Separator Based Data Reduction for the Maximum Cut Problem

Jonas Charfreitag
Institute of Computer Science, University of Bonn, Germany

Christine Dahn
Institute of Computer Science, University of Bonn, Germany

Michael Kaibel
Institute of Computer Science, University of Bonn, Germany

Philip Mayer
Institute of Computer Science, University of Bonn, Germany

Petra Mutzel
Institute of Computer Science, University of Bonn, Germany

Lukas Schürmann
Institute of Computer Science, University of Bonn, Germany

Abstract

Preprocessing is an important ingredient for solving the maximum cut problem to optimality on real-world graphs. In our work, we derive a new framework for data reduction rules based on vertex separators. Vertex separators are sets of vertices, whose removal increases the number of connected components of a graph. Certain small separators can be found in linear time, allowing for an efficient combination of our framework with existing data reduction rules. Additionally, we complement known data reduction rules for triangles with a new one.

In our computational experiments on established benchmark instances, we clearly show the effectiveness and efficiency of our proposed data reduction techniques. The resulting graphs are significantly smaller than in earlier studies and sometimes no vertex is left, so preprocessing has fully solved the instance to optimality. The introduced techniques are also shown to offer significant speedup potential for an exact state-of-the-art solver and to help a state-of-the-art heuristic to produce solutions of higher quality.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Mathematics of computing → Solvers; Theory of computation → Network optimization

Keywords and phrases Data Reduction, Maximum Cut, Vertex Separators

Digital Object Identifier 10.4230/LIPIcs.SEA.2024.4

Supplementary Material
Software (Source Code): `https://github.com/CharJon/MaxCutDataReduction`
archived at `swh:1:dir:2688a073fd6e4ad2414ddb52d40757da911a7bd7`

Funding This work was partially funded by the German Research Foundation (DFG) EXC-2047/1 - 390685813 and FOR-5361 - 459420781, and by the BMBF (Germany) and state of NRW as part of the Lamarr-Institute, LAMARR22B.

Acknowledgements We thank Konstantinos Papadopoulos for his contribution to this work.

1 Introduction

The maximum cut problem (MaxCut) asks for a vertex-bipartition of a given edge-weighted input graph G maximizing the value of the cut. The cut is defined as the set of edges connecting vertices from different partitions and its value is the sum over all weights of edges forming the cut. MaxCut is a fundamental problem in computer science and is one of the NP-hard problems on Karp's famous list [28] of 21 problems. The research on MaxCut is strongly motivated by its applications in, e.g., image processing [37] or VLSI design [5], and its importance for quantum annealing [25]. To tackle general graphs encountered in these applications, exact state-of-the-art solvers like McSparse [8] and QuBowl [35] employ a multitude of different techniques. One of those is preprocessing (also called presolving).

Preprocessing algorithms make use of decomposition techniques and data reduction rules without sacrificing optimal solutions. The former split the given input into several independent smaller instances, and the latter reduce the input size. Efficient and effective data reduction rules for MaxCut have been designed in e.g. [29], [14] and [35]. They have been proven to play a crucial role in solving difficult instances for MaxCut.

MaxCut, as a combinatorial optimization problem on graphs, allows for a natural decomposition of the input into its biconnected components [21]. More sophisticated techniques based on divide-and-conquer algorithms using vertex separators have been used for special cases such as graphs with bounded treewidth [43] or graphs not contractible to K_5 [4] or to $K_{3,3}$ [10]. We call a set of vertices a vertex separator of a graph if its removal from the graph increases the number of connected components.

Our Contribution

We derive new exact data reduction techniques for MaxCut and evaluate their performance in a sophisticated experimental study. In detail:

1. We design a framework for exact data reduction, making use of vertex separators, which have not been made practical use of in the context of data reduction for MaxCut before. Given a vertex separator, we provide a general characterization of when and how it can be exploited for data reduction.
2. For small separators, which can be found efficiently, we derive concrete and effective rules through our framework. For certain structures in the input, our framework also allows to generalize existing rules.
3. For triangles in the input graph we introduce a new data reduction rule, complementing two existing ones.
4. We carefully engineer an algorithm consisting of state-of-the-art techniques and our new ones. We evaluate it on well-established benchmark graphs and compare the results with the state-of-the-art. Additionally, we investigate the influence of our algorithm's individual components and their influence on the solution quality of a state-of-the-art heuristic and runtime of a state-of-the-art exact solver.

Outline

The remainder of the paper is structured as follows: In Section 2 we introduce some basic notation relevant for the theory of our paper, along with an overview of the related literature in the area of data reduction and MaxCut. Section 3 first describes existing data reduction rules that exploit weights of edges in detail. Afterwards, our new one for triangles is presented. Section 4 focuses on vertex separators and generalizes existing data reduction rules through the lens of the new framework and presents our new rules derived from this framework. The algorithm we engineered is described in Section 5 and evaluated extensively in Section 6. Section 7 concludes our findings and results.

2 Preliminaries

2.1 Notation

Throughout the paper, we consider simple undirected graphs $G = (V, E, w)$ with edge weights $w : E \to \mathbb{R}$. For better readability we write $w(e) = w_e$ and $w(\{u, v\}) = w_{uv}$. $N(v)$ for $v \in V$ denotes the neighborhood of v in G and $d(v)$ its degree, so $d(v) = |N(v)|$. For sets $S \subset V$ we write $N(S)$ to capture the union of all neighborhoods of vertices in S excluding all vertices in S. For a set of vertices $V' \subset V$ we denote by $G[V']$ the subgraph induced by V'. We define the contraction of an edge $e = \{u, v\}$ as replacing u and v by a new vertex a. For every vertex $b \in N(\{u, v\})$ an edge $\{a, b\}$ is added with weight $w_{ub} + w_{vb}$ if both u and v are adjacent to b and the same weight as the edge between u or v and b otherwise.

A *connected component* of a graph G is a maximal subgraph of G in which any pair of vertices can reach each other by a walk. If a graph has exactly one connected component, it is called a *connected* graph. A graph with more than $k \geq 1$ vertices is called *k-connected* if it is connected, and the deletion of an arbitrary set of up to $k - 1$ vertices does not change this property. For a graph G a (non-empty) set of vertices $S \subset V$, whose removal increases the number of connected components by at least one, is called a *k-separator* with $k = |S|$.

A bipartition P of the vertices of a graph $G = (V, E)$ is a pair $P = (S_1, S_2)$, with $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = V$. We also define a partial bipartition $P' = (S'_1, S'_2)$ of the vertices, with again $S'_1 \cap S'_2 = \emptyset$, but $S'_1 \cup S'_2 \subset V$. Every vertex bipartition $P = (S_1, S_2)$ induces a set of edges, called a *cut* $\delta_G(P) = \{\{u, v\} \in E \mid u \in S_1 \wedge v \in S_2\}$. If the underlying graph is clear from the context, we write $\delta(P)$. The value of a cut $\delta(P)$ in graph $G = (V, E, w)$ is given by $\sum_{e \in \delta(P)} w_e$. We write $\Delta(G)$ for the value of a maximum cut in G. For a given partial bipartition $P' = (S'_1, S'_2)$ we write $\Delta(G, P')$ to denote the maximum value of all cuts in G respecting P'. A cut respects a partial bipartition if none of its edges connects vertices from the same partition in P'.

Data Reduction. A data reduction rule is applied to a weighted graph $G = (V, E, w)$ in order to reduce its size or complexity. Independent of the concrete technique, a data reduction rule is said to be *valid* if the new instance $G' = (V', E', w')$ retains an optimal solution of the original instance. We call a valid data reduction *feasible* if it can be applied in polynomial time. To capture the validity of graph transformations more formally, we introduce the following condition:

▶ **Condition 2.1** (Valid Data Transformation). *Let $G = (V, E, w)$ be a simple undirected weighted graph. Any MaxCut-valid data transformation transforms an input graph G into G' and outputs some constant offset $\beta \in \mathbb{R}$. The MaxCut-value in G has to be the same as in G', apart from the offset β: $\Delta(G) = \Delta(G') + \beta$.*

A (valid) data transformation is called a (valid) data reduction, if it reduces the number of vertices or edges of the input graph. Note: We clearly distinguish between kernelization and data reduction. Kernelization in the standard literature is only defined in the context of a problem specific input parameter (like treewidth, see e.g. [15]) and a kernelization algorithm is required to produce a so-called *kernel*, whose size is bounded by a polynomial function in the parameter. As the algorithms considered here do not rely on any input-dependent parameter, we will solely use the term data reduction.

2.2 State-of-the-Art

The literature on the three main topics of this paper, namely MaxCut, data reduction and separators, is extensive. Therefore we will only present the most relevant related work in this section, starting with MaxCut.

