# Solving Cluster Editing Using MaxSAT-based Techniques

Imko Marijnissen
Delft University Of Technology
Supervisor: Dr. Emir Demirović

**Abstract**

Clustering is a well-studied problem and several algorithms have been developed to find these clusterings under certain constraints. This work will show the applicability of propositional logic (MaxSAT) based approaches to a specific version of correlation clustering called cluster editing. This is the problem of finding the minimum number of edits to turn a given graph into a disjoint union of cliques. This work proposes a combination of modelling, preprocessing and solving techniques to solve the cluster editing problem via propositional logic. This approach's performance is experimentally evaluated and compared to another search algorithm based on merging vertices and adding/deleting edges. The results show that for certain instances propositional logic based approaches have a satisfactory performance while for other instances the theoretical approach provides a more guided search of the solution space.

## 1 Introduction

Cluster editing is the problem of provided with an undirected graph $G$, what is the minimum number of edge modifications (additions or deletions) to transform $G$ into a disjoint union of cliques (cluster graph)? Cluster editing is a NP-Hard problem but is fixed parameter tractable for the minimum solution size k [1]. There are several reasons as to why it is worthwhile solving cluster editing in an efficient manner. First of all, clustering can determine specific patterns and structures of the input data, this can provide insights into sociological aspects. Furthermore, cluster editing has been applied to the clustering of proteins [2] and other biological systems [3] in bioinformatics.

In this work the cluster editing problem will be related to the (unweighted partial) maximum satisfiability (MaxSAT) problem. That is, provided with a set of hard clauses (which are required to be satisfied) and a set of soft clauses (which are not required to be satisfied) what is the truth assignment that minimizes the number of falsified soft clauses? This is a well-studied problem within computer science with a solid theoretical foundation. Recently, the PACE 2021 challenge[1] has been introduced, a challenge to find efficient approaches for solving cluster editing. In this work state-of-the-art techniques in the MaxSAT field are applied to solve the cluster editing problem.

The work of Berg and Järvisalo [4][5] proposes three encodings to model an instance of correlation clustering as a MaxSAT instance. In this work, the applicability of the encodings for transforming a cluster editing problem instance into a MaxSAT problem instance is discussed. These encodings can be preprocessed using several preprocessing techniques developed for satisfiability problem instances, as described in [6][7][8][9][10]. These techniques are applied to the afforementioned encodings to increase the performance when solving. The

(preprocessed) encoding can then be given to a publicly available state-of-the-art MaxSAT solver such as MaxHS [11] [12] [13] [14] [15], EvalMaxSAT [16] or UWrMaxSat [17]. These solvers make use of a variety of techniques, e.g. linear algorithms, core-guided and hitting-set based approaches. The output of the solver for the encoding can then be used to construct a solution to the original cluster editing instance. However, while these components are well-studied, the aforementioned approach has not specifically been applied to the cluster editing problem. These techniques can potentially be leveraged to efficiently solve cluster editing instances.

This work proposes a combination of encodings, preprocessing techniques and MaxSAT solvers to solve cluster editing instances. An overview of the pipeline is shown in Figure 1.

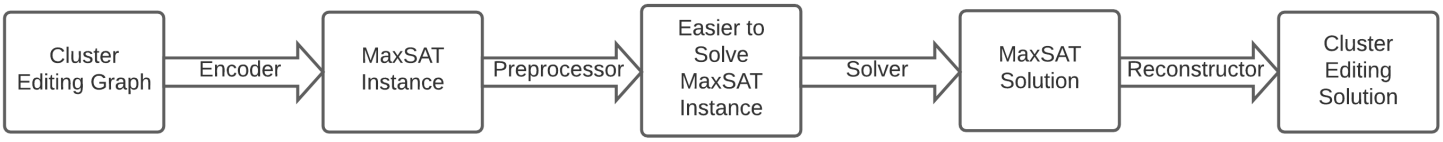



Figure 1: *Proposed Pipeline Overview*

The main research question is: Can state-of-the-art MaxSAT-based techniques be used to efficiently solve cluster editing? This question can be subdivided into several sub-questions:

1. How can the cluster editing problem be modelled as a MaxSAT problem instance? This modelling needs to take into account the different aspects of existing algorithms and preprocessing techniques to ensure compatibility.
2. What preprocessing steps will increase the performance of the cluster editing algorithm? It can occur that certain preprocessing steps are useful in the context of MaxSAT problems but are not applicable to the specific problem instances of the Cluster Editing problem thus introducing unnecessary overhead.
3. What MaxSAT solvers can be used to efficiently solve the instances created by the encoding? Does preprocessing increase the performance of the solver or is this dependent on the solver used? What solver performs the best on the selected encoding?
4. Under which circumstances do MaxSAT-based approaches outperform other Cluster Editing approaches? Is the performance of these approaches based on a specific structure of the Cluster Editing problem instance or are there other factors which contribute to a performance disparity between approaches?

The structure of this paper consists of firstly, a formal definition of the Cluster Editing problem and the MaxSAT problem. Secondly, existing work with regards to cluster editing approaches, encodings and pertinent preprocessing techniques are described. Thirdly, improvements to the encodings based on domain-specific knowledge and the reconstruction component of the pipeline are discussed. Finally, experimentation is performed to determine what encoding, preprocessing techniques and solvers perform most suitably. Followed by conclusions drawn based on the comparative performance between the optimal pipeline structure and a theoretical cluster editing approach.

## 2 Problem Description

In this section the notation is introduced and observations about the problem(s) are made. First, the notation for Cluster editing is provided. The input is provided as a graph $G = (V,\ E)$ where $V$ is a set of vertices and $E$ is a set of edges. An edge between node $i$ and node $j$ (where $i < j$) is signified by $e(i, j)$. The aim of a cluster editing algorithm is to generate a set of edges which when modified in the original input graph, will result in a cluster graph. Let us call this set of edges $S$, for an arbitrary edge $e(i, j) \in S$ there are two cases: If $e(i, j) \in E$ then applying S to G means the edge will be removed in the output graph, conversely, if $e(i, j) \notin E$ then applying S to G means the edge will be added to the output graph. In the unweighted variant of the problem, all edge additions or deletions will incur a cost of one signifying that there is no distinction between edges in terms of cost. In the weighted variant of cluster editing it is possible for edges to be associated with weights other than one, given by the function $w(e(i, j))$. The aim is then to minimize the sum of the weights of the edges contained in the solution S, more formally, the aim is to minimize $\sum_{e(i,j)\in S} w(e(i, j))$. An example of a cluster editing instance can be seen in Figure 2a.

