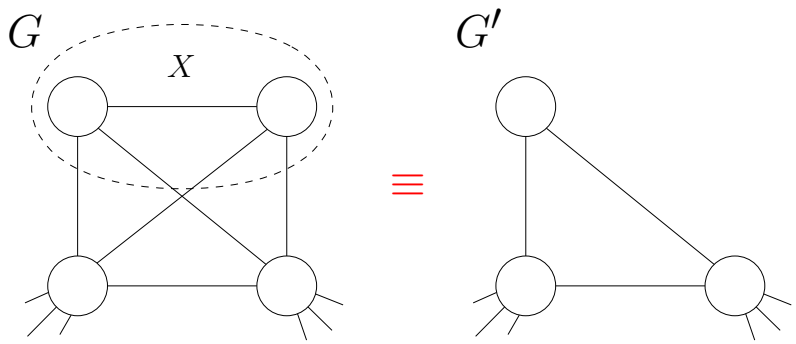

Figure 4.4: Application of Reduction Rule 27. The equality $\beta(G) = \beta(G') + |X| = \beta(G') + 2$ holds.

See Figure 4.4 for a visual representation of the reduction rule. Next, we discuss an algorithm to calculate all applications in linear time.

4.2.5.1 Linear Time Algorithm to Identify Candidates

The following algorithm is inspired by a similar approach as utilized by Etscheid and Mnich [EM18] for Reduction Rule 5. Let $G = (V, E)$ be a graph. In this section, we show an algorithm that identifies all candidates of Reduction Rule 27 in linear time.

First presented is a linear time algorithm that orders the adjacencies of all vertices. That is, for every vertex $v \in V(G)$, the vertices in $N_G(v)$ are sorted according to a numeric identifier assigned to every vertex. To accomplish that, create an auxiliary array of empty lists of size $|V|$. Traverse then the vertices $w \in N_G(v)$ for every vertex $v \in V(G)$ and insert each pair (v, w) in a list identified by indexing the auxiliary array with w. After that is done, iterate once over the array from the lowest identifier to the highest and recreate the graph with sorted adjacencies. In total, this process takes $O(|V| + |E|)$ time.

For any clique X of G that satisfies Reduction Rule 27, the fulfillment of the condition $\forall x_1, x_2 \in V(X) : N_G(x_1) \cup \{x_1\} = N_G(x_2) \cup \{x_2\}$ is required. This can be utilized to identify the applicability of tries [Fre60][De 59] in order to partition all vertices accordingly.

A trie supports two operations, Insert(key,val) and Retrieve(key). The key parameter is an array of integers and val is a single integer. Function Retrieve returns all inserted values by Insert that have the same key.

For vertex $v \in V(G)$, we use the ordered set $N_G(v) \cup \{v\}$ as key and v as the val parameter. Notice that $N_G(v)$ is already sorted. The key $N_G(v) \cup \{v\}$ can be then computed through an insertion of v in the sequence $N_G(v)$ in time $O(|N_G(v)|)$.

After Insert($N_G(v) \cup \{v\}$,v) is done for every vertex $v \in V(G)$, each trie leaf contains all vertices that satisfy the desired condition of Reduction Rule 27. Meaning, for every pair of vertices x_1, x_2 of a trie leaf, the condition $N_G(x_1) \cup \{x_1\} = N_G(x_2) \cup \{x_2\}$ holds. All that remains for us to verify is if the vertex set X of each leaf is a clique. This can be accomplished in $O(|E(X)|)$. As each such vertex set X has to be considered exactly once and the graph is fully partitioned, the summarized time complexity until this point is in $O(|V| + |E|)$.

Lastly, we outline that $|N_G(X)|$ equals to $deg(x) - |X|$ for $x \in X$. This is the last information that is required to check if $|X| > max\{N_G(X)|, 1\}$ holds, which concludes all requirements we need to check for the satisfiability of Reduciton Rule 27.

In Chapter 6, we then also describe a timestamping system that assists the above procedure in not having to repeatedly check the same structures after any amount of vertices and edges are added or removed from G. However, in those later applicability checks, we disregard sorting the adjacencies of all vertices in linear time again. Rather we simply use a comparison based sort on the adjacencies.

4.2.6 New Rule: Weighted Path Compression

This is the only reduction rule addressed by us whose application leads to a Weighted Max-Cut problem. We introduce it to later support its importance through experiments. It will be shown that it is crucial to permit weighted edges in the results in order to achieve small kernels.

Reduction Rule 28. Let G be a graph, $w : E \to \mathbb{Z}$ a weight function, and aba' any 2-path with $N_G(b) = \{a, a'\}$. Let e_1 be the edge between vertex a and b; let e_2 be the one between b and a'. Construct G' from G by deleting vertex b and adding a new edge $\{a, a'\}$ with $w'(\{a, a'\}) = \max\{w(e_1), w(e_2)\} - \max\{0, w(e_1) + w(e_2)\}$. Then the equality $\beta(G, w) = \beta(G', w) + \max\{0, w(e_1) + w(e_2)\}$ holds.

Proof. Consider the bipartition $V_0 \cup V_1 = V(G)$ of a maximum cut of G, and let the 2-path aba' satisfy the conditions of the rule. Meaning, $N_G(b) = \{a, a'\}$. Consider two cases: The endpoints of the path are in the same partition of a maximum cut ($a, a' \in V_0$), and the endpoints are in different partitions ($a \in V_0, a' \in V_1$). Then, depending on where b is placed, the change on the size of a maximum cut can be identified. We always place b in such a way that maximizes the increase by that change.

1. Case $a, a' \in V_0$. Depending on if $b \in V_0$ or $b \in V_1$, the 2-path contributes 0 or $w(e_1) + w(e_2)$ to the size of a maximum cut, respectively.

2. Case $a \in V_0, a' \in V_1$. Depending on if $b \in V_0$ or $b \in V_1$, the 2-path contributes $w(e_2)$ or $w(e_1)$ to the size of a maximum cut.

To handle the first outlined case, we initially assume that the endpoints a and a' are placed in the same partition within a maximum cut. Therefore, by considering both placements of b, an increase of $\max\{0, w(e_1) + w(e_2)\}$ is present on the size of a maximum cut. Our approach then consists of adding that value outright to the size of the maximum cut and subtract the same value from the edge weight between a and a'. Then, when a and a' are instead placed in different partitions of a maximum cut, we nullify the initially made assumption through the added subtraction on the weight of the edge.

In the second case, we add $\max\{w(e_1), w(e_2)\}$ to the edge weight between a and a'. This accounts for best placement of b when a and a' are in different partitions of a cut. With this, we can now summarize everything as $\beta(G, w) = \beta(G', w') + \max\{0, w(e_1) + w(e_2)\}$ with $w'(\{a, a'\}) = \max\{w(e_1), w(e_2)\} - \max\{0, w(e_1) + w(e_2)\}$ and $w'(e) = w(e)$ for all other edges $e \in E(G) \setminus \{\{a, b\}, \{b, a'\}\}$. □

See Figure 4.5 for a visual representation of the reduction rule.

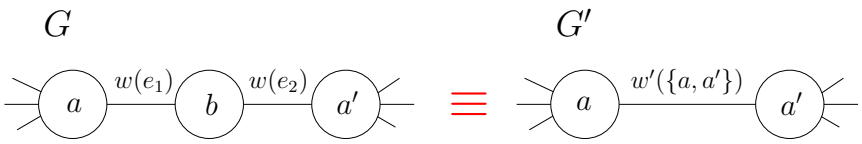

Figure 4.5: Application of Reduction Rule 28 on a 2-path aba' with $N_G(b) = \{a, a'\}$. The equations $w'(\{a, a'\}) = \max\{w(e_1), w(e_2)\} - \max\{0, w(e_1) + w(e_2)\}$ and $\beta(G, w) = \beta(G', w') + \max\{0, w(e_1) + w(e_2)\}$ hold.

4.3 Generalization of Reduction Rule 5

This rule is related to Reduction Rule 5 (Rule 8 by Crowston et al. [Cro+13]). A version only supporting the unweighted case was also previously introduced by Crowston, Jones, and Mnich [CJM15]. We restate Reduction Rule 5 for reference.

Reduction Rule 5. ([Cro+13], Rule 8) Let (G, l) be a signed graph and C a block in $G-S$. If there is a $X \subseteq C_{\text{int}(G-S)}(C)$ such that $|X| > \frac{|V(C)|+|N_G(X)\cap S|}{2} \geq 1$, $N^+_{G,l}(x) \cap S = N^+_{G,l}(X) \cap S$ and $N^-_{G,l}(x) \cap S = N^-_{G,l}(X) \cap S$ for all $x \in X$, then delete two arbitrary vertices $x_1, x_2 \in X$ and set $k' = k$.

In this section, we show that this rule has an equivalence to another rule that has to consider a significantly smaller set of parameters. Furthermore, this new redefinition of the rule will allow us to determine all occurrences of the rule, regardless of S and the clique forest $G - S$. Meaning, after the exhaustive application of this equivalent rule, one can compute any S and Reduction Rule 5 will not be further applicable on $G - S$.

We proceed to prove this redefinition only for when all edges are unweighted. All our findings from this point on can also be extended to the signed variant of the Max-Cut problem. Moreover, our actual implementation for this rule does also include the capability of handling signed graphs. We have left these steps out as we believe that they solely rely on getting the technicalities right. The unweighted and signed version of the reduction rule follow, respectively.

Reduction Rule ${5^+}_{w=1}$. Let X be the vertex set of a clique in G with $|X| > \max\{|N_G(X)|, 1\}$ and $N_G(X) = N_G(x) \setminus X$ for all $x \in X$. Construct the graph G' by deleting two arbitrary vertices $x_1, x_2 \in X$ from G. The equality $\beta(G) = \beta(G') + |N_G(x_1)|$ holds.

Reduction Rule 5^+. Let X be the vertex set of a clique in a signed graph (G, l) such that $\forall e \in E(X) : l(x) =$ "$-$". Let also $|X| > \max\{|N_G(X)|, 1\}$, $N^+_{G,l}(X) = N^+_{G,l}(x) \setminus X$, and $N^-_{G,l}(X) = N^-_{G,l}(x) \setminus X$ hold for all $x \in X$. Construct the graph G' by deleting two arbitrary vertices $x_1, x_2 \in X$ from G. The equality $\beta(G) = \beta(G') + |N_G(x_1)|$ holds.

4.3.1 Proof of Equivalence

We proceed to show the validity of Reduction Rule ${5^+}_{w=1}$. First, a theorem is introduced to outline our goal in this section: We want to show that every application of Reduction Rule 5 can be be also covered by Reduction Rule ${5^+}_{w=1}$.

Theorem 7. *Let S be any vertex set such that $G-S$ is a clique forest. Let C be a block in $G-S$ with $X \subseteq C_{\text{int}(G-S)}(C)$ such that $|X| > \frac{|V(C)|+|N_G(X)\cap S|}{2} \geq 1$, and $N_G(x) \cap S = N_G(X) \cap S$ for all $x \in X$. Therefore, conditions of Reduction Rule 5 are satisfied. Then clique X also satisfies the requirements of Reduction Rule ${5^+}_{w=1}$.*

A proof for this theorem follows through multiple lemmas and observations. First, a lemma is given to more precisely specify the clique forests on which it is sufficient to apply Reduction Rule 5. To do so, we show that for every possible application of Reduction Rule 5, we are able to reduce the requirement of a clique forest to a single clique. The rule is then applicable on this clique in the same way it is on the clique forest, without having any effects on the outcome of the application, nor its capabilities in reducing the graph.

Lemma 8. *Let $S_1 \subseteq V(G)$ be such that $G - S_1$ is a clique forest. Furthermore, let a block C in $G - S_1$ and a vertex set $X \subseteq C_{\text{int}(G-S_1)}(C)$ exist that satisfy Reduction Rule 5. Meaning, the conditions $|X| > \frac{|V(C)|+|N_G(X)\cap S_1|}{2} \geq 1$, and $N_G(x) \cap S_1 = N_G(X) \cap S_1$ for all $x \in X$ are satisfied. Then the same C and X can be selected in $G - S_2$ for $S_2 = V(G - C)$ with Reduction Rule 5 remaining applicable.*

Proof. First, we are required to show that $G - S_2$ is a clique forest for Reduction Rule 5 to be applicable. This is trivial as $G - S_2 = G - (G - C) = C$ is a clique, and therefore a clique forest too. Observe that $X \subseteq C_{\text{int}(G-S_1)}(G[C]) \subseteq C = C_{\text{int}(G-S_2)}(G[C])$ holds. This implies that the same C and X remain selectable in this new clique forest $G - S_2$.

The last thing we need to consider is a change in $N_G(X) \cap S_2$ compared to $N_G(X) \cap S_1$. From $C \subseteq G - S_1$ the following holds: $S_1 = G - (G - S_1) \subseteq G - C = S_2$. This implies that $S_1 \subseteq S_2$ and $N_G(X) \cap S_1 \subseteq N_G(X) \cap S_2$ is true. It therefore suffices to look at vertices $w \in (N_G(X) \cap S_2) \setminus (N_G(X) \cap S_1)$ when investigating the differences.