For general graphs MaxCut is NP-hard [28] and APX-hard [34], even if all edge weights are restricted to 1. For graphs with only non-negative edge weights Goemans and Williamson [16] suggest an algorithm with an approximation guarantee of 0.87856. Polynomial time algorithms for MaxCut have been designed for certain classes of input graphs, e.g., planar graphs [20, 4, 30] and graphs not contractible to K_5 [4] or $K_{3,3}$ [10]. Other tractable classes of graphs require specific edge weight distributions (see, e.g., [32]). For some of those divide-and-conquer paradigms have been used. E.g., for a given planar graph, Barahona [4] suggested transforming the maximum cut problem into a Chinese postman problem in the dual, which was then solved using the planar separator theorem. For the more general class of graphs not contractible to K_5, Barahona [4] suggested a recursive approach based on the so-called k-sum decomposition for $k \leq 3$, which has been shown to exist for this class by Wagner [41]. For graphs with bounded treewidth and a given tree decomposition, Wimer [43] has shown that a bottom-up approach based on dynamic programming will solve the maximum cut problem in linear time.

Vertex separators and k-connected components for small k have been studied in many publications. For $k \leq 3$, k-connected components can be identified in linear time. While the decomposition of a graph into its biconnected components can be computed running a modified depth-first-search [40], the algorithm for decomposition into its triconnected components is more involved [22, 19]. Using Tarjan's decomposition algorithm [40] and the data structure of SPQR-trees [39], the set of all vertex separators of sizes $k \leq 2$ can be enumerated efficiently. There are also theoretical approaches for identifying 4-connected components, however, they admit no clear definition [18]. Still, vertex-separators of size 3 can be found in $O(n^2)$ [27] and for general k techniques based on maximum flow algorithms have been employed [13, 26]. Vertex separators have been made use of to speed up algorithms in the past: E.g. Iwata and Shigemura [24] employ a vertex separator-based pruning in a dynamic programming approach for Steiner trees. Hüffner et al. [23] developed reduction rules for *signed graph balancing* based on vertex separators. We will discuss the differences to our approach in Section 4.

The most recent and well-performing data reduction techniques for MaxCut from the literature will be covered in the next two sections. We refer to [1] for a general overview of recent progress in data reduction algorithms for problems in NP and P.

3 Edge Weight Based Reduction

One type of data reduction rules for MaxCut considers the edges (and their weights) with exactly one endpoint in a set of vertices S. We present them here with our new rule, complementing two existing ones. All rules of this type (implicitly) make use of the following observations.

▶ **Observation 3.1.** *Let $G = (V, E, w)$ be a simple undirected weighted graph. If for an edge $e = \{u, v\} \in E$ in G we can prove the existence of a maximum cut $\hat{\delta}$ with $e \notin \hat{\delta}$, the contraction of u and v is a valid data transformation (with $\beta = 0$).*

In some cases, we find a proof for an edge $e \in E$ that there exists a maximum cut $\hat{\delta}$ with $e \in \hat{\delta}$. Here we can not make use of the above Observation 3.1 right away, but Fact 2 and Definition 4 in the work of Lange et al. [29] still allow to derive a data reduction rule. We restate their observations, for self-containedness:

▶ **Proposition 3.2.** *Let G be an undirected weighted graph $G = (V, E, w)$ and δ^* any cut in G. Transforming G into $H = (V, E, w')$, by negating all weights of edges in δ^* is a valid data reduction with $\beta = \sum_{e \in \delta^*} w_e$. If an edge e is part of an optimal cut $\hat{\delta}_G$ and also of δ^*, there exists an optimal cut $\hat{\delta}_H$, which does not contain e.*

With this, if a maximum cut $\hat{\delta}$ with $\{u, v\} \in \hat{\delta}$ is guaranteed to exist, we can either use $\delta^* = \delta(\{u\})$ or $\delta^* = \delta(\{v\})$ as the cut for the transformation from G to H and then make use of Observation 3.1 for a contraction in H. See Appendix A.1 for a proof.

Dominating Edges

Lange et al. [29] derived data reduction rules for edges with relatively high absolute weight:

▶ **Proposition 3.3** (Dominating Edge). *Let $G = (V, E, w)$. If for any edge $e = \{u, v\} \in E$ and a subset $U \subset V$ with $e \in \delta(U)$ the inequality*

$$|w_e| \geq \sum_{e' \in \delta(U) \setminus \{e\}} |w_{e'}|$$

holds, then there exists a cut $\hat{\delta}$ with maximum value with $e \notin \hat{\delta}$ if $w_e \leq 0$ and $e \in \hat{\delta}$ if $w_e \geq 0$.

Proposition 3.3 naturally extends into a data reduction rule with Observation 3.1 and Proposition 3.2. For vertices with degree one or two, the condition is always true, therefore they can always be removed. Finding candidate sets U for data reduction can be done via Gomory-Hu trees [17]. As already in previous work [29, 35], in our experiments we opt for the faster approach and only consider $U = \{u\}$ and $U = \{v\}$ for an edge $\{u, v\}$, resulting in $O(|E|)$ time.

Similar Vertices

For vertices with similar neighborhoods, one can identify cases in which both end up in the same / opposite partitions of an optimal solution, as Rehfeldt et al. [35] showed.

▶ **Proposition 3.4** (Similar Vertices). *Let $G = (V, E, w)$. If two vertices $u, v \in V$ have the same neighborhood (excluding each other) $N(u) \setminus \{v\} = N(v) \setminus \{u\}$ and there exists an $\alpha \neq 0$ with $w_{ux} = \alpha w_{vx} \forall x \in N(u) \setminus \{v\}$, then, if*

- *$\alpha > 0$ and ($\{u, v\} \notin E$ or $w_{uv} \leq 0$), there is an optimal solution with u and v in same partition*
- *$\alpha < 0$ and ($\{u, v\} \notin E$ or $w_{uv} \geq 0$), there is an optimal solution with u and v in different partitions*

As also described in [35] candidates for this rule can be found quickly by making use of hashing techniques and we follow their suggestion in our implementation.

Triangles

For triangles in the input graph, the literature [29, 35] suggests two closely related data reduction rules. We complement them with a third and new one.

▶ **Proposition 3.5** (Triangles). *Let the edges $\{v_1, v_2\}$, $\{v_1, v_3\}$ and $\{v_2, v_3\}$ form a triangle in G. Additionally, let $U_1 \subset V$ such that $\{\{v_1, v_2\}, \{v_1, v_3\}\} \subseteq \delta(U_1)$ and $U_2 \subset V$ such that $\{\{v_1, v_2\}, \{v_2, v_3\}\} \subseteq \delta(U_2)$.*

1) ***(introduced in [29])*** *If the two inequalities*

$$-w_{v_1v_2} - w_{v_1v_3} \geq \sum_{e' \in \delta(U_1) \setminus \{\{v_1,v_2\},\{v_1,v_3\}\}} |w_{e'}|$$

$$-w_{v_1v_2} - w_{v_2v_3} \geq \sum_{e' \in \delta(U_2) \setminus \{\{v_1,v_2\},\{v_2,v_3\}\}} |w_{e'}|$$

hold, there exists a cut $\hat{\delta}$ with maximum value with $\{v_1, v_2\} \notin \hat{\delta}$.

2) ***(introduced in [35])*** *If the two inequalities*

$$w_{v_1,v_2} + w_{v_1,v_3} \geq \sum_{e' \in \delta(U_1) \setminus \{\{v_1,v_2\},\{v_1,v_3\}\}} |w_{e'}|$$

$$w_{v_1v_2} - w_{v_2v_3} \geq \sum_{e' \in \delta(U_2) \setminus \{\{v_1,v_2\},\{v_2,v_3\}\}} |w_{e'}|$$

hold, there exists a cut $\hat{\delta}$ with maximum value with $\{u, v\} \in \hat{\delta}$.

3) ***(new)*** *If the two inequalities*

$$-w_{v_1v_2} + w_{v_1v_3} \geq \sum_{e' \in \delta(U_1) \setminus \{\{v_1,v_2\},\{v_1,v_3\}\}} |w_{e'}|$$

$$-w_{v_1v_2} + w_{v_2v_3} \geq \sum_{e' \in \delta(U_2) \setminus \{\{v_1,v_2\},\{v_2,v_3\}\}} |w_{e'}|$$

hold, there exists a cut $\hat{\delta}$ with maximum value with $\{v_1, v_2\} \notin \hat{\delta}$.

Proof. For the proof of 1) see [29]. For 2) [35] present a proof, but we suggest a more compact one based on Proposition 3.2, which also shows the correctness of our new implication, 3):

If we apply the technique from Proposition 3.2 to G, by choosing $\delta^* = \delta(v_1)$ for the transformation, resulting in G' and 1) holds for G', we see that there is maximum cut $\hat{\delta}'$ in G' with $\{v_1, v_2\} \notin \hat{\delta}'$ and therefore, because of Proposition 3.2, a maximum cut $\hat{\delta}$ in G with $\{v_1, v_2\} \in \hat{\delta}$. But for 1) to hold in G', 2) needs to hold in G proving the correctness of 2). Following the same pattern, 3) can be derived from 2). Just choose $\delta^* = \delta(v_2)$ for the transformation from G' to G'' and 2) holds in G'' iff 3) holds in G. ◀

As for Proposition 3.3, Proposition 3.5 also extends into data reduction rules with Observation 3.1 and Proposition 3.2. We again follow [35] who consider $\{v_1\}$ and $\{v_2, v_3\}$ for U_1 and $\{v_2\}$ and $\{v_1, v_3\}$ for U_2 in their implementation.