Subsequently, the notation of the partial Maximum Satisfiability (MaxSAT) problem is formalized. An input instance to the partial MaxSAT problem consists of two sets, $F = (H, SC)$ where $H$ are the so-called hard clauses and $SC$ are the so-called soft clauses. The clauses are made up of a set of literals where each clause represents a disjunction of said literals. These literals take on a boolean value (either a zero or a one). The output of a MaxSAT solver is a truth assignment (commonly denoted by $\tau$) which maps each literal to one of the aforementioned values. Any truth assignment $\tau$ is required to satisfy all hard clauses, more formally, $(\forall C \in H)(\exists x \in C)$ such that $\tau(x) = 1$. If a clause is satisfied then $\tau(C) = 1$ and $\tau(C) = 0$ otherwise. The unweighted variant of partial MaxSAT aims to find the minimum number of falsified soft clauses, more formally, the aim is to minimize $|\{C \in SC \mid \tau(C) = 0\}|$. In the weighted variant of partial MaxSAT, each soft clause $C$ is associated with a weight $w$ and the aim is to minimize the sum of weights of the falsified clauses, more formally, the aim is to minimize $\sum_{\substack{(C,w)\in SC \\ \tau(C)=0}} w$. Since MaxSAT is a NP-Hard problem, it is possible to reduce other problems to MaxSAT. A simple example of an unweighted partial MaxSAT problem instance can be seen in Figure 2b


(a) Example of Cluster Editing Instance
Source: Taken from [1]

Hard Clauses: $\{(x \vee y \vee z), (\neg x \vee z), (\neg y \vee z)\}$
Soft Clauses: $\{(\neg y \vee \neg z), (\neg x \vee \neg y), (\neg z)\}$
Solution: $\tau(x) = \tau(y) = 0$ and $\tau(z) = 1$ inducing a cost of 1

(b) Example of MaxSAT Instance

Figure 2: *Problem Examples*

# 3 Existing Work

In this section existing work relevant to the research topic is discussed. This includes a survey of existing approaches, a description of the different encodings and a short definition of the preprocessing techniques pertinent to increasing performance.

## 3.1 Cluster Editing Approaches

This subsection briefly discusses other approaches developed for solving cluster editing. The most basic strategy is a naive branching strategy based on conflict triples which consists of splitting into three branches as seen in Figure 3. This approach results in a large search tree and does not account for structures found in the graph. Parameterized algorithms oftentimes make use of such search tree algorithms while utilizing the knowledge of the amount of edge modifications $k$. An example of such a strategy is [18] which provides a running time of $O(1.62^k + m + n)$. This strategy includes merging vertices under certain conditions, potentially transforming the graph into a weighted graph. Another approach is to formulate the problem as an Integer Linear Programming (ILP) problem based on [19]. An evaluation of a variety of strategies including parameterized and ILP approaches, can be found in [20]. This paper relies on the work of [21] which is based on [18]. This approach employs a heuristic to select which edge to branch on and a Branch and Bound approach is utilized to incrementally update a lower bound.

## 3.2 Encoding

In this subsection the first component of the pipeline is outlined: converting a cluster editing problem instance to a MaxSAT problem instance. Two different encodings are considered, the transitive encoding and the binary encoding as described in [5].

### 3.2.1 Transitive Encoding

The transitive encoding (section 6 of [5]) uses the concept of a conflict triple to ensure well-defined clusters. For a triplet of vertices $(i, j, k)$, if exactly two edges exist between the three vertices then the triplet is called a conflict triple. If there are no conflict triples in a graph then the graph is a cluster graph. An example of a conflict triple and how to resolve it can be found in Figure 3. The transitive encoding introduces a variable $x_{ij}$ for every pair of

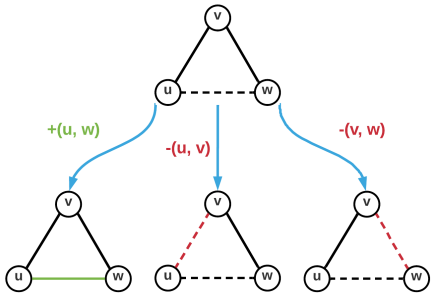



Figure 3: *How to resolve a conflict triple*
Source: *Adapted from [1]*

vertices where $i < j$. If $\tau(x_{ij}) = 1$ then $i$ and $j$ are assigned to the same cluster. Based on these variables and the aforementioned observation the following encoding is produced:

- **Hard clauses:**
  Based on the concept of conflict triples three clauses are generated for all $(i, j, k)$ where $1 \leq i < j < k \leq |V|$:

  - $((x_{ik} \wedge x_{jk}) \rightarrow x_{ij})$ which is equal to $(\neg x_{ik} \vee \neg x_{jk} \vee x_{ij})$
  - $((x_{ij} \wedge x_{jk}) \rightarrow x_{ik})$ which is equal to $(\neg x_{ij} \vee \neg x_{jk} \vee x_{ik})$
  - $((x_{ij} \wedge x_{ik}) \rightarrow x_{jk})$ which is equal to $(\neg x_{ij} \vee \neg x_{ik} \vee x_{jk})$

  This encodes the fact that if there are two edges between the three vertices then it is required that the missing edge is added thus eliminating the conflict triple.

- **Soft clauses:**
  For all $(i, j)$ where $i < j$ a singular clause is generated:

$$\begin{cases} x_{ij} & \text{if } \ e(i,j) \in G \\ \neg x_{ij} & \text{otherwise} \end{cases}$$

  This encodes the fact that a cost of one is induced if two vertices are connected in the output graph while not being connected in the input graph or if two vertices are not connected in the output graph while they are connected in the input graph.

This results in $3 \times \binom{|V|}{3}$ hard clauses and $\binom{|V|}{2}$ soft clauses with a total of $\binom{|V|}{2}$ variables.

### 3.2.2 Binary Encoding

The binary encoding (section 8 of [5]) is based on the observation that each vertex is required to be assigned to a cluster. What is essential to this encoding is that the cost function is faithfully represented. To this end, three types of variables are introduced to ensure these properties. First of all, the value of $k$ is defined as the smallest number such that $2^k \geq |V|$. A variable $b_i^a$ where $1 \leq i \leq |V|$ and $1 \leq a \leq k$ is then created. This represents the $a^{th}$ bit of the cluster assignment of vertex $i$, where $b_i^k ... b_i^1$ describes the binary number indicating the cluster assignment of vertex $i$. Subsequently, a variable indicateing that two vertices $i$ and $j$ have the same bit at position $a$ is created, $EQ_{ij}^a$ where $1 \leq i < j \leq |V|$ and $1 \leq a \leq k$. Lastly, a variable is created (similarly to the transitive encoding) for every pair of vertices $S_{ij}$, indicating whether vertex $i$ and $j$ are assigned to the same cluster. This variable will also be used to represent the cost of clustering two vertices together. Based on these variables, the following encoding is produced:

- **Hard clauses:**
  The meaning of $EQ_{ij}^a$ is encoded. This variable is true iff the bits of vertices $i$ and $j$ are the same at position $a$. Four clauses are created for all values of $(i, j, a)$ where $1 \leq i < j \leq |V|$ and $1 \leq a \leq k$:

  - $(\neg EQ_{ij}^a \wedge \neg b_i^a) \rightarrow b_j^a$ which is equal to $(EQ_{ij}^a \vee b_i^a \vee b_j^a)$
  - $(\neg EQ_{ij}^a \wedge b_i^a) \rightarrow \neg b_j^a$ which is equal to $(EQ_{ij}^a \vee \neg b_i^a \vee \neg b_j^a)$
  - $(EQ_{ij}^a \wedge b_i^a) \rightarrow b_j^a$ which is equal to $(\neg EQ_{ij}^a \vee \neg b_i^a \vee b_j^a)$
  - $(EQ_{ij}^a \wedge \neg b_i^a) \rightarrow \neg b_j^a$ which is equal to $(\neg EQ_{ij}^a \vee b_i^a \vee \neg b_i^a)$

Subsequently, the meaning of the variable $S_{ij}$ is encoded. This variable is true iff all of the bits of $i$ and $j$ are equal or false if a single bit is unequal. This can be represented with the following clauses for all values of $(i, j)$ where $1 \leq i < j \leq |V|$:

- For all values of $a$ where $1 \leq a \leq k$: $S_{ij} \rightarrow EQ_{ij}^{a}$ which is equal to $(\neg S_{ij} \vee EQ_{ij}^{a})$
- $\neg S_{ij} \rightarrow (\neg EQ_{ij}^{1} \vee ... \vee \neg EQ_{ij}^{k})$ which is equal to $(S_{ij} \vee \neg EQ_{ij}^{1} \vee ... \vee \neg EQ_{ij}^{k})$

- **Soft clauses:**
  For all $(i, j)$ where $i < j$ a singular clauses is generated:

$$\begin{cases} S_{ij} & \text{if } \ e(i,j) \in G \\ \neg S_{ij} & \text{otherwise} \end{cases}$$

This encoding results in a total of $(4 \times \binom{|V|}{2} \times \lceil \log_2 |V| \rceil) + (\binom{|V|}{2} \times \lceil \log_2 |V| \rceil + 1)$ hard clauses and $\binom{|V|}{2}$ soft clauses with a total of $((|V| \times \lceil \log_2 |V| \rceil) + (\binom{|V|}{2} \times \lceil \log_2 |V| \rceil) + \binom{|V|}{2})$ variables. Reducing the amount of variables by encoding $S_{ij}$ directly into the soft clauses is possible (at the "cost" of going from unweighted to weighted MaxSAT) but was experimentally evaluated to perform significantly worse.

### 3.2.3 Symmetry Breaking

As specified in [5], the binary encoding suffers from the issue of symmetry. For example, given a cluster assignment, shifting all of the cluster indices by one modulo the amount of clusters will result in a new assignment while the clustering remains the same. The following symmetry breaks are applicable:

- The first symmetry break is straightforward and involves $\lceil \log_2 |V| \rceil$ clauses with no extra variables: assigning the first vertex to cluster zero. Hence, for all $a$ where $1 \leq a \leq k$ the hard clause $(\neg b_1^a)$ is included.
- The second symmetry break is more involved and is especially applicable when there is a significant discrepancy between $2^k$ and $|V|$. If $|V|$ is not a power of two then it is possible for a cluster to be assigned to a number bigger than $|V|$ which is not desirable. Limiting the amount of clusters to $|V|$ rather than $2^k$ results in fewer symmetries. The derivation of the clauses can be found in appendix A.

## 3.3 Preprocessing Techniques

In this subsection the preprocessing techniques pertinent to the aforementioned encodings are discussed. This work utilizes the open-source preprocessor MaxPre [6]. MaxPre implements a variety of SAT and MaxSAT-specific techniques. For the sake of brevity, solely the applicable SAT techniques are discussed. To ensure applicability of these techniques to MaxSAT instances special precautions need to be taken, oftentimes involving labels.

- **Resolvent**: For two clauses $C_1 = (x \vee A)$ and $C_2 = (\neg x \vee B)$ the resolvent of the two clauses with regard to $x$ is as follows: $C_1 \otimes_x C_2 = (A \vee B)$
- **Blocked Clause Elimination** [22]: A literal $x$ will block a clause $C$ if the resolvent with all other clauses containing $\neg x$ is a tautology, it can then be shown that removing such a clause will result in an equi-satisfiable formula.

- **Bounded Variable Elimination** [8]: $F_x$ is the set of all clauses containing the literal $x$ and $F_{\neg x}$ is the set of all clauses containing the literal $\neg x$. Variable elimination of a literal $x$ consists of replacing $F_x$ and $F_{\neg x}$ with $\{C_1 \otimes_x C_2 \mid C_1 \in F_x,\ C_2 \in F_{\neg x},\ C_1 \otimes_x C_2$ is not a tautology$\}$. Variable Elimination is bounded if it only considers eliminating $x$ when $|F_x \otimes_x F_{\neg x}| \leq |F_x| + |F_{\neg x}|$.
- **Unit Propagation**: If there is a unit clause (meaning $|C| = 1$) then this clause can be propagated resulting in a simpler formula. As an example, consider the set of clauses $\{(x), (\neg x \vee y \vee \neg z), (x \vee \neg y), (y \vee z)\}$ since the first clause is a unit clause it can be propagated resulting in the set of clauses $\{(x), (y \vee \neg z), (y \vee z)\}$.

# 4 Algorithmic Ideas

This section will introduce certain improvements based on domain-specific knowledge, specifically reduction rules for cluster editing. Furthermore, this section describes the process of reconstructing a cluster editing solution from the output of a MaxSAT solver.

## 4.1 Improvements to the Encodings

In this subsection, the application of domain-specific knowledge to enhance the encodings of section 3.2 is discussed. There are specific rules regarding the clustering of two vertices which can be used to further improve the encoding. One of these rules is based on the neighbourhood of a vertex which states: If two vertices share at most a single neighbour and $e(i, j) \notin E$ then the two vertices are assigned to different clusters as proven in [23]. By adding a hard clause $(\neg x_{ij})$ (or $(\neg S_{ij})$ in the case of the binary encoding) for all vertices $i$ and $j$ for which this rule holds, it can be ensured that these vertices are not co-clustered. Moreover, this rule introduces the possibility to not only remove the soft clause associated with the forbidden pair but also to decrease the complexity of the hard clauses which contain this pair as described in the following paragraphs. A similar argument can be made for rules which force two vertices to be co-clustered. More reduction rules can be found in [24].