- Membership $w \in G - S_1$ has to hold. If not, then $w \in S_1$ would imply a contradiction. From $w \in N_G(X) \cap S_2$, it follows that $w \in N_G(X)$ and $w \in S_2$ hold. Due to $w \in S_1$, the membership $w \in N_G(X) \cap S_1$ also applies. This cannot be true because w is chosen from $(N_G(X) \cap S_2) \setminus (N_G(X) \cap S_1)$.
- But, also $w \notin C$ as $S_2 = V(G - C)$ holds.
- In total, this implies $w \notin C = G - S_2 \subseteq G - S_1 \ni w$.

These vertices then include those in the clique forest $G - S_1$, but not part of the clique C. Such a vertex w can have at most one edge incident to C, as otherwise $C \cup \{w\}$ would form a block (and not C). Alas, due to $|X| > 1$ and the requirement on w to be connected with all vertices of $X \subseteq C$, it cannot be that w has only a single edge connected to.

This means that no such vertices $w \in (N_G(X) \cap S_2) \setminus (N_G(X) \cap S_1)$ can exist and that $N_G(X) \cap S_2 = N_G(X) \cap S_1$ holds. As the size of C and X do not change, and due to the

latest observation on the neighboring set of X, Reduction Rule 5 remains applicable in the same way.

□

With Lemma 8 we now acquire the possibility of fully disregarding the need to initially select an S for $G - S$ to be a clique forest. Instead, we select any clique C and define the marked vertex set $S = V(G - C)$. We proceed to further simplify the rule.

Lemma 9. *Let C be a clique in a graph G. Furthermore, let $X \subseteq V(C)$ and $S = V(G - C)$ such that $|X| > \frac{|V(C)|+|N_G(X)\cap S|}{2} \geq 1$ and $N_G(x) \cap S = N_G(X) \cap S$ for all $x \in X$. Therefore, conditions of Reduction Rule 5 are satisfied. Then $|X| > \frac{|X|+|N_G(X)\cap S_X|}{2} \geq 1$ for $S_X = V(G-X)$ holds.*

Proof. Consider the two cases: $X \subset V(C)$ and $V(C) = X$. The case $V(C) = X$ is trivial. Therefore, assume $X \subset V(C)$. Since both C and X are cliques, $w \in V(C - X)$ implies both $w \in N_G(X)$ and $w \in N_G(x)$ for all $x \in X$. Due to this and $S_X = V(C - X) \cup S$ being true, for any such $w \in V(C-X)$ the equation $N_G(X) \cap (S \cup \{w\}) = N_G(x) \cap (S \cup \{w\})$ holds for all $x \in X$. These observation can be then summarized into $N_G(X) \cap S_X = N_G(x) \cap S_X$ for all $x \in X$ being satisfied.

With this, it is now possible to infer the following:

$$|X| > \frac{|C| + |N_G(X) \cap S|}{2} = \frac{(|X| + |V(C - X)|) + (|N_G(X) \cap S_X| - |V(C - X)|)}{2}$$

$$= \frac{|X| + |N_G(X) \cap S_X|}{2} > 1.$$

□

Lemma 8 and 9 now let us select any clique X in G, and by defining $S = V(G - X)$, we are able to cover all cases on which Reduction Rule 5 is applicable. Furthermore, we can reform the inequality:

$$|X| > \frac{|X| + |N_G(X) \cap S|}{2} > 1 \iff 2|X| > |X| + |N_G(X) \cap S| > 2$$

$$\iff |X| > |N_G(X) \cap S| > 2 - |X| \iff |X| > |N_G(X)| > 2 - |X|.$$

Last transformation holds due to $S = V(G - X)$. Moreover, the inequality can be also written as $|X| > |N_G(X)| \wedge |X| > 1$ or $|X| > \max\{N_G(X), 1\}$. This all together now concludes the proof of Theorem 7. One last thing that remains to be shown in Reduction Rule ${5^+}_{w=1}$ is the correctness of the change in the cut: $\beta(G) = \beta(G') + |N_G(X)|$.

Already proven in previous work is that Reduction Rule 5 inflicts no change on the parameter k in the problem $(G, k)_{\text{AEE}}$. That is, $(G, k)_{\text{AEE}}$ is satisfiable if and only if $(G', k)_{\text{AEE}}$ is satisfiable. This knowledge helps us to deduce the change in cut for problem instance $(G, k)_{\text{MC}}$ when applying the rule.

We know that for some value k, the equalities $\beta(G) = EE(G) + k$ and $\beta(G') = EE(G') + k$ hold. By solving $\beta(G) - \beta(G')$, we can deduce the change in the size of a maximum cut

between G and G'. The order of vertex removal is crucial in the following calculation. Without loss of generality, let x_1 be the vertex that is removed first by the reduction rule and let x_2 be the second one. We now proceed with the expansion of $\beta(G) - \beta(G')$:

$$\beta(G) - \beta(G') = EE(G) - EE(G') = \frac{|E(G)|}{2} + \frac{|V(G)| - 1}{4} - \left(\frac{|E(G')|}{2} + \frac{|V(G')| - 1}{4}\right).$$

Observe that we created G' by removing $x_1, x_2 \in X$ from G. Therefore, the equations $|E(G')| = |E(G)| - |N_G(x_1)| - (|N_G(x_2)| - 1)$ and $|V(G')| = |V(G)| - 2$ hold. We subtract one from $|N_G(x_2)|$ because the removal of x_1 already accounts for the edge $\{x_1, x_2\}$. The above equation further equals to the following:

$$\frac{|E(G)|}{2} + \frac{|V(G)| - 1}{4} - \left(\frac{|E(G)| - |N_G(x_1)| + (|N_G(x_2)| - 1)}{2} - \frac{(|V(G)| - 2) - 1}{4}\right)$$

$$= \frac{(|V(G)| - 1) - |V(G)| + 2 + 1}{4} - \frac{-|N_G(x_1)| - (|N_G(x_2)| - 1)}{2}.$$

Lastly, we utilize that $N_G(x_1) = N_G(x_2)$ holds to further equate the expression to:

$$\frac{2}{4} - \frac{-|N_G(x_1)| - |N_G(x_1)| + 1}{2} = \frac{2}{4} - \frac{-2|N_G(x_1)| + 1}{2} = \frac{2}{4} - \frac{1}{2} + |N_G(x_1)| = |N_G(x_1)|.$$

With this, we have now wholly proven the correctness of Reduction Rule $5^+{}_{w=1}$. An almost equivalent approach as in Section 4.2.5.1 can be used to find all candidates of this reduction rule in linear time.

4.4 Inclusions Among Reduction Rules

In this section, we explain existing inclusions among the mentioned reduction rules in our work. Note that, even if the effects of a reduction rule can be fully encompassed by another rule, it still possesses potential utility for other rules. See Figure 4.6. Although, such cases are not present among the following inclusions.

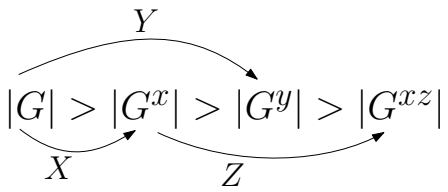

Figure 4.6: Depiction of an exemplary inclusion hierarchy among reduction rules. Even though rule X's effects are included in Y, a rule Z might potentially exist that could build upon the effects of X to lead to even better results.

4.4.1 Reduction Rule 1 and 19 ⊆ Reduction Rule 23

Both of these rules describe the removal of a single vertex that is connected to the remainder of the graph by one edge. For a given graph G, let $u \in V(G)$ be such a removable vertex, and let $\{u, x\} \in E(G), x \in V(G)$, be the corresponding edge. Consider now the subgraph $G[\{u, x\}]$. It is a clique and Reduction Rule 23 is applicable on it. Its application yields the same effects as Reduction Rule 1 and 19.

4.4.2 Reduction Rule 2 and 11 ⊆ Reduction Rule 23

These rules are addressing the reduction of a clique with at most one external vertex. Therefore, Reduction Rule 23 can optimally handle this case.

4.4.3 Reduction Rule 3 ⊆ Reduction Rule 27 and $5^+{}_{w=1}$

Consider a clique X of G on which the Reduction Rule 3 is applicable. Therefore, $|X| > 1$ holds and only two vertices $x_1 \neq x_2 \in V(G - X)$ are connected with X. They form the following cliques: $X \cup \{x_1\}$ and $X \cup \{x_2\}$. Meaning, X is also a clique. With this, and from $N_G(X) = \{x_1, x_2\}$, it is possible to deduce that this reduction rule is a special case of Reduction Rule 27 and $5^+{}_{w=1}$.

4.4.4 Reduction Rule 4 and 9 ⊆ Reduction Rule 23

Notice that Reduction Rule 4 is a generalization of Reduction Rule 9: The $\triangle_{\leq 2}$-blocks are of odd size and contain at most two external vertices. Therefore, they fully fulfill the requirements of blocks X and Y in Reduction Rule 4. Furthermore, the application of both rules on two $\triangle_{\leq 2}$-blocks also leads to the same result: A K_5 clique with at most 3 external vertices.

Consider now only Reduction Rule 4. Let X and Y be two odd sized blocks of $G - S$ with at most two external vertices, of which one is shared by both. Meaning, the two expressions $1 \leq |C_{\text{ext}(G)(X)}|, |C_{\text{ext}(G)(Y)}| \leq 2$ and $X \cap Y = \{z\}$ are true. The two blocks X and Y are now such that Reduction Rule 4 is applicable on them.

In any case, the application of the rule leads to a K_n clique with $\lceil \frac{n}{2} \rceil$ external vertices. Notice that the result must not always be a K_5 clique as the vertices x, y, and z may collapse into each-other. Also, even though the clique decreases in size with the collapse, so does the amount of external vertices. This means that Reduction Rule 23 remains applicable to this result, leaving only the external vertices x, y, and z still intact.

Observe that Reduction Rule 23 could have been also simply applied to X and Y directly, leading to the same result. Therefore, the same outcome is begin achieved without the need for Reduction Rule 4 (and 9).

4.4.5 Reduction Rule 6 ⊆ Reduction Rule 23

Similarly as in the previous case, this one can also be handled by Reduction Rule 23. Let G be a given graph and $S \subseteq V(G)$ be such that $G - S$ is a clique forest. Also, let the following

be given for the applicability of Reduction Rule 6: C a block in $G - S$ with $|V(C)|$ being even, and $X \subseteq C_{\text{int}(G-S)}(C)$ with $|X| = \frac{|V(C)|}{2}$.

From $N_G(X) \cap S = \emptyset$ it follows that $C_{\text{int}(G)}(C) = C_{\text{int}(G-S)}(C)$ holds. This is because the inclusion of vertices from S cannot turn members of $C_{\text{int}(G-S)}(C)$ to become external. This observation implies that clique C has at least half of its vertices as internal. Which, in turn, implies the applicability of Reduction Rule 23 on C. The only distinction remaining between those two rules is on the amount they remove. Reduction Rule 23 removes all edges of the clique, and all internal vertices. Reduction Rule 6 removes only a single internal vertex. This also turns $|V(C)|$ to an odd value, making further application of the same rule not viable. In any case, Reduction Rule 23 will also remove that single vertex, and more.

4.4.6 Reduction Rule 10, 12, and 13 ⊆ Reduction Rule 23 and 24

Let us consider a component-block (connected component and block) C of $G - S$ that satisfies Reduction Rule 10. Therefore, C is either an odd cycle, K_2, or K_1.

Let there be a vertex $s \in S$ such that $|N_G(s) \cap V(C)| \in \{1, 2\}$. Trivially, we may ignore the case of C being K_2 or K_1 as these reductions are equivalently covered by Reduction Rule 23. Therefore, it remains to consider C as a cycle.

According to Rule 10, such a C is removable. Notice that the component C is solely connected through s with the remaining components of the graph. In total, we have a cycle C in G with $|C_{\text{ext}(G)}(C)| \leq 2$.

To handle the case $|C_{\text{ext}(G)}(C)| \leq 1$, compress the cycle's edges with Reduction Rule 24 until 3 edges are left. As the cycle is of odd size and in each application the cycle is reduced by exactly 2 edges, this is possible. After this, simply apply Reduction Rule 23 to remove the resulting triangle.

The case $|C_{\text{ext}(G)}(C)| = 2$ can be similarly handled. Let $C_{\text{ext}(G)}(C) = \{x, y\}$. Instead of compressing the cycle's edges, compress the two path's edges on the cycle, going from x to y. A triangle with two external vertices is the results. Apply Reduction Rule 23 in this instance too.

For Reduction Rule 12 and 13 the same techniques can be used to show that Reduction Rule 24 and 23 cover them.

4.4.7 Reduction Rule 14 and 22 ⊆ Reduction Rule 24

Both of these rules perform the same set of reductions: They transform an induced 4-path to a path of length 2. Although, Reduction Rule 24 is able to reduce any induced 3-path a single edge. This then trivially implies that it is also able to reduce induced 4-paths in the same way Reduction Rule 14 and 22 are able to: By compressing a 3-path within the 4-path to a single edge.

4.4.8 Reduction Rule 15 ⊆ Reduction Rule 24

Consider a $\pentagon$-block $C = \{a, b, c, u, v\}$ of $G - S$ as defined within the reduction rule. Then, the application of Reduction Rule 15 on C and the application of Reduction Rule 24 on the 3-path $uabv$ lead to the same result.

4.4.9 Reduction Rule 16 and 17 ⊆ Reduction Rule 23

Both of these rules can be optimally handled by first fully removing the two $\triangle_{=2}$-blocks, followed by the removal of the isolated path left behind, if it exists. All of these effects are possible to handle Reduction Rule 23.