4 Vertex Separator Based Reduction

Vertex separators of size one split the graph into its biconnected components, which can be solved independently from each other. Vertex separators of larger sizes also allow for preprocessing to be effective, as we will show in this section. Some of the data reduction rules for MaxCut discussed in the literature already make implicit use of vertex separators. We introduce a generalized framework, show how and which new and existing rules can be derived through it, and formalize all cases for which the rules derived from our framework are valid.

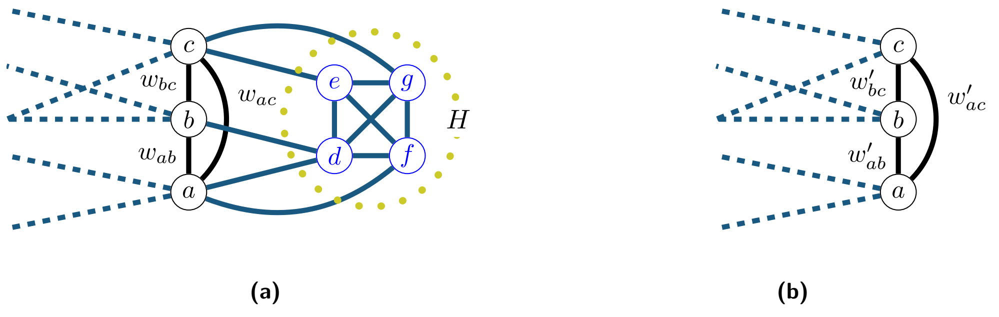

Figure 1 Example of a reduction rule of type vertex separator. (a) Original graph G with vertex separator $S = \{a, b, c\}$ separating H from the rest of the graph. (b) The resulting reduced graph G' with adapted edge weights, i.e., H has been deleted from the remaining graph clearly reducing the size of the instance.

The Vertex Separator Framework

We consider vertex separator-related data reduction rules, captured by the following definition. See Figure 1 for an example.

▶ **Definition 4.1** (Data reduction rule type SEPARATOR). *For a vertex separator S in $G = (V, E, w)$ separating $H \subset V$ from the rest of the graph, a rule is of type SEPARATOR, if it reduces G to $G' = G[V \setminus H]$, effectively deleting all vertices in H from G. All edges keep their original weight, apart from those in $G'[S]$.*

Note: For simplicity we assume $G[S]$ to always be fully connected. If this is not the case, the missing edges can be added with a weight of zero (as this is a MAXCUT-valid data transformation with $\beta = 0$). Edges in $G'[S]$ (may) get their weights updated, to encode the MAXCUT values of $G[H \cup S]$ for all possible bipartitions of S. The goal is to ensure Condition 2.1 for every bipartition P of vertices in $G'[S]$, i.e., the maximum cut value in $G'[S]$ needs to be the same (plus some constant β) as the one in $G[H \cup S]$ when fixing the partition of the vertices in P.

Hüffner et al. [23] developed a concept for the signed graph balancing problem, which is related to Definition 4.1. In their work, a set of vertices, only connected to the graph by a separator, gets replaced by a gadget. This introduces new vertices and edges to the graph. Our rules of type SEPARATOR only remove and never add vertices. In the following we provide a sufficient condition, that allows us to safely apply data reduction rules based on vertex separators.

▶ **Theorem 4.2.** *Let S be a vertex separator in $G = (V, E, w)$ separating $H \subset V$ from the rest of the graph, and $\mathcal{P}_S$ be the set of all possible bipartitions of vertices in S. Then a data reduction rule of type SEPARATOR is valid for G if the system of equations*

$$\sum_{e \in \delta_{G[S]}(P)} (w_e + \gamma_e) + \beta = \Delta(G[H \cup S], P) \quad \forall P \in \mathcal{P}_S$$

has a feasible solution for the variables γ_e and β.

Proof. Let $G' = G[V \setminus H]$ and all edges in $G'[S]$ get their weight updated by adding the corresponding γ_e. This transformation fulfills Condition 2.1 and is a valid data reduction as:

1) For any bipartition $(U', \bar{U'})$ in G' with MaxCut value c' we can find a bipartition $(U, \bar{U})$ in G with a cut value of $c \geq c' + \beta$, because by construction of G' combining $(U', \bar{U'})$ with the optimal partitioning of the vertices in $V(G) \setminus V(G')$ we get one for G with a cut value of exactly $c' + \beta$.

2) For any bipartition $P = (U, \bar{U})$ in G with MaxCut value c we can make sure we can map it to a bipartition $(U', \bar{U'})$ in G' with a cut value $c' + \beta \geq c$, as again by construction of G' the bipartition resulting from removing all vertices from P which are not in G' results in a bipartition whose cut value in G' is at least as high as the one of P in G. ◀

Small Separators

We now explicitly consider small vertex separators of size $k = 2$ and $k = 3$ and derive concrete and valid rules of type Separator. These two rules are known in theory, where they have been introduced in the context of polynomial algorithms for K_5 and $K_{3,3}$ minor-free graphs [4, 10]. To the best of our knowledge, they have not been considered for general graphs before and also have not been used in practice up to now. We start with separators of size 2:

▶ **Corollary 4.3.** *Let $G = (V, E, w)$ be a graph and $S = \{a, b\}$ a 2-separator in G separating $H \subset V$ from the rest of the graph. For more compact notation define $\tilde{H} := H \cup S$. Then the data reduction rule of type Separator is valid with the following values: $\beta = \Delta(G[\tilde{H}], (\{a, b\}, \emptyset))$ and $\gamma_{ab} = \Delta(G[\tilde{H}], (\{a\}, \{b\})) - \Delta(G[\tilde{H}], (\{a, b\}, \emptyset)) - w_{ab}$.*

Proof. There are two possible bipartitions for vertices in S and $G[S]$ contains one edge ($e = \{a, b\}$). Therefore the equation system of Theorem 4.2 gives two equations and two variables.

$$\beta = \Delta(G[\tilde{H}], (\{a, b\} : \emptyset)) \qquad\qquad w_{ab} + \gamma_{ab} + \beta = \Delta(G[\tilde{H}], (\{a\} : \{b\}))$$

Solving the system for β and γ_{ab} yields the stated equations. ◀

This general concept for vertex separators of size 2 covers some existing rules, like rule 2 and 6 of [14] as special cases. Vertex separators of size 3 allow for a generalized data reduction rule in a similar way:

▶ **Corollary 4.4.** *Let $G = (V, E)$ be a graph and $S = \{a, b, c\}$ a 3-separator in G separating H from the rest of the graph. For more compact notation define $\tilde{H} := H \cup S$. Then the data reduction rule of type Separator is valid with the following constant offset β and values for γ:*

$$\begin{aligned}
c_0 &:= \Delta(G[\tilde{H}], (\{a, b, c\} : \emptyset)) & \beta &= c_0 \\
c_1 &:= \Delta(G[\tilde{H}], (\{a, b\} : \{c\})) & \gamma_{ac} &= 1/2 \cdot (c_1 + c_3 - c_0 - c_2) - w_{ac} \\
c_2 &:= \Delta(G[\tilde{H}], (\{a, c\} : \{b\})) & \gamma_{ab} &= 1/2 \cdot (c_2 + c_3 - c_0 - c_1) - w_{ab} \\
c_3 &:= \Delta(G[\tilde{H}], (\{b, c\} : \{a\})) & \gamma_{bc} &= 1/2 \cdot (c_1 + c_2 - c_0 - c_3) - w_{bc}
\end{aligned}$$

Proof. There are four possible bipartitions for vertices in S and $G[S]$ contains three edges. Therefore the equation system of Theorem 4.2 gives four equations and four variables. The constants $c_0, \ldots, c_{|\mathcal{P}|}$, introduced for better readability, represent the right hand sides of the equation system. Solving this system for the constant offset β and the edge weights gives the presented equations. ◀

Separators of Arbitrary Size

For separators larger than three the above techniques only work in specific cases, as the number of potential bipartitions outgrows the number of possible edges in $G[S]$, resulting in an overdetermined system of equations. For a 4-separator we already get eight partitions (resulting in eight equations), but only seven variables (six for edges and β). Nevertheless for graphs with unit weights special rules have been described, for which their proof can be simplified through the lens of our framework and a new and more general one can be deduced. This new rule generalizes rule 1 and extends 5 and 7 from Ferizovic et al. [14] into a less restrictive and therefore more often applicable rule.

