Using graph addition-contraction to compute tighter bounds in hybrid MaxSAT/CP solving for Correlation Clustering

Georgescu, Calin   Schmahl, Katja   Simonov, Alex   Zeilstra, Mika

August 13, 2022

Abstract

Clustering is an unsupervised machine learning technique that groups data points based on similarity, with many distinct formulations and specialized solving techniques. This work focuses on a MaxSAT encoding of the correlation clustering problem and investigates the potential of a specializing a hybrid CP/MaxSAT solver with a novel propagator for this problem. This propagator tracks the state of the underlying graph by contracting and separating nodes based on decisions made, and can compute lower bounds by identifying substructures in the current problem state. The empirical study conducted on both real world and synthetic problems suggests that our implementation performs worse than the baseline for execution time, using both core-guided and linear search. The encoding used with the propagator is much smaller than the baseline, and as such could be beneficial with linear search in memory-limited applications. The most promising avenues for improvement are tighter bounding and a more efficient implementation of the propagation techniques.

1 Introduction

Clustering is one of the most thoroughly studied and widely spread paradigms in the field of data analysis [16, 15]. Although formulations of clustering problems are numerous and nuanced, the core of the problem is splitting a set of data points into different categories that reveal the underlying properties of the data. In general, clustering belongs to the class of unsupervised algorithms, and the partitioning of the data points is performed with regard to a similarity measure.

This work focuses on correlation clustering (CC), a straightforward but versatile tool used in data analysis applications. CC models the data as a graph, where nodes represent data points and edges between them are labeled either positive or negative. This indicates whether the two connected vertices are considered either similar or different [2]. The goal of the optimization problem is to produce a partition of the graph that "respects" as many edge annotations as possible. More specifically, the clustering should minimize the sum of the number of negative edges between nodes in the same cluster and the number of positive edges between nodes in separate clusters. Like many problems belonging to the clustering paradigm, CC belongs to the class of NP-Hard problems.

CC models lend themselves naturally to several real-world problems, and thus our work is applicable to a wide range of problems. Two examples of CC problems are the work by Nijssen et al. [12] and Zimek [17]. Nijssen et al. [12] use CC as a model for mining data sets. Slightly different formulations can model pattern mining as a CC problem and thus a CP optimization problem [7]. Zimek [17] hypotheses that gene expression analysis, metabolic screening, and customer recommendation systems can be faithfully modeled through CC formulations.

Due to the problem's computationally demanding nature, many approaches to correlation clustering are based on heuristic and approximation algorithms. Such approaches include fixing the number of clusters [6] and reducing the problem to the maximal multi-cut problem, then leveraging specialized approximation techniques [5]. However, these approaches do not provide guarantees of optimality, which numerous real-world applications require. This work focuses on exact solving through encoding the problem in the MaxSAT setting and leveraging advanced boolean satisfiability techniques. The aim is to improve the performance of existing exact solvers, which in general scale poorly [9].

The choice of MaxSAT as a means of solving correlation clustering problems is motivated by three key advantages. First, the declarative nature of MaxSAT offers versatility in terms of both solver choices and problem extensions. The former is an advantage for practical performance: since boolean satisfiability is one of the most studied problems in theoretical computer science, increasingly advanced solvers are being developed that improve efficiency without any overhead on the user's end. Approaching the problem in a declarative manner means additional extensions of constrained clustering can also be specified, such as the commonly occurring must- and cannot-link constraints [14]. Last, SAT-based approaches allow for instances to be solved to optimality. This may be an extremely valuable property for difficult-to-obtain data sets or problems relating to sensitive real-life problems. MaxSAT can be approached through linear search, which generates incrementally better solutions until optimality is reached and proven. While potentially slower than other approaches, these techniques enable our formulation to retain the advantage of approximation algorithms, the generation of approximate solutions, while providing optimality guarantees if afforded enough time.