4.4.10 Reduction Rule 18, 20, and 21 ⊆ Reduction Rule 23

Trivially covered by Reduction Rule 23.

4.5 Scaled Reduction Rules

Additionally done by us is weighted scaling of two reduction rules. That is, we extend their applicability from an unweighted subgraph to a subgraph where all edges have the same weight $c \in \mathbb{R}$. We do this for Reduction Rule 23 and Reduction Rule 25.

For example, Reduction Rule 23 is explicitly defined on unweighted graphs. It removes cliques that have at most half of its vertices as external and establishes the relationship $\beta(G) = \beta(G') + \beta(K_{|S|})$, where G is the initially given graph and G' is the result of removing the clique $G[S]$ from G. If every edge in such a clique had the weight c, the relationship would actually equal to $\beta(G) = \beta(G') + c \cdot \beta(K_{|S|})$. This then implies the possibility of also defining reduction rules that are equivalently applicable on subgraphs with uniform weights.

While this outlined generalization might seem too specific to yield any benefit, it does have a significant impact on the instances of image segmentation – as seen later in the evaluation chapter. The definition of the two scaled reduction rules follows.

Reduction Rule 23$_{w=c}$. Let (G, w) be a weighted graph and let $G[S]$, $S \subseteq V(G)$, be a clique with $w(e) = c$ for every edge $e \in E(G[S])$ and some constant $c \in \mathbb{R}$. If $|C_{\text{ext}(G)}(S)| \leq \left\lceil \frac{|S|}{2} \right\rceil$, then $\beta(G, w) = \beta(G', w') + c \cdot \beta(K_{|S|})$ holds for graph $G' = (V \setminus C_{\text{int}(G)}(S), E \setminus E(G[S]))$ with weight function $w'(e) = w(e)$ for every $e \in E(G')$

Reduction Rule 25$_{w=c}$. Let (G, w) be a weighted graph and let $G[S], S \subseteq V(G)$, be a subgraph in G such that it can be transformed into a clique with the addition of a single weighted edge e' between two internal vertices. Furthermore, let $w(e) = c$ hold for every edge $e \in E(G[S])$ and some constant $c \in \mathbb{R}$. Define $G' = G + e'$, $w'(e) = c$, and $w'(e) = w(e)$ for every other $e \in E(G')$. If the value of $|S|$ is odd or $|C_{\text{int}(G)}(S)| > 2$, then $\beta(G, w) = \beta(G', w')$ holds.

4.6 Non-Utilized Reduction Rules

We have decided to exclude the evaluation of two reduction rules from previous works. Those are Reduction Rule 7 and 8. Reduction Rule 7 is not utilized because both of its cases can be covered by Reduction Rule 23 and our tests have shown that the bridge case (which no other rules can handle) yields no utility. Even more, the tested real-world instances only performed worse under its influence.

Reduction Rule 8 is not utilized because we regard that technique to be more relevant for algorithms solving the Max-Cut problem, not performing kernelization. This rule describes a way to solve a type of subgraphs containing a single external vertex. However, being able to determine a maximum cut of any subgraph with a single external vertex makes it possible to remove it. Meaning, each of them actually represent a Max-Cut instance by themselves. In the same spirit, one could have utilized previous works on how to solve a wide range of Max-Cut problems to specifically address such cases. Another reason for our decision is that Reduction Rule 8 seems to require a very specific structure of its applicability. A single vertex has to be connected to every vertex of a larger component, which is required to be of a structure that is a polynomial time solvable Vertex-Weighted Max-Cut instance. Moreover, it would have to consist of more than a single clique, as otherwise Reduction Rule 23 would be able to handle it.

5 Transformation Between Max-Cut Problem Variations

The introduced transformations in the upcoming subsections each represent a single approach on how to convert a Max-Cut instance into another. We settled on a way that seemed intuitively well for our needs and that is able to exhaustively process paths and edges.

5.1 Signed to Weighted

An Unweighted Max-Cut instance can be solved as a Signed Max-Cut problem by labeling all edges with "–". While this does not work the other way around, it is possible to represent a Signed Max-Cut problem as an instance of the Weighted Max-Cut problem. To do so, assign to every "+" edge the weight -1 and to every "–" edge the weight $+1$. Let (G, w) be the weighted graph resulting from this transformation. The size of a maximum cut for the Signed Max-Cut problem on (G, l) is then given by $\beta(G, l) = \beta(G, w) + |E^+(G, l)|$.

Proof. The Signed Max-Cut problem consists of finding an $S \subseteq V(G)$ such that the size of the cut as given by $\beta(G, l) = |E^-_{G,l}(S, V \setminus S)| + |E^+(G[S], l) \cup E^+(G[V \setminus S], l)|$ is maximized. The section where "–" edges are turned into edges with weight $+1$ does not affect the relation between $\beta(G, l)$ and $\beta(G, w)$ – both variations increase the cut under the same circumstances. Consider now the "+" edges. Each of them increases the size of the cut by one if its endpoints are placed into the same partition. Let us actually assign to each such edge the weight -1 for the Weighted Max-Cut problem. In total, this is being done for $|E^+(G, l)|$ edges. The Weighted Max-Cut problem now penalizes each edge whose endpoints are in different partitions by decreasing the size of the cut by one, and does nothing otherwise. In the same case, the Signed Max-Cut withheld an increase on the size of the cut by one, and increased it by one otherwise. Therefore, the maximization of the former case equivalently maximizes the later one. Both values are only offset by $|E^+(G, l)|$. This explains the relation $\beta(G, l) = \beta(G, w) + |E^+(G, l)|$ among the two variations of the Max-Cut problem. □

5.2 Weighted to Unweighted

In some limited scope, it is of use to transform weighted edges into unweighted "gadget" components. Meaning, for an edge with weight w, we expand the graph by $O(w)$ edges/vertices. This particularly applies to signed graphs, where the resulting conversion into a weighted instance is small in absolute value.

For a given graph G with edge weights $w : E \to \mathbb{Z}$, we want to compute an unweighted graph G' such that $\beta(G) = \beta(G') + z$ holds for some $z \in \mathbb{Z}$. We compute the required value of z incrementally. Initially, set $z = 0$. Our approach then iterates over all edges $e \in E(G)$ and differentiates between three cases in its conversion. Let $e = \{u, v\}$.

5.2.1 Case w(e) = 1

This is already an unweighted edge. Just add the same edge e to G'.

5.2.2 Case w(e) > 1

In this case, a single unweighted edge e is added to G' and the following steps are repeated $i = 1 \dots w(e) - 1$ times:

1. Create two vertices $m_{i,1}, m_{i,2}$;
2. Add edges $\{u, m_{i,1}\}, \{m_{i,1}, m_{i,2}\}, \{m_{i,2}, v\}$ to the graph G';
3. $z \leftarrow z - 2$.

These are valid as applying the path compression by Reduction Rule 24 completely removes the effects by this change., which also proves the equivalence of this conversion. See Figure 5.1 for an example and further elaboration.

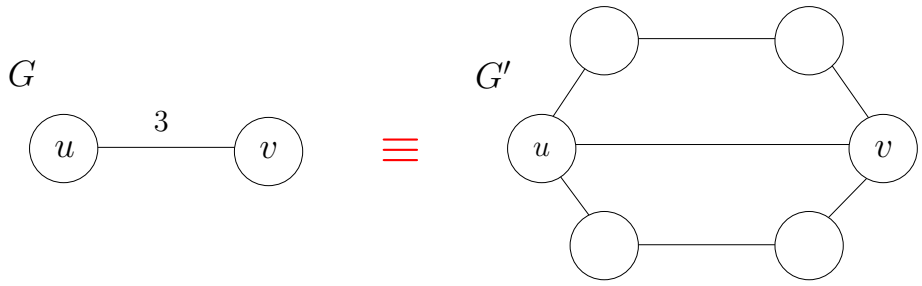

Figure 5.1: Transformation of an edge e with weight $w(e) = 3$ into an unweighted (sub)graph is presented. The equality $\beta(G) = \beta(G') - (w(e) - 1) \cdot 2 = \beta(G') - 4$ holds. To deduce this relationship, apply Reduction Rule 24 on the two unweighted 3-paths from u to v in G'. Summarize multiple edges into a single weighted edge.

5.2.3 Case w(e) < 0

Consider a 2-path with both edges having the weight $-w, w > 0$. The application of Reduction Rule 28 on this path yields a single edge with weight w. Therefore, in the case where $w(e) < 0$ holds, it suffices to create a new vertex m in G' and to form a 2-path umv with each edge having weight $-w(e)$. After this step, we then apply the previous two cases (5.2.1 and 5.2.2) on the two newly formed edges.

The creation of the 2-path inflicts the change $z \leftarrow z - 2|w(e)|$. See Figure 5.2 for an example.

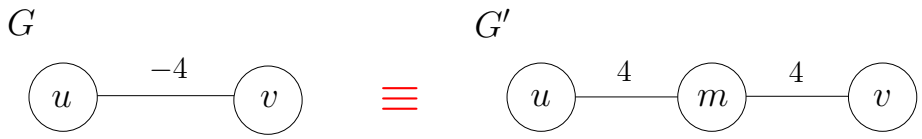

Figure 5.2: Transformation of an edge e with weight $w(e) = -4$ into a 2-path with positive weights is presented. The equality $\beta(G) = \beta(G') - 2|w(e)| = \beta(G') - 8$ holds. See Reduction Rule 28.

6 Implementation

With the introduction of techniques to convert between Max-Cut problem variations in the previous chapter, we can now complete the description of our implementation on everything outlined until this point. Note that for the reduction rules throughout the previous sections, where necessary, the implementation has already been addressed.

Two main features of our implementation still need elaboration: The framework and the timestamping system.

6.1 Framework

The framework of our implementation is given with Algorithm 4.

Algorithm 4 Kernelization Framework.

```
1:  function PerformKernelization(G = (V, E) : Graph, weightedResult : boolean)
2:      change ← true
3:      while change do
4:          change ← false
5:
6:          G.MakeUnweighted()
7:          if G.PerformUnweightedKernelization() then      ▷ Returns true on change
8:              change ← true
9:
10:         G.MakeSigned()
11:         if G.PerformSignedKernelization() then          ▷ Returns true on change
12:             change ← true
13:
14:         G.MakeWeighted()
15:         if G.PerformWeightedKernelization() then        ▷ Returns true on change
16:             change ← true
17:
18:     G.MakeUnweighted()
19:     G.PerformReductionRule26()
20:
21:     if weightedResult then
22:         G.MakeWeighted()
23:     return G
```

We perform the function MakeUnweighted as outlined in Section 5.2 – we replace every weighted edge with an unweighted subgraph of specific structure. After that, the

call PerformUnweightedKernelization executes checks and applications of Reduction Rule 23, $5^+_{w=1}$ (together with 27), 24, 25, in that order. As already mentioned earlier, Reduction Rule $5^+_{w=1}$ is the unweighted version of 5^+.

The signed variant of the input graph is created by MakeSigned. To do so, we exhaustively execute the weighted path compression by Reduction Rule 28 with the restriction that the resulting weights are -1 or $+1$. The call PerformSignedKernelization then performs the reductions by Reduction Rule 5^+. Even though this rule makes Reduction Rule $5^+_{w=1}$ obsolete, we keep both in our suite. When time efficiency is of essence in our evaluation, only the unweighted rules are tested. In all other cases, this overhead is negligible. The same is applicable to the upcoming weighted reduction rules, which also generalize two rules from the unweighted case.

Therefore, after no more signed reductions are possible, we exhaustively apply Reduction Rule 28 to fully compress all paths into weighted edges – done by the MakeWeighted function. This is then succeeded by an application of Reduction Rule $23_{w=c}$ and $25_{w=c}$ by the call PerformWeightedKernelization.

Reduction Rule 26 is applied as the last step in order to avoid cyclic interactions between it and Reduction Rule 25. If a kernel with weighted edges is permitted (as given by the boolean weightedResult), we also exhaustively perform Reduction Rule 28.

Although we do verify that no more reductions are possible in the end (besides possibly Reduction Rule 25 and $25_{w=c}$), no proof is given to support this. The only requirement to do so is to prove that Reduction Rule 26 cannot yield further applications of other rules. We also note that different permutations of the order in which all reduction rules are applied can lead to different results. We decided on the above order as it gave us the best results in terms of kernel size and time efficiency.

6.2 Timestamping

The timestamping system is the last feature of paramount importance to achieve time efficiency. It excludes unnecessary searches when repetitive checks for the applicability of certain reduction rules are performed. To achieve this goal, we assign an integer to every vertex $v \in V$ of the graph $G = (V, E)$. This integer indicates the time at which the most recent change in $N_G(v)$ occurred. We note that the following mechanism is not applied for every reduction rule, nor is it fully applicable in every case. We particularly apply it on Reduction Rule 23 and 5^+ (and all their variations). What differentiates them from other rules is that one can identify their applicability by investigating the neighbors of any vertex v for which $N_G(v)$ has changed.

6.2.1 Mechanism Description

Let the time of the most recent change in the neighbors sets be described by $T : V \to \mathbb{N}_0$ and let the variable $t \in \mathbb{N}$ describe the current time. Initially, $T(v) = 0, \forall v \in V$ and $t = 1$. For every inflicted change by a kernelization rule on $N_G(v)$, $v \in V$, set $T(v) = t, t \leftarrow t + 1$.