▶ **Proposition 4.5** (Same Neighborhood Clique). *Let $G = (V, E, w)$. If for a clique $C \subset V$ in G, for which $w_e = 1$ for all edges e incident to at least one vertex in C, $|C|+1 \geq |N(C)| \geq 1$ and $N(C) = N(u) \setminus C \; \forall u \in C$, removing vertices in C from the graph and updating weights of edges between vertices in $N(C)$ is a valid data reduction of type* Separator*. The constant offset is $\beta = \lfloor(|N(C)| + |C|)/2\rfloor * \lceil(|N(C)| + |C|)/2\rceil$. All weights of edges in $G[N(C)]$ get reduced by 1.*

Proof. The vertex set $S := N(C)$ forms a vertex separator of G with C on one side. Recall, we assume w.l.o.g. $G[S]$ to be fully connected. Consider the equations from Theorem 4.2: Subtracting the weight of edges contributing to the value of the cut from both sides, leaves $\sum_{\delta_{G[S]}(P)}(\gamma_e) + \beta$ as the left-hand side. The resulting right-hand side $\Delta(G[S \cup C], P) - \sum_{\delta_{G[S]}(P)}(w_e)$ can be interpreted as the value of the maximum cut in G': $\Delta(G', P)$, where G' is $G[S \cup C]$, except all edges incident to two vertices in S have a weight of 0. Because of $|S| \leq |C| + 1$, no matter how vertices in $G'[S]$ get partitioned, $\Delta(G', P)$ can always be maximized by partitioning vertices in $S \cup C$ into two partitions, whose sizes differ by at most one. Therefore, for the partial bipartition $P = (S, \emptyset)$, the value of $\Delta(G', P)$ is $\beta = \lfloor(|S| + |C|)/2\rfloor * \lceil(|S| + |C|)/2\rceil$. For general P this value has to be reduced by the number of edges cut in $G'[S]$, as they have a weight of 0 in G': $\beta - |\delta_{G[S]}(P)|$. With $\gamma_e = -1$ this is equal to the left-hand side of the equation $(\beta + \sum_{\delta_{G[S]}(P)}(-1))$. ◀

If $G[N(C)]$ is fully connected with weight 1 edges, then $G[C \cup N(C)]$ forms a clique and thus Proposition 4.5 leads to the same reduction as rule 1 of [14]. Rule 5 and 7 of their work have the same type of clique as their nucleus, but only allow the removal of vertices from C until $|C| = |N(C)|$. Candidates for the rule resulting from Proposition 4.5 can be found fast exactly as described by Ferizovic et al. [14].

Misc. Of the seven rules of Ferizovic et al. [14], two (rule 3 and 4) do not exactly fit into our Separator framework. Their rule 4 removes an edge from a clique, which creates candidates for Proposition 3.4 and their rule 3 adds an edge to the graph, which might lead to candidates for Proposition 4.5, hence we restate them here:

▶ **Proposition 4.6** (Near Clique −). *Let $G = (V, E, w)$ and $C \subseteq V$ be a clique where all edges connecting two vertices in C have weight 1. Define $C_{in} \subseteq C$ as the set of vertices with neighbors in C only. If $|C|$ is odd or $|C_{in}| > 2$, removing one edge connecting vertices in C_{in} is a valid data transformation with $\beta = 0$.*

▶ **Proposition 4.7** (Near Clique +). *Let $G = (V, E, w)$ and $C \subseteq V$ be a subgraph missing only one edge $e = \{u, v\}$, where all edges connecting two vertices in C have weight 1. Define $C_{in} \subseteq C$ as the set of vertices with neighbors in C only. If $u, v \in C_{in}$ and $|C|$ is odd or $|C_{in}| > 2$, adding e is a valid data transformation with $\beta = 0$.*

5 Algorithmic Framework

We now present the algorithm we derived from the existing and our new data reduction rules presented in the preceding sections. The algorithm consists of two components, decomposition and data reduction.

Decomposition

Let G be the input graph for our algorithm. We keep a queue of tuples (g, l). The first element of the tuple is a subgraph of G and the second is a lower bound on the size of any vertex separator in g. In the beginning, we insert the tuple $(G, 0)$ into the queue. We pop elements (g, l) one by one from the queue. We first check if MAXCUT is easy to solve on g (for example because g is small or has a special structure) and if so we solve g immediately. Otherwise if $l \leq 1$ we first check if g is (bi-)connected. If not we compute the (bi-)connected components and add them to the queue with the lower bound on vertex separators of $l + 1$. If neither of these steps worked we attempt to reduce the graph as much as possible using the core data reduction algorithm.

In more detail, the outer / decomposition part of our algorithm removes a tuple (g, l) from the queue and checks in order:

- If g has $\leq k$ vertices, we calculate $\Delta(g)$ by complete enumeration right away. If all edges of g have weight 1, we make use of a linear time algorithm (see Appendix A.2 for details), which either outputs an optimal MAXCUT value or reports that the special structure it is designed for is not present. In the latter case, we just continue.
- If $l = 0$ we test whether or not the graph is connected and add each connected component g_i as the tuple $(g_i, 1)$ to the queue.
- If $l = 1$ we test whether or not the graph is biconnected and add each biconnected component g_i to the queue, as the tuple $(g_i, 2)$.
- If the graph is at least biconnected, we apply the core data reduction algorithm.

Data Reduction

The core data reduction algorithm is based on all data reduction rules presented earlier:

i If the graph has unit weights we first reduce based on the data reduction rules of Ferizovic et al. [14] designed for this special case and our new one (Proposition 4.5).

ii For the resulting graph we apply: The dominating-edge rule from Proposition 3.3, the rule for similar vertices from Proposition 3.4, and all three triangle rules of Proposition 3.5, including our new one. Additionally, we use the SEPARATOR rule for vertex separators of size 3 (Corollary 4.4) by deleting vertices of degree three and updating the edge weights between their neighbors.

iii Finally we turn to our rule of type SEPARATOR for 2-separators: We calculate the SPQR-tree of the remaining graph to remove small subgraphs based on Corollary 4.3. Leaves of the SPQR-tree can be contracted recursively into the edge of the 2-separator, by calculating the solutions for *two* MAXCUT problems on the leaf. For finding exact solutions we again use our enumeration algorithm and therefore only apply this for SPQR-tree leaves of size $\leq k$.

Note: The restriction to only process small ($\leq k$) leaves of the SPQR-tree is necessary, to guarantee the data reduction is feasible (has polynomial runtime in the input size). Between any of the above steps, we make sure the graph is still (bi-)connected. If it loses its (bi-)connectivity, we add the (bi-)connected components to the queue and break. The procedure could be augmented to search for general 3-separators, but the quadratic running time of finding 3-separators [26] might not be suitable for big graphs, so we restrain the algorithm to vertices of degree three here.

To make steps (i) and (ii) efficient, we employ queues and markers to keep track of vertices for which the neighborhood has changed and who might be candidates for one of the data reduction rules. Similar techniques have been suggested by e.g. Ferizovic et al. [14].

6 Computational Experiments

We extensively evaluate the practical impact of our new techniques in multiple ways and compare them against the state-of-the-art. First, we showcase the effectiveness of our rules, measured by the number of vertices and edges removed from the original graph. Second, we highlight the usefulness of our new rules for solving MaxCut in practice. We show that our rules not only result in smaller graphs than the current state-of-the-art, but also help to further speed up an exact state-of-the-art solver and improve the solution quality of a state-of-the-art heuristic. Our rules are therefore not only effective but also efficient. Finally, we perform an ablation study, investigating the influence each of our new components has.

Setup and Solvers. The system we employed for the experiments is equipped with an AMD Ryzen Threadripper 3960X CPU, 128GB of RAM, and has Ubuntu 22.04. as its operating system. Our code is written in C++20, compiled with GCC 11.3. and the “-O3” flag. We make use of multiple open-source libraries, especially NetworKit [38] and OGDF [9] for graph algorithms and simexpal [2] for our experiments. On this system our naive enumeration algorithm takes clearly below one second to calculate the MaxCut value for graphs up to 21 vertices, therefore we set k to 21 in our algorithmic framework (see Section 5).

We use Gurobi 10 for experiments with an exact solver, as recent benchmarks by Hans Mittelmann [33] show Gurobi to be the solver, solving the second most instances to optimality before reaching the timelimit. The only solver with a better result is the non-publicly available QuBowl [35]. To allow Gurobi to efficiently solve MaxCut, we convert the problem to an unconstrained binary quadratic problem, which is a well-known transformation (see e.g. [6]). We restrict Gurobi to the use of one thread and set the MIP-Gap to 10^{-6}. To make the comparison more interesting, we also give Gurobi a hint on the reducibility of the instances, by setting the presolve parameter to *aggressive*.

For experiments with a state-of-the-art heuristic we use the implementation of Dunning et al. [12] of the algorithm of Burer et al. [7]. This algorithm did not only perform very well in the study of Dunning et al. [12], but is also employed in the state-of-the-art solver QuBowl [35].

For all our benchmarks, we run the same experiment with multiple different random seeds. This is to compensate for performance variability [31]. The seed is used to shuffle and relabel all vertices, as this change can lead to different orders of data reduction rule applications and therefore the resulting graphs differ. In experiments involving Gurobi, the seed is also passed to the solver.