We chose to extend this MaxSAT approach to a hybrid MaxSAT/CP solving technique, since this allows us to incorporate problem-specific knowledge to increase solving speed. This way, there is potential for larger speed-ups than when the solver is restricted to Black Box optimization improvements. In particular, our approach is centered around implementing a novel graph propagator that uses Zykov's subtraction-addition recurrence and keeps track of the intermediate graph state [18]. This allows dynamic bounding techniques to prune the search space.

Our empirical analysis suggest that our implementation of the propagator is not efficient enough to outperform the baseline encoding in large real-world problems. However, the results also indicate that the bounds may provide an improvement in the time taken to prove the optimality of solutions in linear search settings, for synthetically generated problems. The convergence time when considering the number of decisions alone is comparable between our work and the baseline.

The remainder of this paper is organized as follows: section 2 discusses related work, and section 3 introduces the necessary background notions. Our contributions and their motivations are described in section 4, section 5 details the empirical analysis carried out to evaluate our results. The results of the empirical study are highlighted in section 6. A summary and conclusion are given in section 7. Lastly, possible future extensions are detailed in section 8.

2 Related work

In this section, we provide an overview of the work that has been done on correlation clustering and motivate the novelty of our approach.

Exact solving for correlation clustering. Bansal et al. [2] laid the foundations for CC by formalizing the problem, showing that it is NP-complete, and developing an efficient approximation algorithm. Multiple exact Integer Linear Programming (ILP) and Quadratic Integer Programming (QIP) models of the CC problem were formulated for specific use cases [13, 1, 4].

Berg and Järvisalo [3] proposed a novel MaxSAT-based formulation along with three encoding strategies. They empirically show that all three encoding types outperform both the ILP and QIP models in terms of runtime and memory usage scalability. Their novel approach scales better but still times out between 14 and 54 percent of the tested instances, depending on the solver. They hypothesize that a more sophisticated pruning method could significantly improve the results. Miyauchi et al. [10] achieved better scalability with their ILP model by perturbing the zero-weight edges in the graph and reducing the number of constraints of the model. They show that their formulation significantly outperforms the MaxSAT formulation of Berg and Järvisalo in several benchmarks. However, they still failed to obtain exact results on several benchmark problems.

Hybrid MaxSAT/CP solving. Marchal's [9] work introduced a new MaxSAT algorithm for CC, and a novel encoding based on the work of Hebrard and Karsirelos [8] using lazy clause representation. This encoding has a smaller size than the encoding proposed by Berg and

Järvisalo [3] and is competitive in terms of runtime performance. He also proposed a problem-specific propagator for a hybrid SAT/CP approach. His bounding was ineffective when using core-guided search. However, he suggests extending the propagator might significantly improve the algorithm's performance [9].

To potentially find a more scalable solution in both encoding size and runtime, we take inspiration from the work of Hebrard and Karsirelos [8]. They propose a specialized SAT/CP solving method for the problem of graph coloring, which shares many symmetry structure challenges with CC. They introduce key problem-specific novelties, including a transitivity aware propagator and two heuristic bound estimators, inspired by Mycielskian graphs [11]. They achieved significantly better solving efficiency than the current state-of-the-art solutions [8]. We have adapted the principle of a transitivity aware propagator that continuously updates the problem state to CC, with the aim of improving runtime performance.

3 Background

This section provides an overview of basic concepts and notation required for understanding our approach.