Now, for each individual kernelization rule r, maintain also a timestamp $t_r \in \mathbb{N}_0$ (initialized with 0), indicating the upper bound until which all vertices have already been

parsed. Meaning, all $v \in V$ with $T(v) \leq t_r$ are not directly interesting for r. The adjective "directly" is used, as some of their neighbors might indirectly still imply their required reconsideration.

For example, consider Reduction Rule 23. A vertex $v \in V$ might be important to consider even though $T(v) \leq t_r$ and $N_G(v)$ has not changed. If, at some point, a clique containing the vertex v forms, then some vertex's set $N_G(w), w \in N_G(v)$, has to have changed. Alas, even though vertex w has formed the clique by a change in $N_G(w)$, it is not necessarily an internal vertex. Such a vertex has to be then searched for in the neighboring vertices of w – where v might be an internal vertex. Therefore, for Reduction Rule 23, the neighbors of all updated vertices have to be parsed.

6.2.2 Implementation

We want to outline that it is possible to efficiently implement the above timestamping system. Besides an array of fixed size for T, an "event" list Q is utilized. In that list, all updates are stored sequentially: It holds that $\forall x, y \in 1, \ldots, |Q| : x < y$ if and only if $Q(x).T < Q(y).T$. Then, instead of having to iterate over all vertices to determine viable values in T, an iteration in Q (alongside partially utilizing the information in T) can be done. See Algorithm 5 for the pseudocode. Observe also the usage of *lastdx* there. It is being used by each rule individually to only check for the newest updates in Q.

To us, the memory usage of this solution has not been an issue. But that could be easily mitigated too by filtering out the candidates in Q whose timestamps have been invalidated. This could be done whenever $|Q| \geq \alpha|V|$ occurs, for some $\alpha > 1$.

Algorithm 5 Functions Required for Timestamping System.

```
1:  function Initialize(G = (V, E) : Graph)
2:      t ← 1
3:      for all v ∈ V(G) do
4:          T[v] ← 0
5:
6:  function UpdateTimestamp(v : V(G))
7:      Q.Append({v : v, T : t})
8:      T[v] ← t
9:      t ← t + 1
10:
11: function GetAfterTimestamp(t_r : Integer, lastdx : Integer )
12:     for i ← lastdx + 1 to |Q| do
13:         if Q[i].T = T[Q[i].v] ∧ Q[i].T ≥ t_r then
14:             R ← R ∪ Q[i].v
15:     return {result : R, lastdx : |Q|}
```

Note again that the only place this speed-up technique is utilized in our implementation is for Reduction Rule 23 and 5^+. Due to the order in which all rules are performed, the remaining rules are utilized far less and do not require a lot of time for their execution.

Experiments show that they all together use less than 50% of the time that Reduction Rule 23 and 5^+ use with the timestamping system. Oftentimes even much less than that.

7 Evaluation

Initially done by us was an implementation of the reduction rules used by Etscheid and Mnich [EM18]. After having introduced the framework of our evaluation, we first present those initial practical results implied by their theoretical work.

Following that, we provide a thorough evaluation of our kernelization. Throughout our analysis, it became clear that the density of a graph strongly dictates the impact all kernelization techniques have. That is, our results perform well on sparse and bad on dense graphs. Succeeding density on the kernelization impact is the distribution of the edges within an instance. For example, graphs with a uniform edge distribution perform significantly worse compared to instances connected according to geometric distances.

These findings, on certain properties significantly influencing the kernelization, have motivated us to execute our performance evaluation on instances created by a random graph generator (KaGen by Funke et al. [Fun+18]). This choice allowed us to adjust the density and, to certain extent, the topology. Furthermore, it led to interesting observations, which otherwise would have remained hidden.

After having scrutinized the efficiency of our kernelization on random graphs, we also show the benefits and drawbacks it induces onto existing Max-Cut solvers: Localsolver [Ben+11] [Gar+14], MqLib [DGS15], and Biq Mac [RRW10]. According to our discoveries, these represent the most significant choices when wanting to determine a maximum cut of a graph. Some capabilities also set them apart. Given enough time, the solvers Biq Mac and Localsolver can provably find a maximum cut. If either of them is unable to find one, they are also able to provide the largest found cut for the allocated time. To the best of our knowledge, MqLib does not have such abilities. It is only able to provide us the largest cut it finds after a given time.

While the convergence towards a maximum cut is significantly fastened with the usage of kernelization, the time required to perform it can be of significance. As we will show, this also applies to our implementation of the kernelization when we test very large instances. This specific situation is able to give an early advantage to the computation of a maximum cut of non-kernelized graphs. This applies to Localsolver and MqLib. Due to its inability to process large instances in a reasonable time, this drawback does not concern the Biq Mac solver.

Finally, we also analyze our performance specifically on small subgraphs (ones with less than or equal to 7 vertices). We show how much reduction is achieved by our implementation, compared to how much is possible at most.

7.1 Datasets

Our research has shown that the utilization of different graph structures is beneficial for showing the usefulness of kernelization from different aspects. Due to this, we consider the following types of test sets.

Dataset A. Randomly generated instances by the KaGen graph generator [Fun+18] [SS16]. The focus of our evaluation is set on graphs with $|E| < 8|V|$, where V is the vertex set and E the edge set. Each of the generated graphs has 2048 vertices with a varying amount of edges.

We analyze 4 graph models: The Barabási–Albert model (BA) [BA99] [SS16], The Erdős–Rényi model (GNM) [ER59], Random Geometric Graphs (RGG) [Jia04], Random Hyperbolic Graphs (RHG) [Kri+10].

Dataset B. Set of random real-world instances.

Consists of a selection of sparse graphs from the repository provided by Rossi and Ahmed [RA15].

Dataset C. Set of practical Max-Cut instances.

We investigate graphs from VLSI design by Koch and Martin [KM98] and image segmentation by Sousa, Haxhimusa, and Kropatsch [SHK13]. In both cases, we gathered the instances through the repository that implements the work by Dunning, Gupta, and Silberholz [DGS15] [18b]. The VLSI instances actually stem from works unrelated to Max-Cut. This is also why the provided edge weights are not suited for our problem and why we regard them as unweighted instead. We note that these instances remain meaningful for us due to their topology.

Dataset D. Graph instances from the Biq Mac Library [18a].

The library is divided into two graph types, named *rudy* [Rin18] and *ising*. Only the first one is relevant to us. The ising instances are poorly suited for our purpose as the absolute value of the edge weights is large and their structure is dense and uniform. These instances are better suited for benchmarking Weighted Max-Cut techniques.

Also, each graph in this set contains multiple generated instances with different seeds, denoted as *graph.x*, with $x \in \{0, \ldots, 9\}$. When providing our results, we give the average for each applicable value.

In Dataset C, we also create a subdivision into smaller and larger graphs. We use the smaller set to benchmark the computation of actual maximum cuts. This particularly applies to Biq Mac and Localsolver. Both are able to determine if a found cut is also a maximum one. The larger graphs provide insights into the required time to compute the kernel and the impact our work has on the convergence towards a maximum cut. We evaluate this for Localsolver and MqLib.

7.2 Evaluated Solvers

We give our attention to three Max-Cut solvers within our benchmark suite. Observe that all of them are also able to handle the Weighted Max-Cut problem, not only the unweighted case.

A. Localsolver [Ben+11] [Gar+14]. [1]

   A commercial software with free licenses for academic purposes. Able to handle a large spectrum of mathematical optimization problems. It heuristically searches for the largest cut it is able to find. Viable to use for the evaluation of graphs of all sizes. It can also determine when the largest found cut is a maximum one.

B. MqLib by Dunning, Gupta, and Silberholz [DGS15] (utilizing the heuristic by Burer, Monteiro, and Zhang [BMZ02]). [2]

   Implements a wide range of heuristic approaches for benchmark purposes. Of them, the best solution is the heuristic by Burer, Monteiro, and Zhang [BMZ02]. This is also the only solver we evaluate through the MqLib framework. It is unable to determine on its own when it reaches a maximum cut and always exhaust the given time limit.

C. Biq Mac solver by Rendl, Rinaldi, and Wiegele [RRW10].

   A semidefinite programming approach to determine a maximum cut. Determining a maximum cut is slow. Even for graphs with up to 200 vertices it may take hours [RRW10]. Therefore, we use this solver to primarily evaluate smaller instances. Within Biq Mac, a separate capability does also exist to compute a lower and upper bound. However, this feature was not further analyzed by us.

7.3 Environment

To fully describe under which circumstances we achieve our results, we need to outline the used environment for our experiments. Intel(R) Xeon(R) CPU E5-4640 @ 2.40GHz is used for the central processing unit. The amount of random-access memory available for our evaluation consisted of 504 GB. Every section of our implementation, including all utilized solvers, are implemented with C++. To compile our implementation, we used gcc version 7.3.0 on the Ubuntu 18.04 operating system. Compilation was done with the optimization flag -O3. The compilation of each solver was handled by the unique configuration that comes with each.

[1] We used Localsolver 8.0 binaries in our tests. See `https://www.localsolver.com/olderversions.html`.
[2] MqLib is hosted on GitHub under `https://github.com/MQLib/MQLib`. The evaluated version by us is described by commit id 0853789.

7.4 Used Metrics

For the upcoming subsections, let G be the initially given graph and G_{ker} be the final graph after all reductions have been exhaustively performed.

Kernelization Efficiency. To evaluate the kernelization efficiency of our implementation, we use the following metric: $e(G) = 1 - \frac{|V(G_{\text{ker}})|}{|V(G)|}$. The value represents the amount of removed vertices by all reduction rules. In rare occasions, $e_E(G) = 1 - \frac{|E(G_{\text{ker}})|}{|E(G)|}$ is also used to study the kernelization efficiency on the edge set. When used, it is going to be explicitly mentioned.

Kernelization Time. We also desire for the kernelization to be time efficient. For that purpose, we also analyze the time spent on kernelization. We denote this as $T_{\text{ker}}(G)$ and, unless otherwise specified, use seconds as the unit.

Time to Compute a Maximum Cut. In instances where we are able to, we use $T_{\text{mc}}(G)$ to denote the required time to determine an exact maximum cut.

7.5 Evaluation of Work by Etscheid and Mnich [EM18]

We initially perform an analysis of the linear kernel construction for Max-Cut AEE by Etscheid and Mnich [EM18]. Evaluated is an implementation of the algorithm to create the marked vertex set S such that $G - S$ is a clique forest. After this, we showcase the stated reduction rules from their paper.

For a decision problem $(G, k)_{\text{AEE}}$, we give an evaluation of their marked vertex set S computation that works in time $O(k \cdot (|V(G)| + |E(E)|))$. A greedy construction was also implemented to measure the effectiveness of their construction. The greedy approach works as follows. Determine an S for which $G - S$ is a clique forest – this S could consist of all vertices, or the ones determined from the construction by Etscheid and Mnich [EM18]. Then, try to remove any single vertex in S and check if $G - S$ remains a clique forest. If yes, remove it and repeat the same approach.

This greedy algorithm works with a customizable initial state for S because it allows us to additionally see the impact it has on reducing the size of the marked vertex set S by Etscheid and Mnich [EM18]. We denote that reduced result of S as S_X.

Following is an evaluation for the computation of S and S_X. This is then followed by our evaluation of their reduction rules.

7.5.1 Computation of the Marked Vertex Set

In the following section, we analyze the computation of a vertex set S such that $G - S$ forms a clique forest. An algorithm to do so was already briefly sketched in Section 3.4.1. For a problem instance $(G, k)_{\text{AEE}}$, it achieves the computation of S in time $O(k \cdot (|V(G)| + |E(G)|))$ and provides a bound on the size of S: $|S| \leq 3k$.

See Table 7.1 for a performance evaluation on real-world instances. As noted, S_X represents the greedy reduction of the marked vertex set S by Etscheid and Mnich [EM18].

Name	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	deg_{avg}	$\lvert S\rvert$ $[\frac{\lvert S\rvert}{\lvert V(G)\rvert}]$	$T(S)$	$\lvert S_X\rvert$ $[\frac{\lvert S_X\rvert}{\lvert V(G)\rvert}]$	$T(S_X)$
ca-CSphd	1882	1740	0.92	32 [0.02]	0.40	12 [0.01]	0.52
ego-facebook	2888	2981	1.03	9 [0.00]	0.92	5 [0.00]	0.96
ENZYMES_g295	123	139	1.13	24 [0.20]	0.01	9 [0.07]	0.02
road-euroroad	1174	1417	1.21	239 [0.20]	0.29	94 [0.08]	0.83
bio-yeast	1458	1948	1.34	196 [0.13]	0.43	93 [0.06]	1.00
rt-twitter-copen	761	1029	1.35	99 [0.13]	0.15	42 [0.06]	0.29
bio-diseasome	516	1188	2.30	79 [0.15]	0.09	27 [0.05]	0.20
ca-netscience	379	914	2.41	116 [0.31]	0.05	56 [0.15]	0.15
soc-firm-hi-tech	33	91	2.76	17 [0.52]	0.00	9 [0.27]	0.00
g000302	317	476	1.50	135 [0.43]	0.07	76 [0.24]	0.15
g001918	777	1239	1.59	380 [0.49]	0.44	202 [0.26]	0.98
g000981	110	188	1.71	57 [0.52]	0.01	30 [0.27]	0.02
g001207	84	149	1.77	38 [0.45]	0.01	22 [0.26]	0.01
g000292	212	381	1.80	98 [0.46]	0.04	82 [0.39]	0.09

Table 7.1: Evaluation of marked vertex set S computation by Etscheid and Mnich [EM18] such that $G - S$ is clique forest. Tested are small graphs from Dataset C and VLSI instances from Dataset D. The evaluation includes 5 iterations and average values are provided. The columns $T(S)$ and $T(S_X)$ represent the required time in seconds to compute S and S_X, respectively. Note that $T(S_X)$ includes the time $T(S)$.