Instances. We collected all instances from the most recent purely data reduction dedicated study we are aware of by Ferizovic et al. [14]. We also added publicly available instances Rehfeldt et al. [35] report results of their preprocessing on, for which preprocessing is effective, but the remaining graph does not become extremely small. The resulting set has no instances with $10\,000 \leq |V| \leq 200\,000$. We fill this gap with graphs of the same structure and from the

Table 1 Characteristics of the benchmark instances considered in our study. The columns $\underline{d}$ and $\overline{d}$ capture the min and max degree, $\underline{w}$ and $\overline{w}$ show the min and max edge weight.

set	instance	$\|V\|$	$\|E\|$	$\underline{d}$	$\overline{d}$	$\underline{w}$	$\overline{w}$
easy	soc-firm-hi-tech	33	91	1	16	1	1
easy	g001207	84	149	1	5	1	100 000
easy	g000981	110	188	2	6	1	100 000
easy	ENZYMES_g295	123	139	1	5	1	1
easy	g000292	212	381	2	4	5	13
easy	g000302	317	476	1	4	5	13
easy	ca-netscience	379	914	1	34	1	1
easy	bio-diseasome	516	1 188	1	50	1	1
easy	rt-twitter-copen	761	1 029	1	37	1	1
easy	g001918	777	1 239	1	4	5	13
easy	imgseg_271031	900	1 027	1	518	93 839 059	285 968 046 836
easy	road-euroroad	1 174	1 417	1	10	1	1
easy	imgseg_35058	1 274	1 806	1	587	-55 510 850 118	112 271 093 673
easy	bio-yeast	1 458	1 948	1	56	1	1
easy	imgseg_106025	1 565	2 629	1	902	93 981 365	136 834 528 589
easy	ca-CSphd	1 882	1 740	1	46	1	1
easy	ego-facebook	2 888	2 981	1	769	1	1
easy	imgseg_105019	3 548	4 325	1	2 753	109 623 218	236 593 516 427
easy	imgseg_374020	5 735	8 722	1	2 213	-46 639 208 299	407 957 172 555
medium	web-google	1 299	2 773	1	59	1	1
medium	inf-power	4 941	6 594	1	19	1	1
medium	ca-Erdos992	5 094	7 515	1	61	1	1
medium	g000677	17 127	27 352	1	4	1	126
medium	g001075	27 019	39 407	1	4	1	228 668
medium	imgseg_147062	28 552	65 453	1	925	-1 567 963 186	67 209 950 110
medium	g000087	38 418	71 657	2	4	1	198
medium	road-luxembourg-osm	114 599	119 666	1	6	1	1
big	web-Stanford	281 903	1 992 636	1	38 625	1	1
big	ca-MathSciNet	332 689	820 644	1	496	1	1
big	web-it-2004	509 338	7 178 413	1	469	1	1
big	ca-coauthors-dblp	540 486	15 245 729	1	3 299	1	1
big	ca-IMDB	896 305	3 782 446	1	1 590	1	1
big	inf-road_central	14 081 816	16 933 413	1	8	1	1
torus	t2g10*	100	200	4	4	-294 541	301 004
torus	t2g15*	225	450	4	4	-294 541	375 001
torus	t2g20*	400	800	4	4	-294 541	375 001
torus	t3g5*	125	375	6	6	-294 541	290 339
torus	t3g6*	216	648	6	6	-294 541	375 001
torus	t3g7*	343	1029	6	6	-298 103	375 001

*For every type of torus graph, there are three instances of the same size. Values are aggregated.

same source as Ferizovic et al. [14]. We group the instances in four sets, based on their type, size, and difficulty. Instances from the "easy" set can be solved in (sometimes significantly) less than ten seconds by the state-of-the-art exact solver Gurobi and are mainly included for comparability to earlier studies (they form the "medium" set of [14] and stem from [36] and [12]). Instances from our set "medium" require roughly ten seconds or more to solve by Gurobi and contain two of the "hard" instances of [14] as well as instances from the Network Repository [36] and the MQLib [12]. Large instances, too challenging for Gurobi as an exact solver, get assigned to the "big" set. The torus instances [42] from statistical physics get their own category, as these graphs have grid structure and therefore differ heavily from the real-world graphs in other sets. Table 1 summarises our set of benchmark instances.

6.1 Effectiveness and Efficiency

We start off by comparing the effectiveness of our data reduction algorithm with the current state-of-the-art in Table 2. In the following, we choose our implementation of the state-of-the-art algorithm of Rehfeldt et al. [35] as the baseline, because: 1) we wanted to rule out side effects, resulting from applying rules in different orders and other implementation details and 2) the only publicly available data reduction code for MaxCut by Ferizovic et al. [14] is not competitive with our state-of-the-art baseline (see Appendix A.3 for details).

Table 2 Average effectiveness of our implementation of the state-of-the-art (**sota**) preprocessing of [35] and **our** new algorithm. Percentage of remaining vertices / edges, as well as average runtime in seconds (pr [s]). The **improvement** columns show, how many vertices / edges relative to the sota algorithm could be removed additionally.

	sota			**our**			**improvement**	
set	$\|V\|$ [%]	$\|E\|$ [%]	pr [s]	$\|V\|$ [%]	$\|E\|$ [%]	pr [s]	$\|V\|$ [%]	$\|E\|$ [%]
easy	10.20	13.73	0.02	**1.59**	**3.93**	0.02	84.44	71.35
medium	19.64	27.16	0.09	**8.95**	**14.65**	0.11	54.43	46.08
big	37.41	53.31	39.71	**30.30**	**47.57**	67.94	19.01	10.76
torus	78.32	85.02	0.00	**75.11**	**82.43**	0.00	4.10	3.04

Table 3 Runtime comparison on the “medium” instances. The runtimes refer to pure Gurobi (**gurobi**) and to Gurobi with our implementation of the state-of-the-art (**sota**) preprocessing and **our** preprocessing. tr [s] is the total runtime it took, to calculate the optimal solution. For sota and our, the remaining columns report the percentage of vertices / edges left after the preprocessing and pr [s] is the reduction runtime only. The speedup (**spd**) is **sota** total runtime divided by **our** total runtime. All values are averages over 5 seeds per instance. Timeout refers to more than 3600 s.

	gurobi	**sota**				**our**				**spd**
instance	tr [s]	$\|V\|$ [%]	$\|E\|$ [%]	pr [s]	tr [s]	$\|V\|$ [%]	$\|E\|$ [%]	pr [s]	tr [s]	
web-google	9.08	5.60	11.18	0.08	0.30	0.00	0.00	0.18	0.18	**1.70**
inf-power	9.25	16.15	23.47	0.02	8.23	5.49	11.40	0.02	2.58	**3.18**
ca-Erdos992	1152.01	14.92	36.43	0.01	667.82	10.48	33.41	0.01	397.62	**1.68**
g000677	58.07	22.04	28.94	0.03	26.68	7.51	11.73	0.03	7.85	**3.40**
g001075	45.08	11.38	15.93	0.04	17.35	1.73	2.95	0.04	1.67	**10.41**
imgseg_147062	2757.79	49.65	56.78	0.22	1855.78	27.04	32.96	0.24	804.97	**2.31**
g000087	timeout	32.99	38.08	0.05	1341.55	19.03	24.00	0.06	455.03	**2.95**
road-luxembourg-osm	40.19	4.39	6.51	0.27	23.14	0.32	0.72	0.28	1.75	**13.25**

Table 4 Solution value comparison on the “big” instances. The heuristic (**burer**) of Burer et al. [7] was run for 1800 s. When paired with the state-of-the-art (**sota**) or **our** preprocessing, the runtime was reduced by the time the preprocessing took (pr [s]). The bv column shows the best value found when no preprocessing is employed. bvi is the absolute improvement over pure **burer**. The remaining columns report the percentage of vertices / edges left after the preprocessing. All values are averages over 5 seeds per instance.

	burer	**sota**				**our**			
instance	bv	|V| [%]	|E| [%]	pr [s]	bvi	|V| [%]	|E| [%]	pr [s]	bvi
web-Stanford	1584791	52.15	61.47	15.30	13169	44.68	58.47	30.57	**15469**
ca-MathSciNet	600010	30.82	52.58	2.53	3347	22.38	46.14	4.28	**3856**
web-it-2004	4052029	4.11	4.94	2.56	1043	3.64	3.67	2.89	**1049**
ca-coauthors-dblp	8219970	70.31	85.77	35.69	6068	67.10	84.04	222.35	**6237**
ca-IMDB	3361135	46.74	87.18	10.07	25215	40.86	86.96	32.70	**39354**
inf-road_central	15349816	20.30	27.90	172.12	749600	3.12	6.15	114.88	**843518**

The effectiveness of our proposed algorithmic framework can clearly be seen, as we improve upon the state-of-the-art on every instance set. For the "easy" set the most significant improvement can be observed; On average only 3.93 % of edges remain after applying our data reduction algorithm. For 15 out of the 19 instances, our algorithm fully solves the input to optimality, resulting in an empty transformed graph. This is only the case for 8 instances with the sota algorithm. The torus set profits the least from preprocessing, because of the special structure of those graphs. Still, applying our rules does not increase the runtime and helps to remove some additional vertices / edges.

Next, we show the effect of our preprocessing on the runtime of an exact state-of-the-art solver. Instances in the "easy" set can be solved extremely fast by state-of-the-art solvers like Gurobi. They require 0.32 s with our implementation of the state-of-the-art preprocessing and 0.29 s with our preprocessing on average. For the "torus" set, the overall speedup over the state-of-the-art is about 1.5 %, which is to be expected considering the small difference in effectiveness. The "big" instances seem too challenging for exact solvers like Gurobi, hence we will consider this set separately.