### 4.1.1 Simplifying Transitive Encoding

Simplifying the Transitive encoding results in less soft clauses, hard clauses and potentially reduces the size of the remaining hard clauses. For a triplet of vertices $(i, j, k)$ where $1 \leq i < j < k \leq |V|$, the following simplification is then derived:

- In case of forbidden pairs:
  - If more than a single pair is forbidden then all of the clauses are satisfied.
  - If a single pair is forbidden then a single clause is created specifying that one of the remaining pairs also cannot be co-clustered. For example, if the pair $x_{ij}$ is forbidden then this results in the clause $(\neg x_{ik} \vee \neg x_{jk})$
- In case of forced pairs:
  - If a single pair is forced then two clauses are generated ensuring that either all of the edges in the triplet are added or none of the non-forced edges are added. For example, if $x_{ij}$ is forced then this results in the clauses $\{(x_{ik} \vee \neg x_{jk}), (\neg x_{ik} \vee x_{jk})\}$
  - If two pairs are forced then a single clause is generated containing the non-forced pair. For example, if $x_{ij}$ and $x_{jk}$ are forced then this results in the clause $(x_{ik})$

#### 4.1.2 Simplifying Binary Encoding

Similarly, simplifying the binary encoding results in a smaller number of soft clauses, hard clauses and the size of the remaining hard clause associated with a forbidden or forced pair is reduced:

- In case of forbidden pairs:
  - If the pair is forbidden to be co-clustered then clauses of the form $(\neg S_{ij} \vee EQ^a_{ij})$ are removed since these clauses are satisfied. Furthermore, the clause of the form $(S_{ij} \vee \neg EQ^1_{ij} \vee ... \vee \neg EQ^k_{ij})$ can be transformed into $(\neg EQ^1_{ij} \vee ... \vee \neg EQ^k_{ij})$, this signifies that since the vertices are not co-clustered, it must be the case that $(\exists a)\ s.t.\ b^a_i \neq b^a_j$.
- In the case of forced pairs:
  - If the pair is forced to be co-clustered then the clauses of the form $(\neg S_{ij} \vee EQ^a_{ij})$ are transformed into the clause $(EQ^a_{ij})$ signifying that $i$ and $j$ are required to be assigned the same cluster. Furthermore, the clause of the form $(S_{ij} \vee \neg EQ^1_{ij} \vee ... \vee \neg EQ^k_{ij})$ is automatically satisfied.

### 4.2 Reconstruction

The aim of cluster editing is to find the edges which need to be modified. To this end, what clause represents the connection between a pair of vertices is recorded. Then, given an assignment for such a variable: If the variable is assigned true and $e(i,j) \notin E$ then the edge is added, similarly, if the variable is assigned false and $e(i,j) \in E$ then the edge is removed. In the case that neither of these two cases apply, no edge modification is necessary. The solution is then a set containing all of the aforementioned edges.

## 5 Experimental Study

### 5.1 Experimentation Setup

The experiments were performed under a Windows 10 environment using Ubuntu on an Intel Core i7-3770 3.4GHz CPU with 16GB of RAM. The memory limits used are the default limits imposed by the solvers (no issues were encountered in this regard). The time limit is specified in each Figure but was commonly set to 600 seconds.

The solver used for the final experimentation is MaxHS[1] Version 3.2 with CPLEX version 20.1.0.0. The preprocessor used is MaxPre[2], last updated on the 23rd of August 2019. For the experimentation in section 5.3, the solvers EvalMaxSAT and UWrMaxSat were taken from the MaxSAT Evaluation of 2020[3]. The source code of [21] is available on GitHub[4], last updated on the 18th of June 2021.

The benchmarks[5] that are used to evaluate the performance are part of released benchmarks for the PACE challenge. These benchmarks consist of a variety of structures, some are densely connected and others are loosely connected. If a graph consists of multiple connected components then these components can be solved separately.

[1] https://github.com/fbacchus/MaxHS
[2] https://github.com/Laakeri/maxpre
[3] https://maxsat-evaluations.github.io/2020/descriptions.html
[4] https://github.com/LHolten/Cluster-Editing
[5] https://fpt.akt.tu-berlin.de/pace2021/exact.tar.gz

As for the settings/flags that are used during the experimentation, the default settings of the preprocessor were maintained. The settings used for MaxHS were *"-no-printOptions -printSoln -verb=0"* while for the other solvers, the provided configurations were used. Experimentation with different settings was performed in an attempt to enhance the performance, however, preliminary parameter tuning did not exhibit a significant difference.

The experiments do not include the time necessary to generate the model nor the reconstruction. This is due to the fact that the current structure of the pipeline introduces a significant amount of overhead for writing the model to a file (on instance exact099, it takes ∼3s to generate the model and ∼40s to write to file). By integrating the model generation directly into the preprocessor and solver, this overhead would be eliminated.

## 5.2 Evaluation of Preprocessing

In this subsection the applicability of preprocessing techniques to the encodings defined in section 3.2.1 and section 3.2.2 is discussed. There is a vast difference between the two encodings hence techniques which are applicable to one of the encodings are not necessarily applicable to the other. When preprocessing the aforementioned encodings, it becomes apparent that there are non-obvious interactions between components.

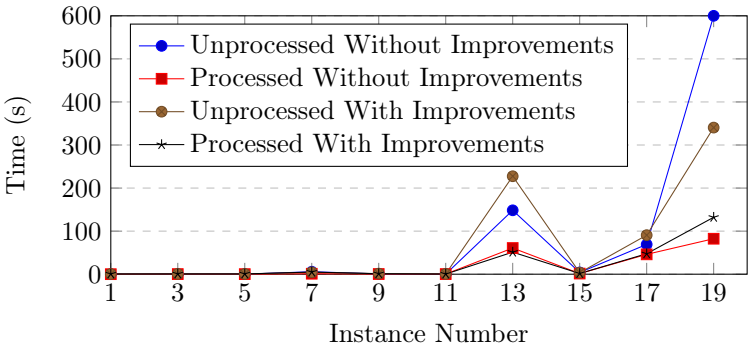
### 5.2.1 Preprocessing Transitive Encoding

Applying preprocessing on the original transitive encoding without improvements increases the performance of the solver used on this encoding but does not remove any clauses nor variables. This is the case when using the preprocessor with the default techniques, the number of variables and clauses is slightly increased due to the fact that MaxPre adds labels and performs certain operations on said labels. However, an increase in the number of clauses/variables does not directly correlate to a worse performance of the solver when compared to the performance on an unprocessed instance as can be seen in Figure 4a. This Figure shows that the processed transitive encoding performs better than when an unprocessed encoding is used. This is especially apparent for instance 19 on which the unprocessed encoding reaches the time limit while the processed encoding is solved within 100 seconds. For the smaller instances, the unprocessed variant performs slightly better due to the fact that there is less overhead involved than when using the preprocessor.