For both, S and S_X, it is desired for their size to be as small as possible. After having been selected, they remain the same throughout all the upcoming sections. Their size also represents an exponential factor for the maximum cut computation using the algorithm outlined in Section 3.3.

We can notice two things from the table. First is that the size of the marked vertex set correlates to the density: The larger it is, the larger the marked vertex set. The other observation is that the computation time correlates to the size of the graph – as one would expect.

Evident also is that the greedy reduction of the marked vertex set has consistently a large impact on the resulting size. In every single case is the set S significantly larger than set S_X. Therefore, for practical purposes, it is of importance to consider ways of reducing this set. Other approaches have been also briefly verified by us, but none led to any more significant reduction. Moreover, if the greedy approach uses an empty set instead of the set S for its initial state, worse results are generally achieved. Due to this, we also use S_X for evaluating the kernelization in the next section.

Name	$\|V(G)\|$	$\|E(G)\|$	deg_{avg}	$\|S_X\|$ $[\frac{\|S_X\|}{\|V\|}]$	$e(G)$	$T_{ker}(G)$	$T_{mc}(G)$
ca-CSphd	1882	1740	0.92	12 [0.01]	0.09	1.10	8967.41
ego-facebook	2888	2981	1.03	5 [0.00]	0.00	0.97	61.74
ENZYMES_g295	123	139	1.13	9 [0.07]	0.00	0.02	1.35
road-euroroad	1174	1417	1.21	94 [0.08]	0.02	0.93	-
bio-yeast	1458	1948	1.34	93 [0.06]	0.01	1.07	-
rt-twitter-copen	761	1029	1.35	42 [0.06]	0.02	0.32	-
bio-diseasome	516	1188	2.30	27 [0.05]	0.26	0.32	-
ca-netscience	379	914	2.41	56 [0.15]	0.21	0.21	-
soc-firm-hi-tech	33	91	2.76	9 [0.27]	0.00	0.00	0.29
g000302	317	476	1.50	76 [0.24]	0.00	0.15	-
g001918	777	1239	1.59	202 [0.26]	0.00	0.98	-
g000981	110	188	1.71	30 [0.27]	0.00	0.02	-
g001207	84	149	1.77	22 [0.26]	0.02	0.02	13509.73
g000292	212	381	1.80	82 [0.39]	0.00	0.09	-

Table 7.2: Evaluation of the kernelization and maximum cut computation by Etscheid and Mnich [EM18] (shown in column $T_{mc}(G)$). Tested are small graphs from Dataset C and VLSI instances from Dataset D. The evaluation includes 5 iterations and average values are provided. Times in $T_{mc}(G)$ are labeled with "–" if $|S_X| > 30$ or the execution exceeded 10 hours. All times are given in seconds.

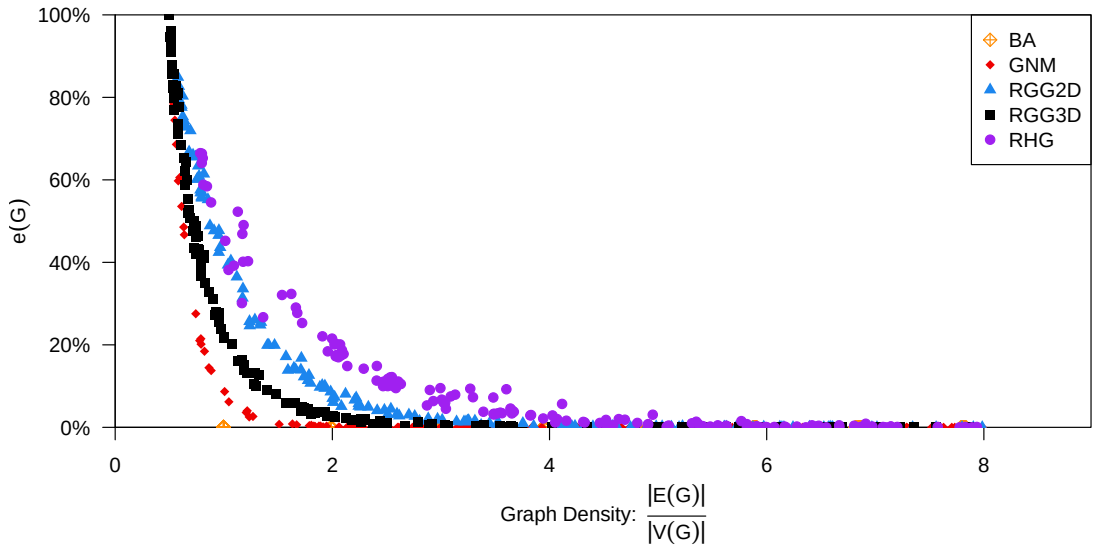

Figure 7.1: Kernelization efficiency for reduction rules by Etscheid and Mnich [EM18] on graphs generated by KaGen. Tested were 150 instances for each graph type.

7.5.2 Kernelization Performance

The kernelization by Etscheid and Mnich [EM18] consists of two reduction rules: Reduction Rule 5 and 9. Of those two, we observed in our tests that Reduction Rule 5 is predominantly applied. Moreover, it is worth noting that Reduction Rule 5 solely removes vertices. The other rule exists only to assist it in that goal.

As recognizable from Table 7.2, the achieved results generally perform poor on real-world instances. In all cases, the kernelization efficiency is below 30% and oftentimes not above 5%. In the table present is also the field $T_{\text{mc}}(G)$. This is the time to find a maximum cut of G using the algorithm by Etscheid and Mnich [EM18] with runtime $O(2^{|S_X|} \cdot (|V| + |E|))$. For all colorings of the vertices in S_X, it solves the Vertex-Weighted Max-Cut problem on the remaining graph. This computation has been performed on graphs where $|S_X| < 30$ holds. We have done this evaluation to provide some insight into the possibilities of creating a new Max-Cut solver. When compared to the results by current state of the art Max-Cut solvers, it generally performs worse. However, it is important to note that our implementation has not exhausted the existing possibilities. For example, the marked vertex set does not include cycles, but which are possible to include [MSZ18].

Created by us to further evaluate the kernelization efficiency is Figure 7.1. It depicts our evaluation on Dataset A from Section 7.1. The benchmark consists of randomly generated graphs by KaGen, each with 2048 vertices. For every graph type, we created 150 instances. Where possible, the amount of edges was uniformly selected from 0 to $8|V|$. In fact, only BA graphs did not have this ability of freely adjusting the amount of edges. They limit the number of edges to specifically be a multiple of the amount of vertices.

We are able to observe that kernelization does work in a lot of cases, but this mostly includes very sparse graphs. Furthermore, with increasing density, the kernelization efficiency decreases very fast. For $|E| > 2|V|$, three graph types already reach the domain where almost no further reduction is possible.

7.6 Impact of Kernelization on Random Instances by KaGen

In this section, we provide an analysis of our selected kernelization techniques. We do this on the exact same graphs that we have used at the end of the previous section – the instances form within Dataset A. We consider the same amount of instances and all graphs also have 2048 vertices. As previously mentioned, these instances represent our main selection to evaluate our kernelization suite. They allow us to clearly show how density and topology influence the kernelization efficiency.

Figure 7.2 depicts the resulting kernelization efficiency across all graphs. As visible, the density and type of the graph significantly dictate the amount of reduction that is possible. On all graph types, we can see that the kernelization efficiency drops with the increase of density within the instances. Also, a clear differentiation on the efficiency is visible depending on the considered graph type.

We explain the worse performance on GNM graphs in comparison to the other types through the governing topology within them. In GNM graphs, we created every edge by uniformly choosing two random endpoints from the available vertices. For a given

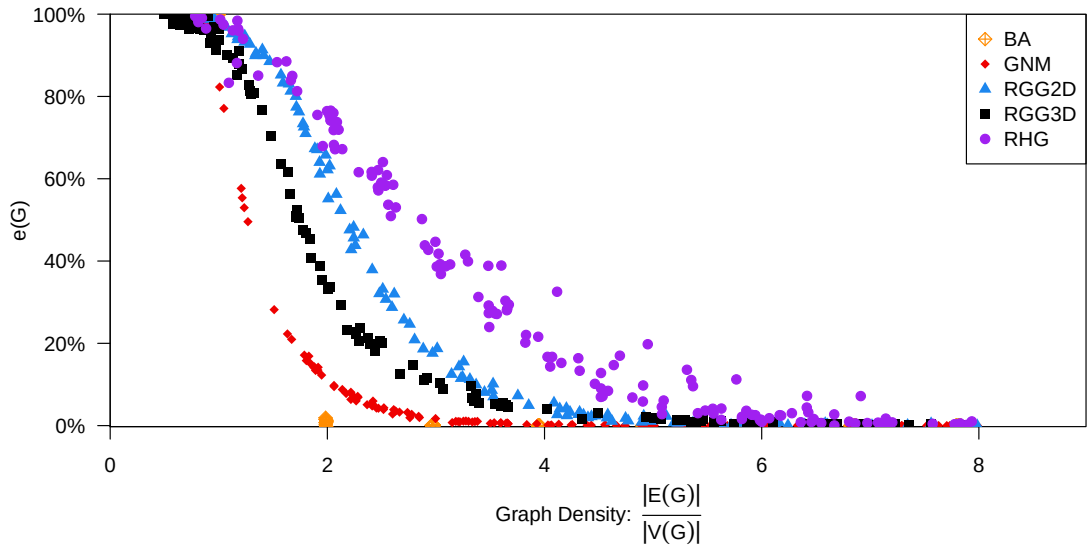

Figure 7.2: Kernelization efficiency for graphs generated by KaGen. Tested were 150 instances for each graph type.

GNM graph G and a randomly selected vertex subset $S \subseteq V(G)$, the probability of it being external is large even for small $|S|$. A single edge connected to such a vertex in S would already lead the probability of it being external in $G[S]$ lie beyond $1 - \frac{|S|}{|V(G)|}$ (the likelihood for the other endpoint to be outside of $G[S]$).

Reduction rules within our work rely on the existence of opposite structures, compared to those in GNM graphs. The existence of small subgraphs with a few external vertices is being desired for. Graphs possessing sporadic structures with denser internal connectivity should intuitively perform better, which is also strongly supported by our showcased experiments.

The required time to perform the kernelization was negligible for all instances. The maximum amount of used time for a single case is below 100 milliseconds; the total time to kernelize all 750 instances is below 30 seconds. The denser the graphs were, the larger was the required kernelization time. In other words, the size of the graph mainly influenced the required time.

In Figure 7.3, we also show how much the kernelization efficiency improves compared to that by Etscheid and Mnich [EM18].

Note: The Barabási-Albert model (BA) does not permit freely adjustable graph density. That is the reason why discrete points exist in Figure 7.2.

7.6.1 Changes when Including Weighted Path Compression

We also analyze the impact of weighted path compression by Reduction Rule 28. This was initially not our interest due its reductions possibly changing the nature of the problem – it likely turns an Unweighted Max-Cut problem into an instance of Weighted Max-Cut. The effects are significant, though.

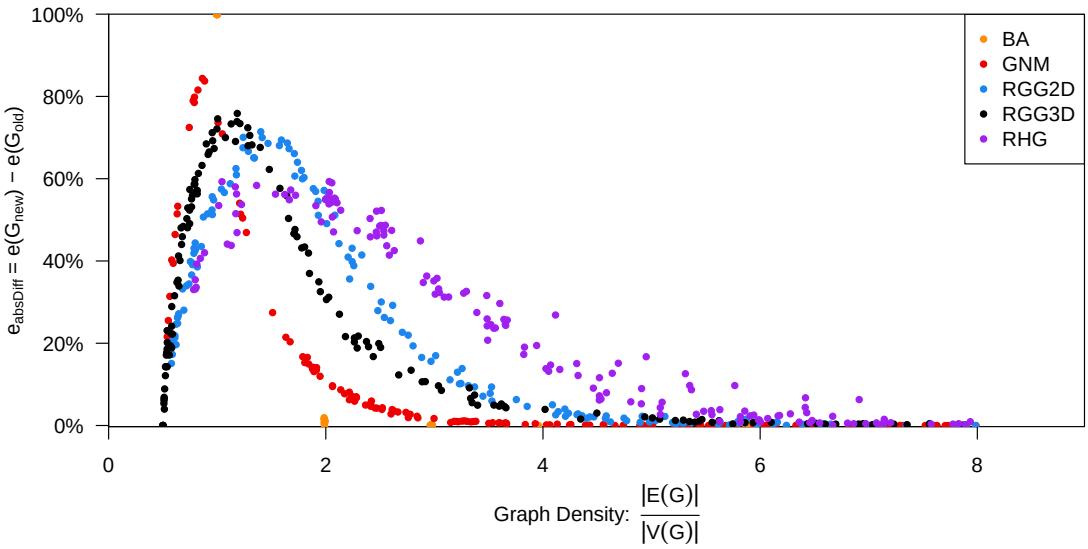

Figure 7.3: Absolute increase in kernelization efficiency for graphs generated by KaGen when our approach is utilized, compared to the reduction rules by Etscheid and Mnich [EM18]. Tested were 150 instances for each graph type.