Table 3 shows the results for the "medium" set. For all instances Gurobi is faster when external preprocessing is employed and the total solving time is the smallest if our algorithm reduces the input. Directly comparing the total runtime of the state-of-the-art approach and our algorithm, shows speedups of up to one order of magnitude: For road-luxembourg-osm the solving time decreases from 23.14 s to as little as 1.75 s, a speedup of 13x. Overall the additional time required by our new rules, easily makes up for it, when solving real-world MaxCut instances to optimality. To investigate the efficiency on the "big" set, we compare the best solution values the heuristic of [7] finds on these graphs with a timelimit of 1800 seconds. Table 4 shows the results. Again, our preprocessing strictly improves over the state-of-the-art. For every instance, the additional time spent in our preprocessing pays off. Even in the case of ca-coauthors-dblp, where the preprocessing is slower by quite a bit (leading to a reduced runtime of the heuristic in our setting), the best solution found is on average better by 169. Also, relatively small differences in objective values (e.g. for web-it-2004) can matter greatly, if the instances get within reach of exact solvers.

6.2 Ablation study

As our preprocessing algorithm turns out quite powerful, we also investigate the influence of each component in more detail. Although our data reduction algorithm is useful for exact and heuristic solving, we restrict the analysis in the following to the exact solving of the "medium" set.

Cliques and Triangles. The rule for cliques introduced in Proposition 4.5 is relevant for graphs, where all edges have the same positive weight. For these graphs, we experimented with turning off our rule and reverting to the version of Ferizovic et al. [14], as well as not applying rules for cliques of this type altogether. Similar to Ferizovic et al. in their experimental study we did not notice big differences in the sizes of the resulting graphs in general. Nevertheless, for web-it-2004 the setting has a significant impact. Without our generalized rule, 4.9 % of all edges remained after applying our full data reduction algorithm. If we also included our new clique rule, only 3.7 % of all edges remained, a 24 % improvement.

The new rule for triangles from Proposition 3.5 has a noticeable impact on the graph size and the solving time of Gurobi, as shown in Figure 2. Making use of all three rules for triangles helps to reduce 6 out of the 8 graphs even further (web-google is fully reduced anyways and for ca-Erdos992 triangles have no effect). Especially for the weighted g00 and imgseg instances, the rules tend to remove many edges. Also, including our new rule for triangles (”all-tri”) always improves over deactivating it individually (”wo3-tri”). Sometimes the benefit seems small, but we clearly observed diminishing returns for all triangle rules. When activating triangle rules, independent of the order, every additional rule yields less benefit. As it also requires very little code to implement an additional triangle rule, if one is there already, we strongly suggest implementing all of them (including our new one).

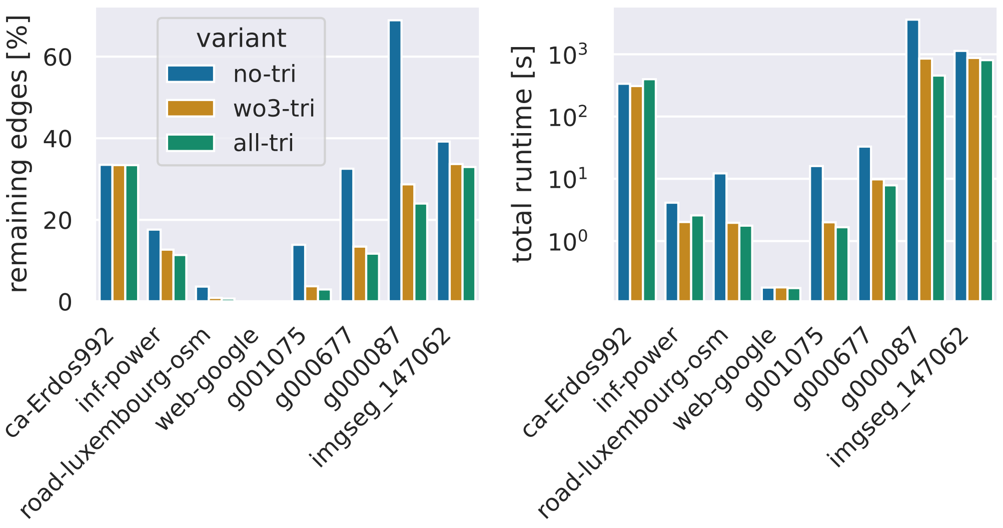

Figure 2 Effectiveness and efficiency of our full algorithm, with certain triangle-related rules turned on / off on the “medium” set. The variant with all triangle rules deactivated is **no-tri** and **all-tri** has all rules activated. In **wo3-tri** only our new rule is deactivated. Left: Relative number of remaining edges. Right: Total runtime in seconds (data reduction + Gurobi, logarithmic scaling). All values are averaged over five seeds per instance.

Vertex Separator. We now turn to our rules for 2- and 3-separators. To investigate their impact, we compare three settings of our algorithm in Figure 3. While most of the time the use of 2-separators only slightly reduces the size of the graph, also considering 3-separators has an impact on all graphs. We also see the total runtime (preprocessing + exact solving via Gurobi) does not always improve when 2-separators are considered; the extra effort for enumerating them is not always worth it. Our rule for 3-separators on the other hand, is not only extremely fast and effective but also attributes for much of the speedups over the state-of-the-art seen earlier in Table 2. E.g. for g000677 the total solving time decreases significantly, from about 24 s to 8 s, when 3-separators are also considered.

7 Conclusion

Our new vertex separator-based data reduction framework for MaxCut allows for the derivation of fast rules for separators of size 2 and 3 and additionally covers some known rules from the literature. The new separator-based rules and our new rule for triangles prove to be highly effective and efficient when paired with well-performing techniques from the literature and tested on established benchmarks. Even when purely employed as a preprocessing, our data reduction algorithm already fully solves 15 out of 19 instances from

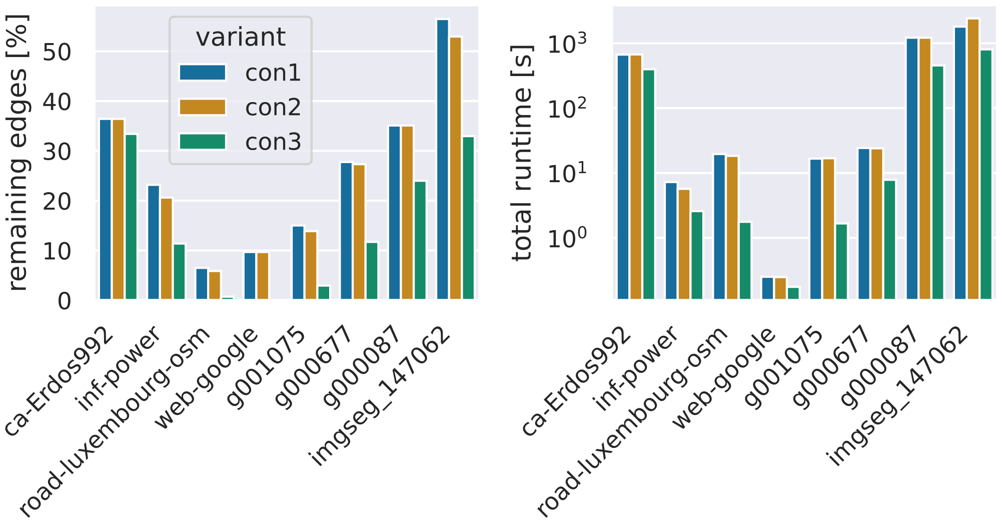

Figure 3 Effectiveness and efficiency of different settings of our algorithm: The full algorithm including rules for 2- and 3-separators is **con3**. The version of our algorithm, where only the rule for 3-separators is turned off is **con2**. For **con1**, the data reduction making use of 2-separators via the SPQR-tree decomposition is disabled as well. Left: Relative number of remaining edges. Right: Total runtime in seconds (data reduction + Gurobi, logarithmic scaling) for each instance of the "medium" set. All values are averaged over five seeds per instance.

a common benchmark data set to optimality. When paired with a state-of-the-art heuristic, our preprocessing helps to find strictly better solutions and when combined with an exact state-of-the-art solver we see solving times improving by up to an order of magnitude.

References

1 Faisal N. Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent advances in practical data reduction. In Hannah Bast, Claudius Korzen, Ulrich Meyer, and Manuel Penschuck, editors, *Algorithms for Big Data: DFG Priority Program 1736*, pages 97–133. Springer Nature Switzerland, Cham, 2022. `doi:10.1007/978-3-031-21534-6_6`.

2 Eugenio Angriman, Alexander van der Grinten, Moritz von Looz, Henning Meyerhenke, Martin Nöllenburg, Maria Predari, and Charilaos Tzovas. Guidelines for experimental algorithmics: A case study in network analysis. *Algorithms*, 12(7), 2019. `doi:10.3390/a12070127`.

3 Claudio Arbib. A polynomial characterization of some graph partitioning problems. *Inf. Process. Lett.*, 26(5):223–230, 1988. `doi:10.1016/0020-0190(88)90144-5`.