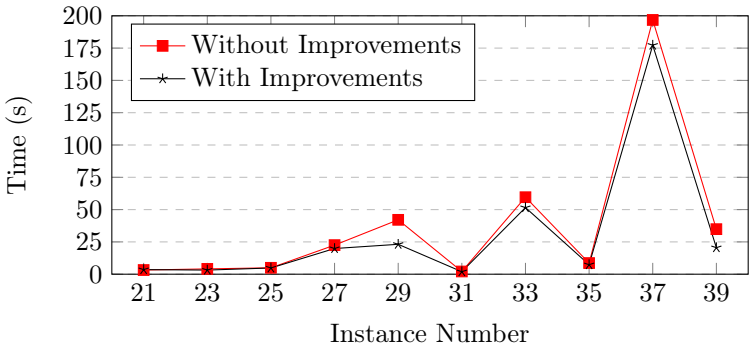
When applying the improvements based on section 4.1.1, the applicability of particular techniques changes. Due to the fact that certain hard clauses and soft clauses are removed, it is possible to apply blocked clause elimination as described in section 3.3. Applying this techniques results in a lower number of clauses. The performance gain depends on the number of hard clauses removed as a result of the simplification. The difference in performance between the processed and unprocessed encoding can be seen in Figure 4a. Once again, it can be observed that the processed variant of the encoding performs better than the unprocessed instance while containing more clauses/variables.

Furthermore, the processed transitive encoding with improvements performs better than the encoding without improvements. A comparison can be seen in Figure 4a and Figure 4b which show the time necessary to solve a certain instance by MaxHS. While the encoding without improvements has a marginally better performance on instance 19, this trend does not appear to continue. For all following instances, the encoding with improvements performs at an equal or better level than the encoding without improvements. This indicates that the improvements cause less strain for the MaxSAT solver due to the reduction in the

number of soft and hard clauses. In conclusion, the improvements generally ameliorate the performance but this is not the case for all instances.




(a) MaxHS, Unprocessed vs Processed Transitive Encoding With and Without improvements 1-19, 600 second timeout




(b) MaxHS, Processed Transitive Encoding With vs Without improvements 21-39, 600 second timeout

Figure 4: *Transitive Encoding Comparison between Processed and Unprocessed*

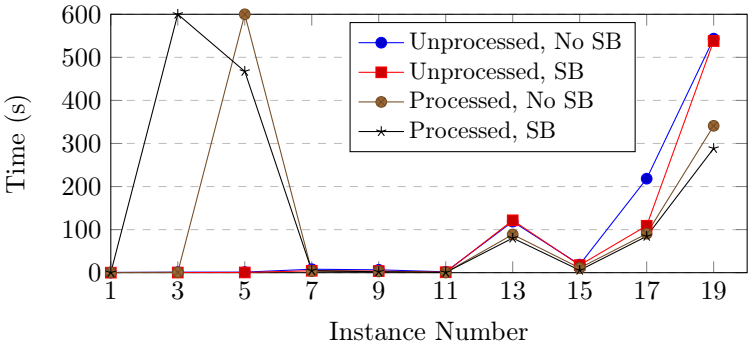
### 5.2.2 Preprocessing Binary Encoding

Different preprocessing techniques are applicable to the binary encoding dependent on whether the similarity breaking constraints of section 3.2.3 have been supplied and whether the improvements based on section 4.1.2 have been applied. First, the applicability without the improvements is discussed. In the case in which no symmetry breaking has been implemented, two techniques remove clauses namely blocked clause elimination and bounded variable elimination (as specified in section 3.3). Moreover, bounded variable elimination also removes certain variables rather than only clauses. If symmetry breaking has been implemented then the aforementioned techniques are similarly applicable, however, unit propagation becomes applicable and is able to remove both clauses and variables. While the formula with symmetry breaking includes more variables and clauses before preprocessing than the formula without symmetry breaking, after preprocessing the formula with symmetry breaking has less variables and (hard) clauses than its counterpart. The difference in performance can be seen in Figure 5a. This Figure shows that the processed encodings perform at an equal or better level than the unprocessed encodings from instance 7 onward. However, it can be observed that the processed encodings do not perform well on instance 3 and/or 5 for reasons which are unclear.

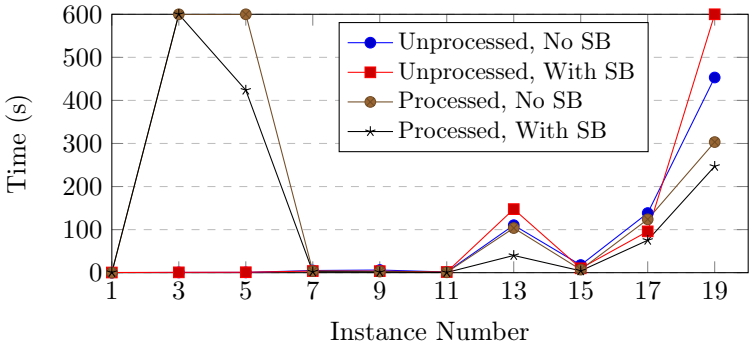
The effects of preprocessing if the improvements based on section 4.1.2 have been applied are similar to the effects discussed in the previous paragraph, the same techniques are applicable and the same observation about the number of clauses and variables when comparing the formula with and without symmetry breaking can be made. However, there is a minute difference between the amount of variables and clauses when comparing the formulas with and without improvements. The formulas with the improvements have less variables and clauses than their counterparts without the improvements. The difference in performance between the encodings can be seen in Figure 5b. The same observation can be made on the effect on performance of processing for instances 3 and/or 5 as in the prior paragraph. Additionally, the processed encodings perform at an equal or better level than

the unprocessed encodings. When comparing Figure 5a and Figure 5b it can be noted that the processed variant with symmetry breaking with improvements performs the best.