3.1 Correlation clustering

First introduced by Bansal et al. [2], correlation clustering is a similarity-based clustering problem that seeks to cluster together nodes that are similar. Formally, the problem consists of a set of vertices $V = \{v_i\}\ \forall i \in \{1, .., \mathbf{N}\}$ and two sets of edges $E^+, E^- \subseteq (V \times V)$ with the property $E^+ \cap E^- = \emptyset$. Two nodes v_i and v_j are said to be *similar* if $(v_i, v_j) \in E^+$ and dissimilar if $(v_i, v_j) \in E^-$. We also assume that edge similarity is symmetrical, i.e., that $(v_i, v_j) \in E^s \iff (v_j, v_i) \in E^s, s \in \{+, -\}$. The problem can be abstracted into a graph $G = (V, E^+, E^-)$, which represents the relations between all data points in the problem. The more general formulation of the problem can be extended from the basic version by introducing a weight function $W : E^+ \cup E^- \to \mathbb{R}$ that assigns weights to all edges in the graph, which can be interpreted as a more granular measure of similarity. For simplicity purposes, the remainder of this paper addresses the non-weighted variation of CC ($W(v_i, v_j) \in \{-1, 1\}\ \forall v_i, v_j \in E^+ \cup E^-$), although all concepts can be trivially extended to the weighted version. A clustering is a function $cl : V \to \mathbb{N}$ that assigns each point in the graph to a cluster represented by a natural number. Two points v_i and v_j are said to be co-clustered if $cl(v_i) = cl(v_k)$.

Intuitively, the optimization goal is to minimize the cost of the clustering, which corresponds to a sum of positive and negative mistakes. A positive mistake is an assignment of two dissimilar nodes to the same cluster, and a negative mistake is an assignment of two similar nodes to different clusters. More formally, the optimization function is given in the equation below.

$$\sum_{\substack{(v_i, v_j) \in E^+ \\ cl(v_i) \neq cl(v_j)}} W(v_i, v_j) + \sum_{\substack{(v_i, v_j) \in E^- \\ cl(v_i) = cl(v_j)}} W(v_i, v_j)$$

3.2 SAT and MaxSAT

The satisfiability problem, also called SAT, is one of the core problems in computer science. Its purpose is to assign variables boolean values according to constraints. More formally, let $v_1, v_2, ..., v_n$ be a list of variables, and let there be m constraints of the form $\bigvee_{i=0}^{k} v_i$. The SAT problem is to determine whether there exists an assignment of values to variables such that all the constraints are satisfied. Constraints in SAT are called clauses, and the conjunction of clauses is called a proposition.

MaxSAT is an extension of SAT problem where the goal is to maximize the number of satisfied clauses. There are different variations of MaxSAT. Weighted MaxSAT gives every clause a weight, and the objective is to minimize the sum of weights of falsified clauses. The variation of weighted MaxSAT is weighted partial MaxSAT, which introduces the notion of hard and soft clauses. In such a variation, the objective is to satisfy all the hard clauses and minimize the number of unsatisfied clauses.

More formally, let (C, w) be a pair of a clause C and the cost w imposed for falsifying the clause. If the clause C is hard, w is equal to infinity. Let $\varphi = \{(C_1, w_1), (C_2, w_2), ..., (C_m, w_m), (C_{m+1}, \infty), ..., (C_n, \infty)\}$ be a multiset of weighted clauses and I be the truth assignment. Then weighted partial MaxSAT problem is the problem of minimizing the cost of assignment I on φ. If the cost is infinity, then one of the hard clauses is not satisfied, so the multiset becomes unsatisfiable.

Two main approaches exist for solving MaxSAT. Core-guided search is a lower bound approach which uses the information of unsatisfiable sets of clauses (cores) impose a bound on the problem. Linear search is an upper bound approach that uses a SAT-solver iteratively to find assignments that satisfy increasingly many soft clauses.