In Figure 7.4, we plot the absolute increase in kernelization efficiency. This represents the percentage of vertices that have been additionally removed. An explanation for the significant improvement exists. For a graph G, weighted path compression is able to optimally reduce 2-paths aba' with $N_G(b) = \{a, a'\}$. This implies a viable reduction for any vertex that has exactly two neighbors.

The achieved result motivates us to believe that, even if we look at kernelization for unweighted graphs, it is meaningful to allow weighted edges in the kernel. Furthermore, all solvers we have seen do permit weighted edges. Even more, we have not encountered a solver only supporting unweighted instances.

7.6.2 Performance of Individual Rules

To analyze the impact each individual reduction rule has on the kernelization efficiency, we measure the effects their removals have on the size of the kernel. For that purpose, we created Figure 7.5 and 7.6 to depict our evaluation of RGG2D and GNM graphs, respectively. We have settled on those two types as they encompass the most important findings.

Consider Figure 7.5. We can see that Reduction Rule 23 induces the most significant reduction. Its absence always worsens the result more than any other rule. Figure 7.6 was additionally provided because it gives more insight on the usefulness of other rules. Furthermore, it also shows that the reduction rules are differently important, depending on the graph type.

We excluded Reduction Rule 26 from the figures as it only removes edges. Although, even if we consider $e_E(G)$, its effects are negligible. The same would also apply if we had tried to differentiate between ${5^+}_{w=1}$ and 5^+ – the signed and unweighted version of the

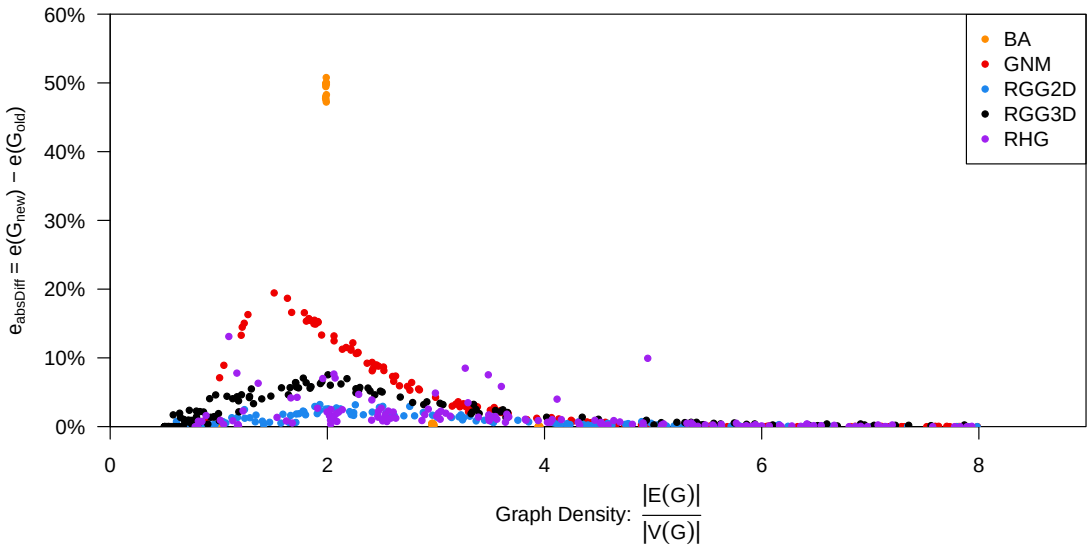

Figure 7.4: Absolute increase in kernelization efficiency for graphs generated by KaGen when weighted path compression by Reduction Rule 28 is included. Tested were 150 instances for each graph type.

reduction rule. Meaning, the signed version did not lead to any noticeable improvement within the results.

7.7 Impact of Kernelization on Biq Mac Library Instances

The Biq Mac library from Dataset D contains a variety of graphs created for benchmarking Max-Cut solvers. All graphs in their suite have an uniform edge distribution and are more dense than for what we have found kernelization to perform well on. Added to that is the case that only a small portion of their graphs is unweighted. These are exactly those prefixed with "g05" in Table 7.3.

As is inferable from the same table, we found no positive results by using our kernelization in this case. Kernelization does in most cases nothing and, when it does, it removes 2 vertices at most. Even more, this analysis also includes the allowance for weighted edges in the final kernel.

Our explanation for this is twofold. First, all instances contain a uniform edge distribution. As outlined through GNM instances, this is bad for achieving kernelization. Second, the graphs are dense enough in structure to imply worse kernelizability. Across all cases, the average vertex degree is larger than 3.5. Thus, the inability of finding any applicability in these cases is consistent to everything we have outlined until this point.

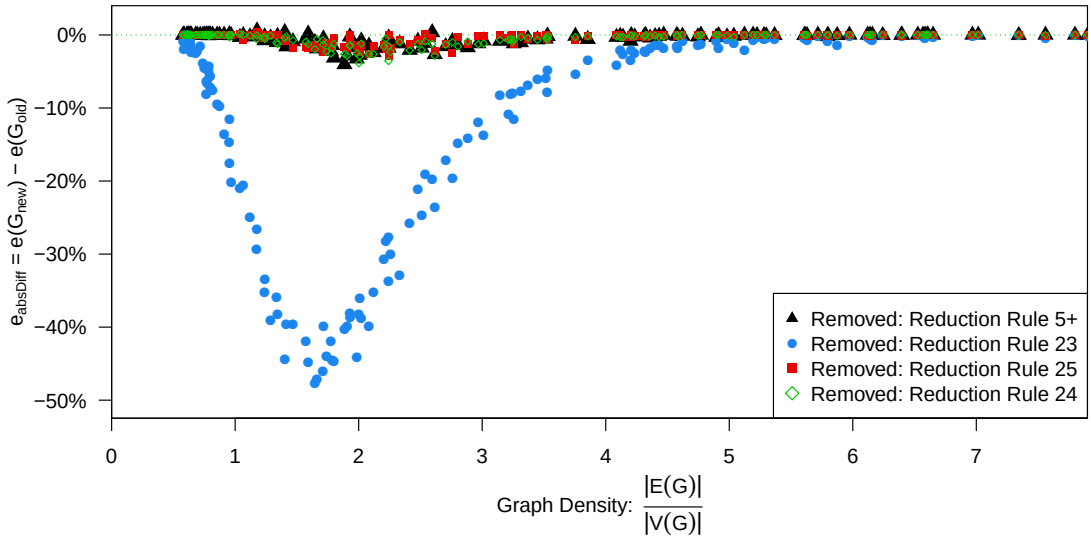

Figure 7.5: Test consist of 150 RGG2D instances generated by KaGen. Change in kernelization efficiency is presented when a reduction rule is removed (G_{new}), compared to when all are present (G_{old}).

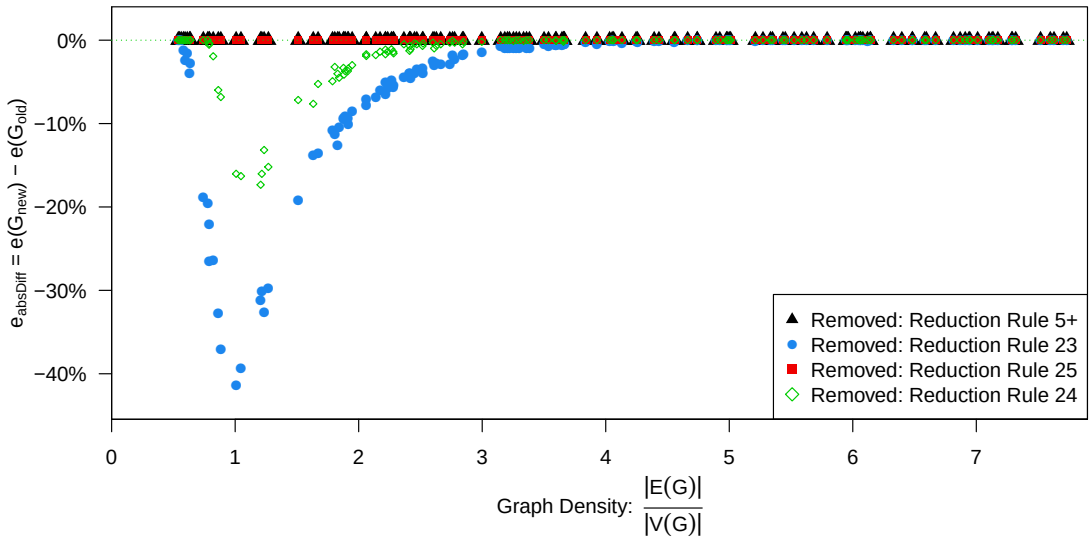

Figure 7.6: Test consist of 150 GNM instances generated by KaGen. Change in kernelization efficiency is presented when a reduction rule is removed (G_{new}), compared to when all are present (G_{old}).

Name	$\|V(G)\|$	$\|E(G)\|$	deg_{avg}	$e(G)$
g05_100	100	2475	24.75	0.00
g05_60	60	885	14.75	0.00
g05_80	80	1580	19.75	0.00
pm1d_100	100	4901	49.01	0.00
pm1d_80	80	3128	39.10	0.00
pm1s_100	100	495	4.95	0.00
pm1s_80	79	316	3.95	0.01
pw01_100	100	495	4.95	0.00
pw05_100	100	2475	24.75	0.00
pw09_100	100	4455	44.55	0.00
w01_100	100	470	4.70	0.00
w05_100	100	2356	23.56	0.00
w09_100	100	4245	42.45	0.00

Table 7.3: Result of performing every reduction rule from our work on the instances from Dataset D: Graphs from within the Biq Mac Library.

7.8 Max-Cut Solvers Benchmark

Since state of the art Max-Cut solvers are the main gateway to compute a maximum cut in practice, it is sensible for us to also investigate their performance on our kernelized graphs. For that purpose, we first present the amount of times by which Localsolver and Biq Mac are able to compute a maximum cut faster with kernelization. In all tested cases, a positive outcome is available. The maximum cut is always reached more quickly and, in some cases, we compute a maximum cut where it was not possible before. This analysis is followed by a performance evaluation of Localsolver and MqLib on very large graphs. Particularly of interest in this part is the time required to execute the kernelization and how the convergence towards a maximum cut changes.

Note that MqLib is not used when evaluating the time performance of reaching a maximum cut. It is not able to determine when it achieves a maximum cut. Furthermore, Biq Mac is not used when evaluating very large instances as it was not designed for that purpose. While capabilities exist to provide upper and lower bounds, the solver will crash when given graphs with more than 3000 vertices.

7.8.1 Exact Computation of a Maximum Cut

The approach by Localsolver has proven to perform very fast in our experiments. This allowed us to execute it over a wide spectrum of graph instances. Besides being able to provide the best found cut after a given time, it is also able to determine when a cut is a maximum one, too. This last capability also applies to the Biq Mac solver. This permitted us to evaluate the improvements kernelization induces on the time it takes for both of these solvers to determine a maximum cut. See Table 7.4a for an evaluation where weighted

path compression is not utilized at the end, and Table 7.4b where it has been used. We made this distinction because the weighted path compression yields a weighted kernel.

One can easily notice that kernelization is able to have enormous benefits on these instances. Where 10 hours have not been enough on some cases, it is now possible to find a maximum cut in seconds.

Noteworthy also is our improvement for the instances from VLSI design and image segmentation. Even though we had our focus set on unweighted kernelization, a big impact is available for weighted graphs, too. This particularly includes the cases within image segmentation where large edge weights exist. Suprisingly, Reduction Rule $23_{w=c}$ is alone responsible for the impact on image segmentation. These findings might particularly improve the work by Sousa, Haxhimusa, and Kropatsch [SHK13], which also affects the work by Dunning, Gupta, and Silberholz [DGS15].

In summary, our presented techniques are a simple tool for speeding up the computation of a maximum cut by the applicable solvers. We reinforce this with the analysis that makes one able to also determine on which instances kernelization performs well. Moreover, as one can infer from the provided tables, even our simple weighted path compression by itself is able to have a significant impact.

7.8.2 Analysis on Large Instances

To provide insight into how fast our kernelization framework works, we made an analysis on very large graph instances. In this analysis, we also measured the impact kernelization has on the execution of Localsolver and MqLib. See Table 7.5 for our results.

Notice that kernelization performs well in terms of time even for the largest graph. It achieves kernelization in minutes for 14 million vertices/edges. The achieved kernelization efficiency is also good across all the selected instances. Yet, the improvement on the size of the largest found cut by Localsolver and MqLib is generally small for the given time limit of 3 hours.

As shown in Figure 7.7, the convergence towards a maximum cut is faster on the kernelized graph for Localsolver. So, the results on the kernelized graph will generally be better after a specific time. Observe that the time performance of the kernelization plays a pivotal role on how early that occurs.