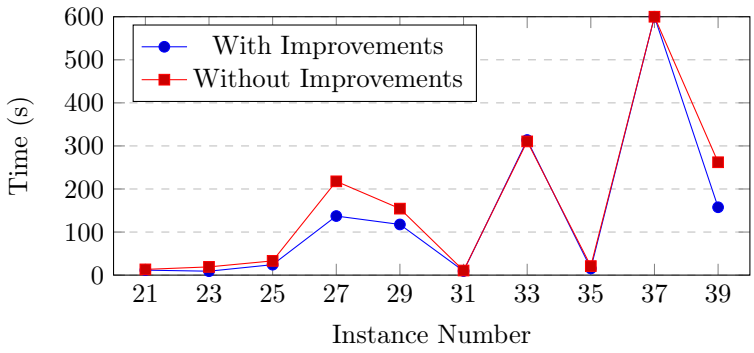
Furthermore, when comparing the results for the processed binary encoding with and without improvements there seems to be a difference in performance for certain instances. Based on Figures 5a, 5b and 5c on instances 13, 27, 29 and 39 the encoding with improvements outperforms the encoding without improvements. This indicates that the performance gain facilitated by the domain-specific knowledge is dependent on the instance but can potentially provide faster solving times. The conclusions about the comparative different are similar to the conclusions based on the transitive encoding.




(a) MaxHS Processed VS Unprocessed Binary Encoding **without** improvements, 600 second timeout




(b) MaxHS Processed VS Unprocessed Binary Encoding **with** improvements, 600 second timeout




(c) MaxHS With Improvements vs Without Improvements Processed Binary Encoding **with** Symmetry Breaking, 600 second timeout

Figure 5: *Binary Encoding Comparison between Processed and Unprocessed*

## 5.3 Evaluation of MaxSAT Solvers

The main criterion for evaluating a solver is based on the solving time of the formulas produced by the encodings (potentially preprocessed). To this end, the results of the MaxSAT Evaluation of 2020 are used as a point of reference [25]. Since the problem of *unweighted* cluster editing is being addressed, the assessment of the unweighted complete track is of

most importance. The top three solvers in this category are MaxHS [11] [12] [13] [14] [15], EvalMaxSAT [16] and UWrMaxSat [17].

As can be seen in Figure 6, MaxHS performs the best in terms of solving time hence this particular solver has been chosen for the evaluation. Due to the structure of the pipeline, the solver can be substituted with ease allowing for different solvers depending on the problem instance. Please note that, while not part of this work, it is possible that due to kernelization the unweighted graph is transformed into a weighted instance which could potentially cause other solvers to outperform the currently chosen solver.

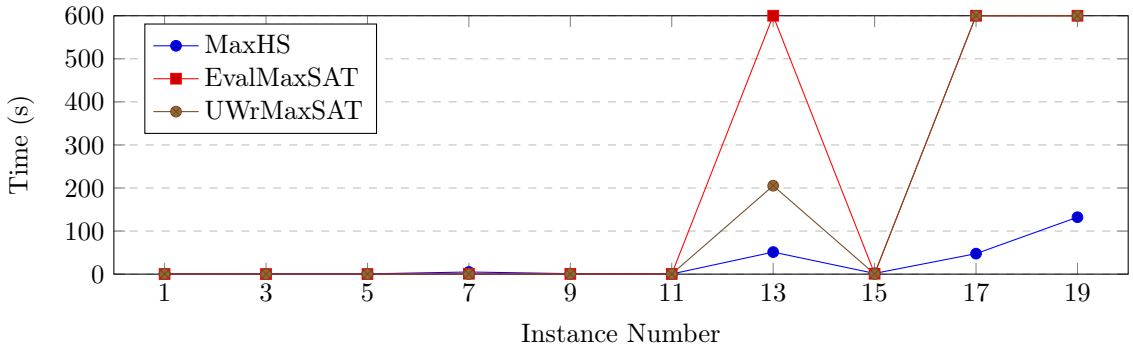



Figure 6: *Comparison solvers, Preprocessed Transitive Encoding with Improvements, 600 second timeout*

## 5.4 Comparison to Another Approach

In this subsection the performance of the MaxSAT-based approach for cluster editing is described. The results of the experimentation are shown and observations about these results are made. The transitive encoding with preprocessing and improvements solved by MaxHS is used as this pipeline has been shown to provide the best performance in prior sections.

### 5.4.1 Results

First, observations are made about the results of the experimentation (as seen in Figure 7) after which the underlying reasons for these results are discussed. Figure 7 shows the time it takes to solve a specific benchmark for the MaxSAT-based approach compared to the branching strategy implemented in [21].

- **Observation:** For certain instances the MaxSAT-based approach outperforms the branching strategy (e.g. instances 17, 19, 27 and 29)
  **Reason:** This is likely due to the fact that the graph structure cannot be exploited within the branching strategy. It is also possible that the heuristic which the branching strategy utilizes to select an edge to branch on leads to a sub-optimal path.
- **Observation:** For small instances, the branching techniques oftentimes achieve faster solving times (e.g. 1, 3, 5 and 7)
  **Reason:** The preprocessor (MaxPre) and solver (MaxHS) contain a certain amount of overhead associated with file reading/writing and the initialization of CPLEX. While

using another solver remedies this particular issue, it would be detrimental to the solving time on larger instances (as discussed in section 5.3).

- **Observation:** For certain instances, the solving time is significantly longer for the MaxSAT-based approach (e.g. instance 13)
  **Reason:** This is presumed to be due to the fact that MaxSAT can be seen as a "generalized" approach to cluster editing. The graph structure is not explicitly utilized which can lead the solver to explore more of the solution space than necessary. In addition to the aforementioned issue, the MaxSAT-based approach can (currently) not make use of reduction techniques/kernelization more than once while this is not necessarily the case when using a branching strategy. After merging a vertex/solving a conflict triple, the structure of the graph is changed (potentially into a weighted graph) which allows certain reduction rules/kernelization techniques to be used once more. Since the MaxSAT-based approach can not make use of this fact, certain instances will cause this approach to spend more time traversing a complex search space.

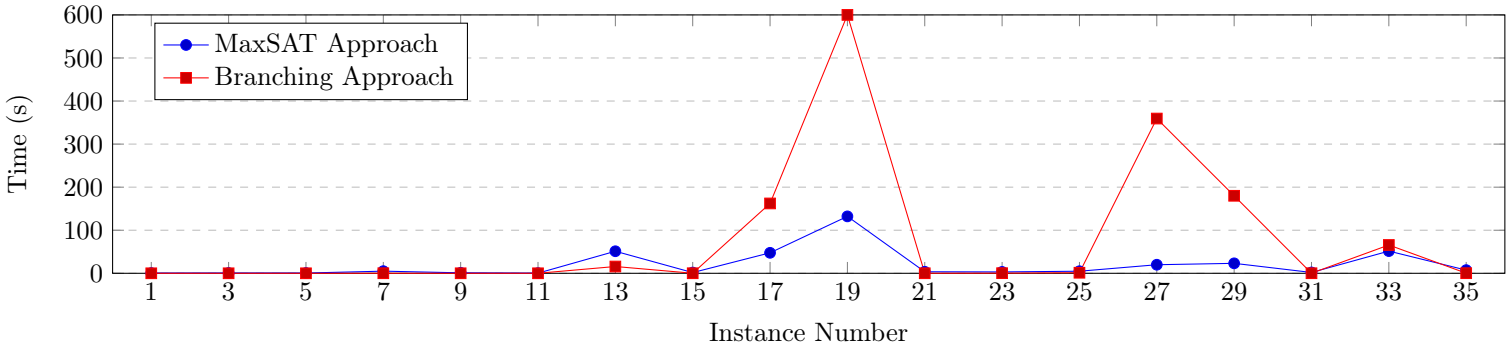



Figure 7: *Comparison Strategies, MaxSAT (Processed Transitive Encoding) vs Branching 600 second timeout*