4 Francisco Barahona. The max-cut problem on graphs not contractible to K5. *Oper. Res. Lett.*, 2(3):107–111, August 1983. `doi:10.1016/0167-6377(83)90016-0`.

5 Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Oper. Res.*, 36(3):493–513, 1988. `doi:10.1287/opre.36.3.493`.

6 Francisco Barahona, Michael Jünger, and Gerhard Reinelt. Experiments in quadratic 0-1 programming. *Math. Program.*, 44(1-3):127–137, 1989. `doi:10.1007/BF01587084`.

7 Samuel Burer, Renato D. C. Monteiro, and Yin Zhang. Rank-two relaxation heuristics for MAX-CUT and other binary quadratic programs. *SIAM J. Optim.*, 12(2):503–521, 2002. `doi:10.1137/S1052623400382467`.

8 Jonas Charfreitag, Michael Jünger, Sven Mallach, and Petra Mutzel. Mcsparse: Exact solutions of sparse maximum cut and sparse unconstrained binary quadratic optimization problems. In Cynthia A. Phillips and Bettina Speckmann, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022*, pages 54–66. SIAM, 2022. `doi:10.1137/1.9781611977042.5`.

9 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W Klau, Karsten Klein, and Petra Mutzel. The open graph drawing framework (ogdf). *Handbook of graph drawing and visualization*, 2011:543–569, 2013.

10 Markus Chimani, Martina Juhnke-Kubitzke, Alexander Nover, and Tim Römer. Cut polytopes of minor-free graphs. *CoRR*, abs/1903.01817, 2019. `doi:10.48550/arXiv.1903.01817`.

11 George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–277, 1957. `doi:10.1287/opre.5.2.266`.

12 Iain Dunning, Swati Gupta, and John Silberholz. What works best when? A systematic evaluation of heuristics for max-cut and QUBO. *INFORMS Journal on Computing*, 30(3):608–624, 2018. `doi:10.1287/ijoc.2017.0798`.

13 Shimon Even and Robert Endre Tarjan. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975. `doi:10.1137/0204043`.

14 Damir Ferizovic, Demian Hespe, Sebastian Lamm, Matthias Mnich, Christian Schulz, and Darren Strash. Engineering kernelization for maximum cut. In Guy E. Blelloch and Irene Finocchi, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020*, pages 27–41. SIAM, 2020. `doi:10.1137/1.9781611976007.3`.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. `doi:10.1007/3-540-29953-X`.

16 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995. `doi:10.1145/227683.227684`.

17 R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. `doi:10.1137/0109047`.

18 Martin Grohe. Quasi-4-Connected Components. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.ICALP.2016.8`.

19 Carsten Gutwenger and Petra Mutzel. A linear time implementation of spqr-trees. In Joe Marks, editor, *Graph Drawing, 8th International Symposium, GD 2000, Colonial Williamsburg, VA, USA, September 20-23, 2000, Proceedings*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2000. `doi:10.1007/3-540-44541-2_8`.

20 F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM J. Comput.*, 4(3):221–225, 1975. `doi:10.1137/0204019`.

21 Dorit S. Hochbaum. Why should biconnected components be identified first. *Discret. Appl. Math.*, 42(2):203–210, 1993. `doi:10.1016/0166-218X(93)90046-Q`.

22 J.E. Hopcroft and R.E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973. `doi:10.1137/0202012`.

23 Falk Hüffner, Nadja Betzler, and Rolf Niedermeier. Separator-based data reduction for signed graph balancing. *J. Comb. Optim.*, 20(4):335–360, 2010. `doi:10.1007/s10878-009-9212-2`.

24 Yoichi Iwata and Takuto Shigemura. Separator-based pruned dynamic programming for steiner tree. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1520–1527. AAAI Press, 2019. `doi:10.1609/AAAI.V33I01.33011520`.

25 Michael Jünger, Elisabeth Lobe, Petra Mutzel, Gerhard Reinelt, Franz Rendl, Giovanni Rinaldi, and Tobias Stollenwerk. Quantum annealing versus digital computing: An experimental comparison. *ACM J. Exp. Algorithmics*, 26:1.9:1–1.9:30, 2021. doi:10.1145/3459606.

26 Arkady Kanevsky. Finding all minimum-size separating vertex sets in a graph. *Networks*, 23(6):533–541, 1993. doi:10.1002/net.3230230604.

27 Arkady Kanevsky and Vijaya Ramachandran. Improved algorithms for graph four-connectivity. *J. Comput. Syst. Sci.*, 42(3):288–306, 1991. doi:10.1016/0022-0000(91)90004-O.

28 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

29 Jan-Hendrik Lange, Bjoern Andres, and Paul Swoboda. Combinatorial persistency criteria for multicut and max-cut. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 6093–6102. Computer Vision Foundation / IEEE, 2019. doi:10.1109/CVPR.2019.00625.

30 Frauke Liers and G. Pardella. Partitioning planar graphs: a fast combinatorial approach for max-cut. *Comput. Optim. Appl.*, 51(1):323–344, 2012. doi:10.1007/s10589-010-9335-5.

31 Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. In *Theory driven by influential applications*, pages 1–12. INFORMS, 2013. doi:10.1287/educ.2013.0112.

32 S. Thomas McCormick, M. R. Rao, and Giovanni Rinaldi. Easy and difficult objective functions for max cut. *Math. Program.*, 94(2-3):459–466, 2003. doi:10.1007/s10107-002-0328-8.

33 Hans Mittelmann. Qubo benchmark. Website, 2023. Accessed on 2023-08-07. URL: https://plato.asu.edu/ftp/qubo.html.

34 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

35 Daniel Rehfeldt, Thorsten Koch, and Yuji Shinano. Faster exact solution of sparse maxcut and qubo problems. *Mathematical Programming Computation*, pages 1–26, 2023. doi:10.1007/s12532-023-00236-6.

36 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 4292–4293. AAAI Press, 2015. doi:10.1609/aaai.v29i1.9277.

37 Carsten Rother, Vladimir Kolmogorov, Victor S. Lempitsky, and Martin Szummer. Optimizing binary mrfs via extended roof duality. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society, 2007. doi:10.1109/CVPR.2007.383203.

38 Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: A tool suite for large-scale complex network analysis. *Netw. Sci.*, 4(4):508–530, 2016. doi:10.1017/NWS.2016.20.

39 R. Tamassia and G. Di Battista. Incremental planarity testing. In *30th Annual Symposium on Foundations of Computer Science*, pages 436–441, Los Alamitos, CA, USA, November 1989. IEEE Computer Society. doi:10.1109/SFCS.1989.63515.

40 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

41 K Wagner. Beweis einer Abschwächung der Hadwiger-Vermutung. *Mathematische Annalen*, 153(2):139–141, 1964.

42 Angelika Wiegele. Biq Mac Library – a collection of Max-Cut and quadratic 0-1 programming instances of medium size. http://biqmac.aau.at/biqmaclib.html, 2009.

43 Thomas Victor Wimer. *Linear Algorithms on K-Terminal Graphs.* PhD thesis, Clemson University, USA, 1987. AAI8803914.

A Appendix

A.1 Proof of Proposition 3.2

▶ **Proposition 3.2.** *Let G be an undirected weighted graph $G = (V, E, w)$ and δ^* any cut in G. Transforming G into $H = (V, E, w')$, by negating all weights of edges in δ^* is a valid data reduction with $\beta = \sum_{e \in \delta^*} w_e$. If an edge e is part of an optimal cut $\hat{\delta}_G$ and also of δ^*, there exists an optimal cut $\hat{\delta}_H$, which does not contain e.*

Proof. We first show, why the transformation from $G = (V, E, w)$ to $H = (V, E, w')$ with $\beta = \sum_{e \in \delta^*} w_e$ is a valid data transformation: As the only difference between G and H are edge weights, any cut in G is a cut in H and vice versa. The symmetric difference between all cuts in G and δ^* is a bijection from cuts in G to cuts in H. This bijection maps any cut with value c in G to one in H with the exact same value (because of the definition of β). Therefore $\Delta(G) = \Delta(H) + \beta$ and the transformation is valid. Let e be any edge in G part of an optimal cut $\hat{\delta}_G$ and also present in δ^*. By transforming G into H with δ^*, the symmetric difference of $\hat{\delta}_G$ and δ^* gives a cut with optimal value in H, which does not contain e. ◀

A.2 Solving certain unit weight graphs in linear time

In the decomposition part of our algorithm (see Section 5) we detect special graph structures, allowing for a linear-time algorithm for MaxCut. The following algorithm calculates $\Delta(G)$ for graphs G from two classes of graphs, for which all edges have the same positive weight, in time $O(n + m)$. The basic idea for the second case was described by Arbib [3]. 1) Bipartite graphs and 2) graphs containing a $K_{\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil}$ subgraph.