7.9 Reducibility of Small Subgraphs

In this section, we provide an overview on how much kernelization can possibly reduce small subgraphs. To do so, we categorize subgraphs into equivalence classes. We define two subgraphs H_1, H_2 as equivalent if they have the same external vertex set X and for every 2-coloring of X the relationship $\beta_\delta(H_1) = \beta_\delta(H_2) + c$ holds for some integer constant $c \in \mathbb{Z}$. This categorization is meaningful because any such two equivalent subgraphs are interchangeable within the context of computing a maximum cut of a graph. That is, if a graph G contains subgraph H_1, one can exchange H_1 by the equivalent subgraph H_2 to get G', and the relationship $\beta(G) = \beta(G') + c$ will hold. One can also reaffirm this observation through the usage of Lemma 3. Further notice that all outlined reduction

Name	$\|V(G)\|$	deg_{avg}	$e(G)$	$T_{\text{LS}}(G)$	$T_{\text{LS}}(G_{\text{ker}})$	$T_{\text{BM}}(G)$	$T_{\text{BM}}(G_{\text{ker}})$
ca-CSphd	1882	0.92	0.98	24.79	1.12 [22.23]	-	0.32 [∞]
ego-facebook	2888	1.03	0.93	20.39	1.72 [11.83]	967.99	1.42 [682.04]
ENZYMES_g295	123	1.13	0.82	1.83	0.36 [5.09]	0.96	0.37 [2.60]
road-euroroad	1174	1.21	0.69	-	- -	-	- -
bio-yeast	1458	1.34	0.72	-	- -	-	- -
rt-twitter-copen	761	1.35	0.80	-	409.47 [∞]	-	101.14 [∞]
bio-diseasome	516	2.30	0.93	-	6.66 [∞]	-	0.35 [∞]
ca-netscience	379	2.41	0.67	-	4116.61 [∞]	-	2.10 [∞]
soc-firm-hi-tech	33	2.76	0.30	4.92	2.34 [2.10]	0.29	0.31 [0.94]
g000302	317	1.50	0.10	0.71	0.50 [1.41]	1.28	0.89 [1.44]
g001918	777	1.59	0.06	1.67	1.51 [1.10]	14.90	11.69 [1.27]
g000981	110	1.71	0.22	11.32	1.97 [5.74]	0.98	0.44 [2.23]
g001207	84	1.77	0.17	1.56	0.15 [10.11]	0.47	0.37 [1.28]
g000292	212	1.80	0.01	0.69	0.51 [1.35]	0.56	0.62 [0.91]

(a) Weighted path compression by Reduction Rule 28 *is not* used at the end – kernel is unweighted.

Name	$\|V(G)\|$	deg_{avg}	$e(G)$	$T_{\text{LS}}(G)$	$T_{\text{LS}}(G_{\text{ker}})$	$T_{\text{BM}}(G)$	$T_{\text{BM}}(G_{\text{ker}})$
ca-CSphd	1882	0.92	0.99	24.07	0.32 [75.40]	-	0.06 [∞]
ego-facebook	2888	1.03	1.00	20.09	0.09 [228.91]	-	0.01 [∞]
ENZYMES_g295	123	1.13	0.86	1.22	0.33 [3.70]	0.82	0.13 [6.57]
road-euroroad	1174	1.21	0.79	-	- -	-	- -
bio-yeast	1458	1.34	0.81	-	- -	-	32726.75 [∞]
rt-twitter-copen	761	1.35	0.85	-	834.71 [∞]	-	1.77 [∞]
bio-diseasome	516	2.30	0.93	-	4.91 [∞]	-	0.07 [∞]
ca-netscience	379	2.41	0.77	-	956.03 [∞]	-	0.67 [∞]
soc-firm-hi-tech	33	2.76	0.36	4.67	1.61 [2.90]	0.09	0.06 [1.41]
g000302	317	1.50	0.21	0.58	0.49 [1.17]	1.88	0.74 [2.53]
g001918	777	1.59	0.12	1.47	1.41 [1.04]	31.11	17.45 [1.78]
g000981	110	1.71	0.28	10.73	4.73 [2.27]	531.47	21.53 [24.68]
g001207	84	1.77	0.19	1.10	0.16 [6.88]	53.20	0.06 [962.38]
g000292	212	1.80	0.03	0.45	0.45 [1.01]	0.43	0.37 [1.14]
imgseg_271031	900	1.14	0.99	10.66	0.19 [55.94]	-	0.17 [∞]
imgseg_105019	3548	1.22	0.93	234.01	22.68 [10.32]	f	13748.62 [∞]
imgseg_35058	1274	1.42	0.37	34.93	24.71 [1.41]	-	- -
imgseg_374020	5735	1.52	0.82	1739.11	72.23 [24.08]	f	- -
imgseg_106025	1565	1.68	0.68	159.31	34.05 [4.68]	-	- -

(b) Weighted path compression by Reduction Rule 28 *is* used at the end – kernel is weighted.

Table 7.4: Impact of kernelization on the computation of a maximum cut by Localsolver (LS) and Biq Mac (BM). Times are given in seconds. Kernelization is accounted for within the timings for G_{ker}. Values in brackets provide the speedup and are derived from $\frac{T(G)}{T(G_{ker})}$. Times labeled with "–" exceeded the 10 hours time limit and those with "f" denote crashes within the solvers.

Name	$\|V(G)\|$	deg_{avg}	$e(G)$	$T_{ker}(G)$	Δ_{LS}	Δ_{MQ}
inf-road_central	14081816	1.20	0.59	362.32	inf%	2.70%
inf-power	4941	1.33	0.62	0.04	1.64%	0.45%
web-google	1299	2.13	0.79	0.01	0.69%	0.19%
ca-MathSciNet	332689	2.47	0.63	8.02	1.33%	0.55%
ca-IMDB	896305	4.22	0.42	27.55	0.97%	0.32%
web-Stanford	281903	7.07	0.18	105.17	0.34%	0.30%
web-it-2004	509338	14.09	0.91	22.10	0.08%	0.02%
ca-coauthors-dblp	540486	28.20	0.25	72.39	0.05%	0.04%

Table 7.5: Evaluation of large graph instances. A 3 hours time limit was used and 5 iterations were performed. The columns Δ_{LS} and Δ_{MQ} indicate the percentage by which the size of the largest computed cut is larger on the kernelized graph compared to the non-kernelized one, for LocalSolver and MQLib, respectively.

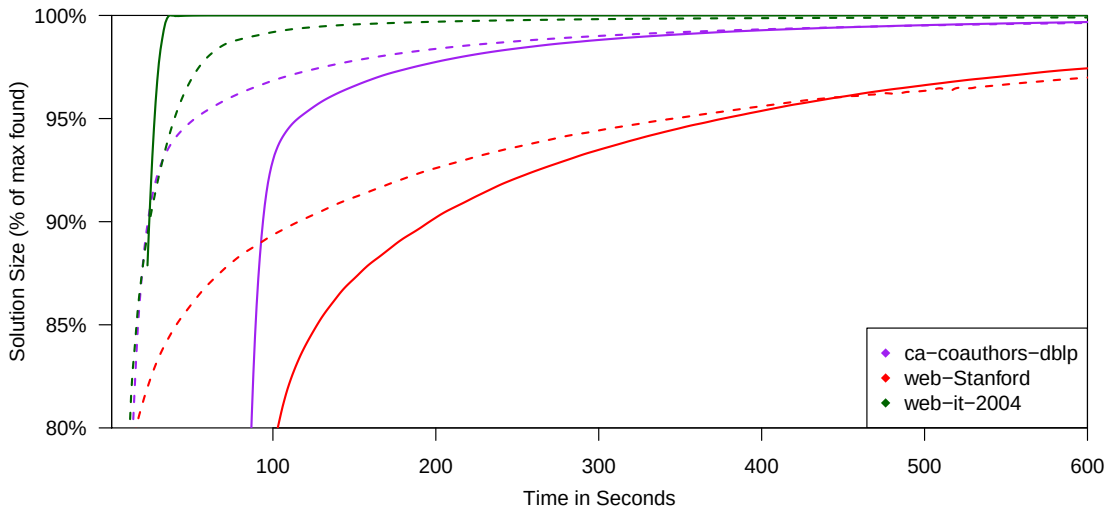

Figure 7.7: Impact of kernelization on solution convergence of LocalSolver on large graph instances. The dashed line represents the size of the cut for the non-kernelized graph, while the full line does so for the kernelized graph.

rules until now essentially describe when two subgraphs are equivalent according to the above properties.

Our goal in this chapter is to classify all subgraphs that have at most 7 vertices according to the outlined equivalence relation. This particularly includes all elements of $(n, m)_C$ for $n = 1 \dots 7$. To determine then if two subgraphs are equivalent, we try all possible 2-colorings δ of the external vertices and check if all computed β_δ values are offset by a constant. These steps also explain why we restrict ourself to small subgraphs. The computation of all subgraphs is an exponential factor, as is the checking of all 2-colorings.

The outlined classification algorithm is shown in Algorithm 6 and Table 7.6 is the result of running its implementation.

Algorithm 6 Equivalence classification of graphs in $(n, m)_C$.

```
1:  function GetDiffs(cuts : Ordered list of integers)
2:      return {cuts_i − cuts_{i+1} | i ∈ {1, ..., |cuts|}}
3:
4:  function CreateEquivalenceClasses
5:      Classes ← ∅                                          ▷ Hash table
6:      for all H ∈ (n, m)_C do
7:          cuts ← ∅                                         ▷ Ordered list
8:          for all δ : {1, ..., m} → {0, 1} do
9:              H^c ← H with first m vertices colored according to δ
10:             cuts.Append(β_δ(H^c))
11:         ClassKey ←GetDiffs(cuts)    ▷ Guarantees equivalence for constant offsets
12:         Classes[ClassKey] ← Classes[ClassKey] ⋃ H
        return Classes
```

Observable from Table 7.6 is that the number of external vertices plays a significant role on how well it is possible to reduce an induced subgraph. The more external vertices a subgraph has, the more equivalence classes exist in which it could belong to. As each such class is reducible to at most a single graph, a large amount of classes indicates a lesser possibility for reduction in total. For that purpose, we also created Figure 7.8 to show how much all our reduction rules are able to reduce all classes into a single representative. As we remove the isomorphisms, this provides rather a theoretical viewpoint than a practical one.

$\|S\|$	$\|C_{\text{ext}(G)}(S)\|$	# of classes	# of non-isomorph classes	# of graphs with $\|S\|$ vertices
3	2	3	3	8
3	3	8	4	8
4	2	5	5	64
4	3	28	11	64
4	4	64	11	64
5	2	6	6	1024
5	3	89	28	1024
5	4	576	56	1024
5	5	1024	34	1024
6	2	8	8	32768
6	3	169	48	32768
6	4	4970	376	32768
6	5	22528	380	32768
6	6	32768	156	32768
7	2	9	9	2097152
7	3	305	80	2097152
7	4	30978	1929	2097152
7	5	516272	6383	2097152
7	6	1654784	3980	2097152
7	7	2097152	1044	2097152
8	3	$\geq$ 396	-	268435456
8	4	$\geq$ 58637	-	268435456
8	5	$\geq$ 707714	-	268435456
9	2	$\geq$ 11	-	68719476736
9	3	$\geq$ 512	-	68719476736
9	4	$\geq$ 87509	-	68719476736
9	5	$\geq$ 878067	-	68719476736
10	2	$\geq$ 11	-	35184372088832

Table 7.6: Overview of equivalence classes. The sections where values are prefixed by "≥" are proven lower bounds. In those cases, there are too many graphs for an efficient evaluation, so a sampling of 10^6 random graphs was done. Entries marked with "–" have not been computed due to time/memory restriction.

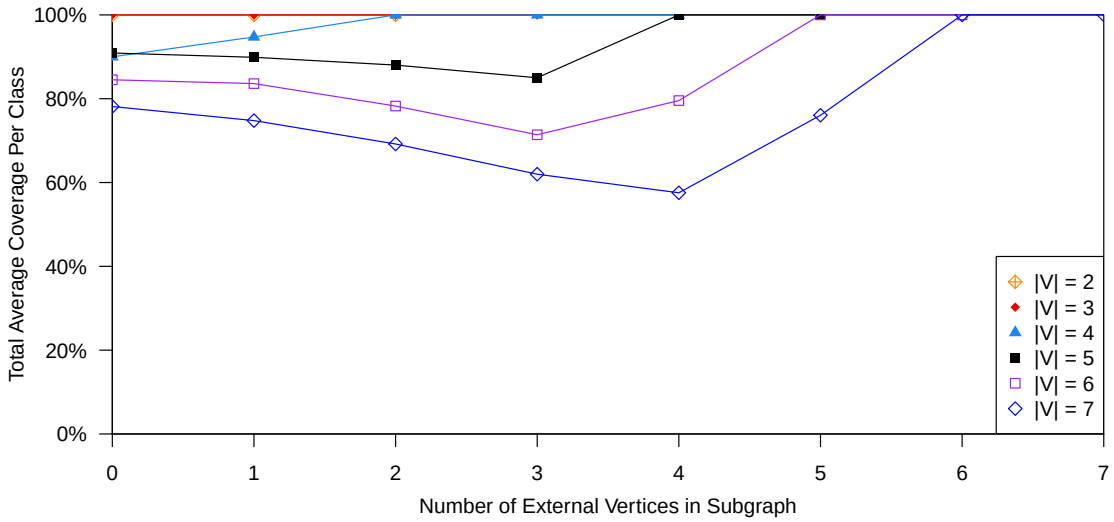

Figure 7.8: Reducibility of small subgraphs. For each equivalence class, the amount of subgraphs reduced to a single instance is given. Across all classes, the average is taken. Isomorphic graphs are treated the same. See Figure 9.1 in Appendix for the plot where isomorphisms are treated separately.