### 5.5 Comparison to Existing Results

In this section the results gathered in this work will be compared to similar works (specifically [5]). Due to the limitations with regards to running time (1800s timeout for the PACE challenge), the results are similar but not entirely identical. For small/medium instances the transitive encoding outperforms the binary encoding which is similar to the results found in [5]. On larger instances it is likely that the binary encoding will outperform the transitive encoding. However, due to the fact that the solver oftentimes reaches the time limit for larger instances, this difference will not be visible. Furthermore, it is possible that on industrial instances the performance will differ.

## 6 Conclusions

In conclusion, it can be observed that it is possible to apply MaxSAT-based techniques to the cluster editing problem with certain caveats.

1. **Question:** *How can the cluster editing problem be modelled as a MaxSAT instance?*
   **Answer:** There are two known approaches to model cluster editing as a MaxSAT instance: Ensuring well-defined clusters through conflict triples (Transitive Encoding) and ensuring well-defined clusters by assigning each vertex to a cluster (Binary Encoding). The transitive Encoding was shown to be the most effective.
2. **Question:** *What preprocessing steps will increase the performance of cluster editing algorithms?*
   **Answer:** What preprocessing techniques are applicable is dependent on the encoding. The transitive encoding benefits from the addition of labels and similar techniques, while this does not reduce the amount of clauses nor variables applying these techniques has been shown to increase performance. Applying improvements based on cluster editing rules causes blocked clause elimination to become applicable. The binary encoding benefits from blocked clause elimination and bounded variable elimination. Unit propagation is applicable if symmetry breaking has been applied. However, in contrast to the transitive encoding, applying preprocessing to clauses created by the binary encoding can result in a worse performance dependent on the instance.
3. **Question:** *What MaxSAT-solvers can be used to efficiently solve the instances created by the encoding?*
   **Answer:** MaxHS was shown to be the most efficient on the given instances. For smaller instances, the other solvers oftentimes outperformed MaxHS due to the fact that MaxHS introduces overhead for (among other things) initializing CPLEX. If certain kernelization/reduction rules are applied which transform the instances into a weighted instance then other solvers are potentially applicable.
4. **Question:** *Under which circumstances do MaxSAT-based approaches outperform other Cluster Editing approaches?*
   **Answer:** MaxSAT-based techniques provide a "general" approach to solving cluster editing. These techniques do not exploit the structure of the graph in contrast to branching strategies. This can cause the search of the solution space by MaxSAT to be inefficient on certain instances. However, in the case that the graph structure does not provide accurate guidance, MaxSAT techniques can outperform the branching strategy. Conversely, a graph structure for which the MaxSAT-based techniques are likely to be outperformed by the branching strategy is when a graph has a large number of vertices but the solution contains a small number of edits. In this case, to ensure well-defined clusterings, the MaxSAT approach will generate a large number of triplets/pairs introducing overhead and decreasing performance.

# 7 Future Work

In future work, additional rules can be supplemented to further simplify the formula provided to the solver. Kernelization techniques can serve as a guide in this process. The MaxSAT-based approach suffers from a certain drawback as specified in section 5.4.1, in branching strategies in which vertices are merged and connected in subsequent iterations it is possible to make use of these techniques more than once while this is not necessarily the case for MaxSAT-based approaches. Integrating this concept into a MaxSAT-based approach could potentially significantly improve performance. Furthermore, in future works, new preprocessing and/or solving techniques might enhance the capabilities of a MaxSAT-based approach which would alter the applicability of this approach.

# 8 Responsible Research

In this section we will discuss the ethical aspects of the research and the reproducibility of the experiments. First of all, we would like to note that there are no surface level ethical implications of the research. This is partially due to the fact that our research does not harm others nor does it use user data in a malevolent fashion. User data is not collected nor required further decreasing the potential ethical implications of the research. Furthermore, while certain algorithms rely on datasets (e.g for training) which potentially introduce bias to the algorithm, this is not the case for this work. Due to these reasons, we will focus mainly on the reproducibility of the experimentation.

To ensure that the experiments are reproducible, the setup has been discussed in section 5.1. Furthermore, we have appended the different settings used for the solver/preprocessor to further ensure that the research can be reproduced. When using the same specifications and theory as described previously, similar results should be gathered. This allows others to verify the results in an accurate manner, which provides more credibility to the results. Additionally, the benchmarks are publicly available allowing others to attempt applying the techniques as specified in this work.

# Appendix A Derivation of Clustering |V| or Less

As specified in section 3.2.3, the binary encoding can assign a vertex to more clauses than necessary resulting in symmetries. This issue can be solved by introducing the clauses specified in section 8.5 of [5]. In this appendix, a derivation of the clauses in CNF is provided:

- **Base Case:**
  $DefB(i,1) = B_i^1 \leftrightarrow (\neg b_i^1 \wedge (K^1 = 1))$

  If $K^1 = 1$:

  $B_i^1 \leftrightarrow \neg b_i^1$ which leads to the following two cases:

  1. $B_i^1 \rightarrow \neg b_i^1$
     $\neg B_i^1 \vee \neg b_i^1$
  2. $\neg b_i^1 \rightarrow B_i^1$
     $b_i^1 \vee B_i^1$

  Else:

  $\neg B_i^1$

- **Recursive Case:**
  $DefB(i,j) = B_i^j \leftrightarrow ((\neg b_i^j \wedge (K^j = 1)) \vee ((b_i^j \leftrightarrow K^j) \wedge B_i^{j-1}))$

  If $K^j = 1$:

  $B_i^j \leftrightarrow (\neg b_i^j \vee (b_i^j \wedge B_i^{j-1}))$ which leads to the following two cases:

  1. $B_i^j \vee \neg(\neg b_i^j \vee (b_i^j \wedge B_i^{j-1}))$
     $B_i^j \vee \neg(\neg b_i^j \vee B_i^{j-1})$
     $B_i^j \vee (b_i^j \wedge \neg B_i^{j-1})$
     $(B_i^j \vee b_i^j) \wedge (B_i^j \vee \neg B_i^{j-1})$
  2. $\neg B_i^j \vee (\neg b_i^j \vee (b_i^j \wedge B_i^{j-1}))$
     $\neg B_i^j \vee \neg b_i^j \vee B_i^{j-1}$

  Else:

  $B_i^j \leftrightarrow (\neg b_i^j \wedge B_i^{j-1})$ which leads to the following two cases:

1. $\neg B_i^j \vee (\neg b_i^j \wedge B_i^{j-1})$
   $(\neg B_i^j \vee \neg b_i^j) \wedge (\neg B_i^j \vee B_i^{j-1})$

2. $B_i^j \vee \neg(\neg b_i^j \wedge B_i^{j-1})$
   $B_i^j \vee b_i^j \vee \neg B_i^{j-1}$

# References


[1] *Pace 2021 - cluster editing*, 2020. [Online]. Available: `https://pacechallenge.org/2021/cluster-editing/`.