Proof. A check if a graph is bipartite can be implemented in $O(n + m)$ and using the bipartition as the MaxCut solution all edges are included in the cut, clearly making this the optimal solution.
If the graph is not bipartite we check if $m \geq \lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil$. If not the graph can not have a $K_{\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil}$ subgraph. Otherwise, we check for the existence of a $K_{\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil}$ subgraph by using the following observation about the connected components of the complement graph $\overline{G}$:
There is a $K_{\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil}$ subgraph in G iff there is a bipartition (V_1, V_2) of V with $|V_1| = \lfloor \frac{n}{2} \rfloor, |V_2| = \lceil \frac{n}{2} \rceil$ such that in $\overline{G}$ there is no edge connecting vertices from V_1 to vertices in V_2. We can check for the existence of such a bipartition, by solving the SubsetSum Problem. The input is a list of the sizes of every connected component in $\overline{G}$. We use the fact that SubsetSum reduces to Knapsack by making one item per given number x_i, which has $w_i = p_i = x_i$. Then we can use the dynamic programming algorithm for Knapsack by [11], which runs in time $\mathcal{O}(nC)$ where C is the capacity. As there are at most n inputs for the SubsetSum problem and our target sum is n this yields an $\mathcal{O}(n^2)$ algorithm to check if a graph has a $K_{\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil}$ subgraph. As this case is only relevant, when $m \in \Omega(n^2)$, the whole algorithm runs in $O(n + m)$. ◀

A.3 Additional Experiments

In our experiments in Section 6 we benchmark our new algorithm against our implementation of the state-of-the-art. To show the validity of this decision, we compare the preprocessing effectiveness of our implementation of the state-of-the-art and our new algorithm, with the publicly available code[1] of Ferizovic et al. [14] in Table 5. All experiments were conducted on the same machine (see Section 6). Because it was developed in parallel / earlier, the implementation of Ferizovic et al. [14] does not incorporate the rules of Lange et al. [29] and Rehfeldt et al. [35]. Hence, as expected, our state-of-the-art implementation (sometimes significantly) outperforms the older implementation in preprocessing effectiveness. Please note: Some results for FHLMSS reported here deviate from those in [14] for two main reasons. For the "imgseg" instances, their paper mentions a scaling from floating point to integer via multiplication with 10e6. In the publicly available code, values are scaled with 10e5. We opted for 10e6, for higher precision. For the "g00" instances, we could only reproduce the results of Ferizovic et al. [14] if unweighted versions of the original graphs are benchmarked. We use the original, weighted versions of all instances of type "g00", as this was also the case in other studies [8, 35]. For some rules, the implementation of Ferizovic et al. [14] uses gadgets, to transform weighted input graphs into unweighted ones. As a result, large edge weights may lead to excessive RAM usage (in our experiments more than 128 GB) and we were not able to collect results for the "torus" and some "g00" graphs. For these, our state-of-the-art implementation did not require more than 27 MB of main memory.

[1] We work with a public fork of the original code (`https://github.com/Amtrix/fpt-max-cut`), which offers some quality of life changes like a simplified compile step: `https://github.com/CharJon/fpt-max-cut`

Table 5 Effectiveness comparison of three different preprocessing implementations: Ferizovic et al. [14] (**FHLMSS**), our baseline implementation of Rehfeldt et al. [35] state-of-the-art (**sota**) preprocessing and **our** new algorithm. Percentage of remaining vertices ($|V|$ [%]) and edges ($|E|$ [%]), as well as runtime in seconds (pr [s]). The *na* entries indicate the implementation of **FHLMSS** required more RAM than available on the system (128 GB) and therefore the data is not available.

	FHLMSS			**sota**			**our**		
instance	$\|V\|$ [%]	$\|E\|$ [%]	pr [s]	$\|V\|$ [%]	$\|E\|$ [%]	pr [s]	$\|V\|$ [%]	$\|E\|$ [%]	pr [s]
soc-firm-hi-tech	63.64	73.63	0.00	63.64	73.63	0.00	0.00	0.00	0.06
g001207	*na*	*na*	*na*	0.00	0.00	0.00	0.00	0.00	0.00
g000981	*na*	*na*	*na*	0.00	0.00	0.00	0.00	0.00	0.00
ENZYMES_g295	13.82	23.74	0.00	0.00	0.00	0.00	0.00	0.00	0.00
g000292	96.70	98.16	0.05	21.70	21.05	0.00	0.00	0.00	0.00
g000302	79.50	86.13	0.08	0.00	0.00	0.04	0.00	0.00	0.00
ca-netscience	22.43	22.65	0.00	17.89	20.81	0.03	0.00	0.00	0.02
bio-diseasome	6.20	7.91	0.00	6.40	9.48	0.00	0.00	0.00	0.00
rt-twitter-copen	14.59	31.29	0.01	14.06	30.61	0.00	9.20	26.43	0.00
g001918	87.64	92.25	0.18	23.22	25.00	0.00	1.03	1.58	0.00
imgseg_271031	8.33	7.89	0.14	0.00	0.00	0.00	0.00	0.00	0.00
road-euroroad	20.78	32.60	0.01	15.91	25.58	0.00	8.40	17.87	0.00
imgseg_35058	68.21	62.02	0.30	4.16	7.32	0.00	0.00	0.00	0.00
bio-yeast	18.38	33.26	0.02	17.56	32.19	0.00	11.54	28.87	0.00
imgseg_106025	34.50	50.59	0.14	8.29	13.04	0.02	0.00	0.00	0.00
ca-CSphd	0.96	1.49	0.01	0.00	0.00	0.00	0.00	0.00	0.00
ego-facebook	0.42	0.97	0.41	0.00	0.00	0.00	0.00	0.00	0.00
imgseg_105019	62.18	58.57	4.44	1.05	2.20	0.08	0.00	0.00	0.08
imgseg_374020	21.12	29.04	1.86	0.00	0.00	0.15	0.00	0.00	0.15
web-google	12.32	20.77	0.01	5.60	11.18	0.08	0.00	0.00	0.18
inf-power	22.57	32.03	0.03	16.15	23.47	0.02	5.49	11.40	0.02
ca-Erdos992	15.47	37.41	0.05	14.92	36.43	0.01	10.48	33.41	0.01
g000677	89.27	92.98	10.79	22.04	28.94	0.03	7.51	11.73	0.03
g001075	*na*	*na*	*na*	11.38	15.93	0.04	1.73	2.95	0.04
imgseg_147062	76.28	83.50	0.13	49.65	56.78	0.22	27.04	32.96	0.24
g000087	99.55	99.76	83.73	32.99	38.08	0.05	19.03	24.00	0.06
road-luxembourg-osm	6.14	8.98	0.54	4.39	6.51	0.28	0.32	0.72	0.28
web-Stanford	76.10	94.02	42.04	52.15	61.47	15.30	44.68	58.47	30.57
ca-MathSciNet	31.68	53.38	5.29	30.82	52.58	2.53	22.38	46.14	4.28
web-it-2004	8.17	11.33	11.03	4.11	4.94	2.56	3.64	3.67	2.89
ca-coauthors-dblp	73.56	89.34	38.80	70.31	85.77	35.69	67.10	84.04	222.35
ca-IMDB	46.92	87.30	20.22	46.74	87.18	10.07	40.86	86.96	32.70
inf-road_central	27.45	37.42	350.18	20.30	27.90	172.12	3.12	6.15	114.88
t2g10_5555	*na*	*na*	*na*	55.00	65.50	0.00	47.00	58.50	0.00
t2g10_6666	*na*	*na*	*na*	64.40	74.10	0.00	62.00	73.00	0.00
t2g10_7777	*na*	*na*	*na*	66.00	75.50	0.00	58.00	70.00	0.00
t2g15_5555	*na*	*na*	*na*	61.96	72.84	0.00	54.76	66.40	0.00
t2g15_6666	*na*	*na*	*na*	63.11	72.44	0.00	55.11	66.00	0.00
t2g15_7777	*na*	*na*	*na*	57.24	68.44	0.00	48.53	60.62	0.00
t2g20_5555	*na*	*na*	*na*	63.00	73.12	0.00	59.50	70.62	0.00
t2g20_6666	*na*	*na*	*na*	63.25	72.28	0.00	55.50	66.12	0.00
t2g20_7777	*na*	*na*	*na*	62.75	73.33	0.00	58.75	70.00	0.00
t3g5_5555	*na*	*na*	*na*	93.60	97.87	0.00	93.60	97.87	0.00
t3g5_6666	*na*	*na*	*na*	95.20	98.40	0.00	95.20	98.40	0.00
t3g5_7777	*na*	*na*	*na*	95.20	97.87	0.00	95.20	97.87	0.00
t3g6_5555	*na*	*na*	*na*	93.52	97.53	0.00	93.52	97.53	0.00
t3g6_6666	*na*	*na*	*na*	95.83	98.46	0.00	95.83	98.46	0.00
t3g6_7777	*na*	*na*	*na*	92.59	97.07	0.00	92.59	97.07	0.00
t3g7_5555	*na*	*na*	*na*	96.50	98.64	0.00	96.21	98.45	0.00
t3g7_6666	*na*	*na*	*na*	95.92	98.64	0.00	95.92	98.64	0.00
t3g7_7777	*na*	*na*	*na*	94.75	98.25	0.00	94.75	98.25	0.00