8 Conclusion

We have shown that there are a lot of areas where kernelization techniques are a viable tool to get better results. They are effective for a variety of graphs and are a particularly powerful tool within the domain of sparse instances.

Where successful, kernelization enables the Max-Cut solvers to perform better. We achieve a faster convergence and enormous speed-ups within the computation of a maximum cut by Biq Mac and Localsolver.

Also outlined were the necessary steps to achieve an efficient implementation of all kernelization rules. For medium-sized instances (less than 10^4 vertices), kernelization induces a negligible overhead; and with enough time allocated, it has also made a positive impact on the results of larger instances (even for graphs with more than 10^6 vertices).

Multiple theoretical contributions were also given. We introduced new kernelization rules and studied the relationship among all found rules. Furthermore, our experiments revealed under which circumstances kernelization performs best and under which worst. We have seen that the reduction of a subgraph is tightly tied to the amount of external vertices it contains: The greater their count, the less kernelization is possible. We strengthened this observation through different experiments. Notable here is the total reduction analysis on very small instances, and the comparison of the efficiency between graphs with uniform edge distribution and those with a non-uniform one.

Challenges remain. We believe that the investigation of new reduction rules is an important course for future studies. The consideration of weighted kernelization could be a significant key in that direction. This specifically includes reduction rules that also yield a weighted kernel as a result of their application. Also of importance may be further contribution towards an even more efficient implementation of the kernelization – possibly with a multi-threaded approach. This could then lead to a state where kernelization is also always advisable for large instances.

9 Appendix

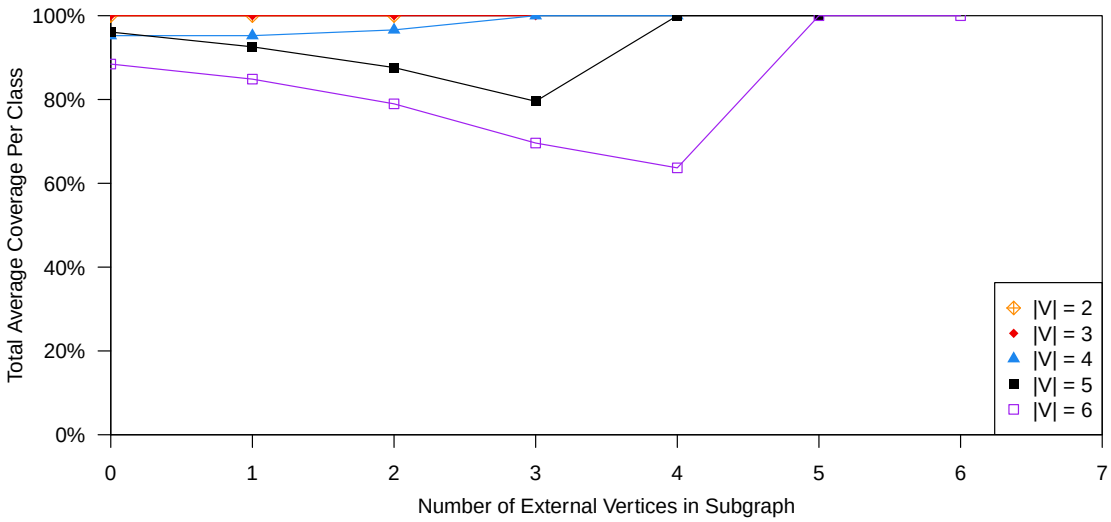

Figure 9.1: Reducibility of small subgraphs. For each equivalence class, the amount of subgraphs reduced to a single instance is given. Across all classes, the average is taken. Isomorphic graphs are treated separately.

Bibliography

[18a] *BiqMac Library*. `http://biqmac.aau.at/biqmaclib.html`. [Online; accessed 2-September-2018]. 2018.

[18b] *MQLib*. `https://github.com/MQLib/MQLib`. [Online; accessed 2-September-2018]. 2018.

[Abu+04] Faisal N Abu-Khzam, Rebecca L Collins, Michael R Fellows, Michael A Langston, W Henry Suters, and Christopher T Symons. "Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments". In: *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics*. 2004, pp. 62–69.

[BA99] Albert-László Barabási and Réka Albert. "Emergence of scaling in random networks". In: *science* 286.5439 (1999), pp. 509–512.

[Bar+88] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. "An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design". In: *Operations Research* 36.3 (1988), pp. 493–513.

[Bar82] Francisco Barahona. "On the computational complexity of Ising spin glass models". In: *Journal of Physics A: Mathematical and General* 15.10 (1982), p. 3241.

[Bar96] Francisco Barahona. "Network design using cut inequalities". In: *SIAM Journal on optimization* 6.3 (1996), pp. 823–837.

[Ben+11] Thierry Benoist, Bertrand Estellon, Frédéric Gardi, Romain Megel, and Karim Nouioua. "Localsolver 1.x: a black-box local-search solver for 0-1 programming". In: *4OR* 9.3 (2011). [used in this work: Localsolver 8.0], p. 299. URL: `https://www.localsolver.com/`.

[BM+76] John A Bondy, U S R Murty, et al. *Graph theory with applications*. Vol. 290. Citeseer, 1976.

[BMZ02] Samuel Burer, Renato D C Monteiro, and Yin Zhang. "Rank-two relaxation heuristics for max-cut and other binary quadratic programs". In: *SIAM Journal on Optimization* 12.2 (2002), pp. 503–521.

[Chi+07] Charles Chiang, Andrew B Kahng, Subarnarekha Sinha, Xu Xu, and Alexander Z Zelikovsky. "Fast and efficient bright-field AAPSM conflict detection and correction". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.1 (2007), pp. 115–126.

[CJM15] Robert Crowston, Mark Jones, and Matthias Mnich. "Max-cut parameterized above the Edwards-Erdős bound". In: *Algorithmica* 72.3 (2015), pp. 734–757.

[Cro+13] Robert Crowston, Gregory Gutin, Mark Jones, and Gabriele Muciaccia. "Maximum balanced subgraph problem parameterized above lower bound". In: *Theoretical Computer Science* 513 (2013), pp. 53–64.

[Cro+14] Robert Crowston, Michael Fellows, Gregory Gutin, Mark Jones, Eun J Kim, Fran Rosamond, Imre Z Ruzsa, Stéphan Thomassé, and Anders Yeo. "Satisfying more than half of a system of linear equations over GF (2): A multivariate approach". In: *Journal of Computer and System Sciences* 80.4 (2014), pp. 687–696.

[Cyg+] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Vol. 3. Springer.

[De 59] Rene De La Briandais. "File searching using variable length keys". In: *Papers presented at the the March 3-5, 1959, western joint computer conference*. ACM. 1959, pp. 295–298.

[DF12] Rodney G Downey and Michael R Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[DGS15] Iain Dunning, Swati Gupta, and John Silberholz. "What Works Best When? A Framework for Systematic Heuristic Evaluation". In: (2015).

[Edw73] Christopher S Edwards. "Some extremal properties of bipartite subgraphs". In: *Canadian Journal of Mathematics* 25.3 (1973), pp. 475–485.

[Edw75] Christopher S Edwards. "An improved lower bound for the number of edges in a largest bipartite subgraph". In: *Proceedings of the Second Czechoslovak Symposium on Graph Theory, Prague*. 1975, pp. 167–181.

[EM18] Michael Etscheid and Matthias Mnich. "Linear Kernels and Linear-Time Algorithms for Finding Large Cuts". In: *Algorithmica* 80.9 (Sept. 2018), pp. 2574–2615.

[ER59] Paul Erdős and Alfréd Rényi. "On Random Graphs I". In: *Publicationes Mathematicae Debrecen* 6 (1959), p. 290.

[Erd65] Paul Erdős. "On some extremal problems in graph theory". In: *Israel Journal of Mathematics* 3.2 (1965), pp. 113–116.

[Fre60] Edward Fredkin. "Trie memory". In: *Communications of the ACM* 3.9 (1960), pp. 490–499.

[Fun+18] Daniel Funke, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. "Communication-free Massively Distributed Graph Generation". In: *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21 – May 25, 2018*. 2018.

[Gar+14] Frédéric Gardi, Thierry Benoist, Julien Darlay, Bertrand Estellon, and Romain Megel. *Mathematical Programming Solver Based on Local Search*. FOCUS Series in Computer Engineering. ISTE Wiley, 2014, p. 112.

[GY10] Gregory Gutin and Anders Yeo. “Note on maximal bisection above tight lower bound”. In: *Information Processing Letters* 21.110 (2010), pp. 966–969.

[Had75] Frank Hadlock. “Finding a maximum cut of a planar graph in polynomial time”. In: *SIAM Journal on Computing* 4.3 (1975), pp. 221–225.

[Har59] Frank Harary. “On the measurement of structural balance”. In: *Behavioral Science* 4.4 (1959), pp. 316–323.

[HLW02] Frank Harary, Meng-Hiot Lim, and Donald C Wunsch. “Signed graphs for portfolio analysis in risk management”. In: *IMA Journal of management mathematics* 13.3 (2002), pp. 201–210.

[HSS18] Demian Hespe, Christian Schulz, and Darren Strash. “Scalable Kernelization for Maximum Independent Sets”. In: *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2018, pp. 223–237.

[Jia04] Xingde Jia. “Wireless networks and random geometric graphs”. In: *7th International Symposium on Parallel Architectures, Algorithms and Networks, 2004. Proceedings.* IEEE. 2004, pp. 575–579.

[Kar72] Richard M Karp. “Reducibility among combinatorial problems”. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[KM98] Thorsten Koch and Alexander Martin. “Solving Steiner tree problems in graphs to optimality”. In: *Networks: An International Journal* 32.3 (1998), pp. 207–232.

[Kőn31] Dénes Kőnig. “Gráfok és mátrixok”. In: *Matematikai és Fizikai Lapok* 38 (1931), pp. 116–119.

[Kri+10] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. “Hyperbolic geometry of complex networks”. In: *Physical Review E* 82.3 (2010).

[Lam+17] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F Werneck. “Finding near-optimal independent sets at scale”. In: *Journal of Heuristics* 23.4 (2017), pp. 207–229.

[Len12] Thomas Lengauer. *Combinatorial algorithms for integrated circuit layout*. Springer Science & Business Media, 2012.

[MR99] Meena Mahajan and Venkatesh Raman. “Parameterizing above guaranteed values: MaxSat and MaxCut”. In: *Journal of Algorithms* 31.2 (1999), pp. 335–354.

[MRS09] Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. “Parameterizing above or below guaranteed values”. In: *Journal of Computer and System Sciences* 75.2 (2009), pp. 137–153.

[MSZ18] Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. “Max-Cut Above Spanning Tree is Fixed-Parameter Tractable”. In: *Computer Science - Theory and Applications - 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6-10, 2018, Proceedings*. 2018, pp. 244–256.

[PR12] Fahad Panolan and Ashutosh Rai. "On the Kernelization Complexity of Problems on Graphs without Long Odd Cycles". In: *Computing and Combinatorics*. Ed. by Joachim Gudmundsson, Julián Mestre, and Taso Viglas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 445–457.

[Pri05] Elena Prieto. "The Method of Extremal Structure on the K-maximum Cut Problem". In: *Proceedings of the Eleventh Australasian Symposium on Theory of Computing*. Vol. 41. CATS '05. Newcastle, Australia: Australian Computer Society, Inc., 2005, pp. 119–126.

[RA15] Ryan A Rossi and Nesreen K Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization". In: *AAAI*. Vol. 15. 2015, pp. 4292–4293.

[Rin18] Giovanni Rinaldi. *Rudy*. `http://biqmac.aau.at/library/tar_files/mac_all.tar.gz`. [Online; accessed 2-September-2018]. 2018.

[RRW10] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. "Solving Max-Cut to Optimality by Intersecting Semidefinite and Polyhedral Relaxations". In: *Mathematical Programming* 121.2 (2010), p. 307.

[SHK13] Samuel de Sousa, Yll Haxhimusa, and Walter G Kropatsch. "Estimation of distribution algorithm for the max-cut problem". In: *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer. 2013, pp. 244–253.

[SS16] Peter Sanders and Christian Schulz. "Scalable generation of scale-free graphs". In: *Information Processing Letters* 116.7 (2016), pp. 489–491.

[Suc+17] Rubens Sucupira, Ignasi Sau, Sulamita Klein, and Luerbio Faria. "Improved kernels for Signed Max Cut parameterized above lower bound on (r, l)-graphs". In: *Discrete Mathematics & Theoretical Computer Science* 19 (2017).

[VT93] Nguyen Van Ngoc and Zsolt Tuza. "Linear-time approximation algorithms for the max cut problem". In: *Combinatorics, Probability and Computing* 2.2 (1993), pp. 201–210.

[XK88] Xiao-Ming Xiong and Ernest S Kuh. "The constrained via minimization problem for PCB and VLSI design". In: *Proceedings of the Twenty-Fifth ACM/IEEE Design Automation Conference*. IEEE Computer Society Press. 1988, pp. 573–578.