[2] T. Wittkop, J. Baumbach, F. Lobo, and S. Rahmann, "Large scale clustering of protein sequences with force -a layout based heuristic for weighted cluster editing", *BMC bioinformatics*, vol. 8, p. 396, Feb. 2007. DOI: `10.1186/1471-2105-8-396`.

[3] R. Sharan, A. Maron-Katz, and R. Shamir, "Click and expander: A system for clustering and visualizing gene expression data", *Bioinformatics (Oxford, England)*, vol. 19, pp. 1787–99, Oct. 2003. DOI: `10.1093/bioinformatics/btg232`.

[4] J. Berg and M. Järvisalo, "Optimal correlation clustering via maxsat", in *2013 IEEE 13th International Conference on Data Mining Workshops*, 2013, pp. 750–757. DOI: `10.1109/ICDMW.2013.99`.

[5] J. Berg and M. Järvisalo, "Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability", *Artificial Intelligence*, vol. 244, pp. 110–142, 2017, Combining Constraint Solving with Mining and Learning, ISSN: 0004-3702. DOI: `https://doi.org/10.1016/j.artint.2015.07.001`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0004370215001022`.

[6] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo, "Maxpre: An extended maxsat preprocessor", in *Theory and Applications of Satisfiability Testing – SAT 2017*, S. Gaspers and T. Walsh, Eds., Cham: Springer International Publishing, 2017, pp. 449–456, ISBN: 978-3-319-66263-3.

[7] A. Belov, A. Morgado, and J. Marques-Silva, "Sat-based preprocessing for maxsat", in *Logic for Programming, Artificial Intelligence, and Reasoning*, K. McMillan, A. Middeldorp, and A. Voronkov, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 96–111, ISBN: 978-3-642-45221-5.

[8] N. Eén and A. Biere, "Effective preprocessing in sat through variable and clause elimination", in *Theory and Applications of Satisfiability Testing*, F. Bacchus and T. Walsh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 61–75, ISBN: 978-3-540-31679-4.

[9] M. Leivo, J. Berg, and M. Järvisalo, "Preprocessing in incomplete maxsat solving", in *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarın, and J. Lang, Eds., ser. Frontiers in Artificial Intelligence and Applications, vol. 325, IOS Press, 2020, pp. 347–354. DOI: `10.3233/FAIA200112`. [Online]. Available: `https://doi.org/10.3233/FAIA200112`.

[10] L. Zhang, "On subsumption removal and on-the-fly cnf simplification", in *Theory and Applications of Satisfiability Testing*, F. Bacchus and T. Walsh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 482–489, ISBN: 978-3-540-31679-4.

[11] J. Davies and F. Bacchus, "Exploiting the power of mip solvers in maxsat", in *Theory and Applications of Satisfiability Testing – SAT 2013*, M. Järvisalo and A. Van Gelder, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 166–181, ISBN: 978-3-642-39071-5.

[12] ——, "Postponing optimization to speed up maxsat solving", in *Principles and Practice of Constraint Programming*, C. Schulte, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 247–262, ISBN: 978-3-642-40627-0.

[13] ——, "Solving maxsat by solving a sequence of simpler sat instances", in *Principles and Practice of Constraint Programming – CP 2011*, J. Lee, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 225–239, ISBN: 978-3-642-23786-7.

[14] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko, "Reduced cost fixing in maxsat", in *Principles and Practice of Constraint Programming*, J. C. Beck, Ed., Cham: Springer International Publishing, 2017, pp. 641–651, ISBN: 978-3-319-66158-2.

[15] R. Hickey and F. Bacchus, "Speeding up assumption-based sat", in *Theory and Applications of Satisfiability Testing – SAT 2019*, M. Janota and I. Lynce, Eds., Cham: Springer International Publishing, 2019, pp. 164–182, ISBN: 978-3-030-24258-9.

[16] F. Avellaneda, "A short description of the solver evalmaxsat", Jul. 2020.

[17] M. Piotrów, "Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems", in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2020, pp. 132–136. DOI: 10.1109/ICTAI50040.2020.00031.

[18] S. Böcker, "A golden ratio parameterized algorithm for cluster editing", *Journal of Discrete Algorithms*, vol. 16, pp. 79–89, 2012, Selected papers from the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011), ISSN: 1570-8667. DOI: https://doi.org/10.1016/j.jda.2012.04.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570866712000597.

[19] M. Grötschel and Y. Wakabayashi, "A cutting plane algorithm for a clustering problem", *Math. Program.*, vol. 45, no. 1–3, pp. 59–96, Aug. 1989, ISSN: 0025-5610.

[20] S. Böcker, S. Briesemeister, and G. W. Klau, "Exact algorithms for cluster editing: Evaluation and experiments", in *Experimental Algorithms*, C. C. McGeoch, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 289–302, ISBN: 978-3-540-68552-4.

[21] L. Holten, *Fast vertex merging for cluster editing.*

[22] M. Järvisalo, A. Biere, and M. Heule, "Blocked clause elimination", in *Tools and Algorithms for the Construction and Analysis of Systems*, J. Esparza and R. Majumdar, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 129–144, ISBN: 978-3-642-12002-2.

[23] C. Komusiewicz and J. Uhlmann, "Cluster editing with locally bounded modifications", *Discrete Applied Mathematics*, vol. 160, no. 15, pp. 2259–2270, 2012, ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2012.05.019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X12002259.

[24] R. van Bevern, V. Froese, and C. Komusiewicz, "Parameterizing edge modification problems above lower bounds", *Theory of Computing Systems*, vol. 62, no. 3, pp. 739–770, Jan. 2017, ISSN: 1433-0490. DOI: 10.1007/s00224-016-9746-5. [Online]. Available: http://dx.doi.org/10.1007/s00224-016-9746-5.

[25] *Maxsat evaluation 2020*. [Online]. Available: https://maxsat-evaluations.github.io/2020/index.html.