3.3 Transitive encoding

The transitive encoding, as introduced by Berg and Järvisalo [3], is a direct mapping from a correlation clustering problem instance to a MaxSAT problem. It consists of a variable for each possible edge between vertexes, soft clauses with a weight of either 1 or -1 for each edge, and hard clauses which enforce transitivity. More formally, for a correlation clustering instance $G = (V, E^+, E^-)$, the transitive encoding generates (boolean) variables $x_{ij} = x_{ji}\ \forall 1 \leq i \leq j \leq |V|$, the set of hard clauses $F_h = \{(\neg x_{ij} \vee \neg x_{jk} \vee x_{ik})\}\ \forall$ distinct i, j, k , and the set of soft clauses $F_s = \{x_{ij}\ \forall\ (i,j) \in E^+\} \cup \{\neg x_{ij}\ \forall (i,j) \in E^-\}$

The hard clauses are necessary to enforce transitivity. If two edges in a triangle of vertexes are set to true, all three must be in the same cluster, and thus we can deduce that the third edge must also be true. In a logical setting, these clauses enforce that $(x_{ij} \wedge x_{jk}) \rightarrow x_{ik}$. Without these constraints, the cluster assignments might not be consistent.

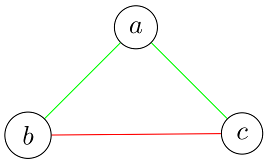
This encoding was chosen as the underlying representation for our solver over other possibly more efficient encoding types because it is very straightforward to decode what each variable means and has the closest resemblance to our proposed parallel graph structure.

3.4 Using substructures as lower bounds

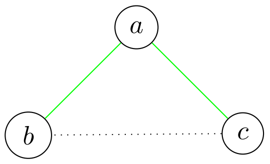
One of the potentially effective ways to boost solving efficiency is to prune the MaxSAT search space. This can be done by using lower bounds on the total cost of the problem based on the internal representation of the graph at each step.

We conceptually divide the costs present in CC graphs into two types: direct and future costs. Direct costs correspond to the number of (soft) constraints already violated. This type of cost is incremented whenever a decision or propagation is made. Future costs are a lower bound on the cost that must still be incurred. Our estimation method is centered around counting the number of specific substructures present in the graph in arbitrary states. To do this, we use 2 triangle-based structures.

Figure 1 introduces the triangle types used to count lower bounds. There are five possible assignments to clusters for each triangle. The crucial idea is that for any clustering of the nodes a, b and c, incurring a cost of at least 1 is inevitable. For example, Figure 1a shows a simple triangle with two positive edges and one negative edge. If nodes a and c are combined into one cluster and b is in a separate cluster, the cost would be 1 because nodes a and b are in different clusters. The same pattern applies to all the possible clusterings of the graph. Therefore, without any solver decisions, it can be inferred that the cost of the graph with such a triangle is at least 1. The same logic can be applied to a different triangle structure: Λ-triangle in Figure 1b. For more insight into these types of triangles, we refer the reader to the Master thesis of Marchal [9]. These triangles must be completely disjoint because otherwise the greedy computation might overestimate the cost, as exemplified in Figure 2.



(a) Triangle with a cost of at least one. Positive edges are green, negative edges are red.



(b) Λ-triangle. Dashed lines indicate that the node are separated.

Figure 1: Bounding triangle substructures.

3.5 Recurrence structure

To improve the efficiency of the algorithm, we prune the search space throughout the run. During execution, the solver keeps track of changes in the underlying graph. At any point, the solver can choose to either contract two nodes into the same cluster, or separate them. In this context, separation means that the two

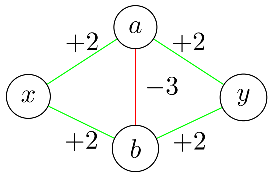
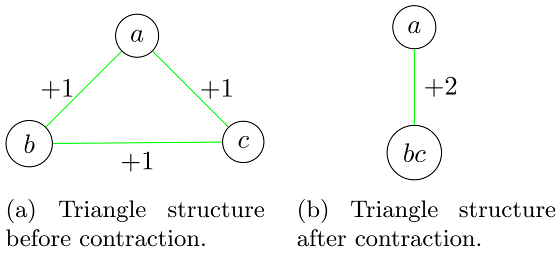


Figure 2: Two bounding triangles sharing an edge. If a, b, c and d are merged together only the cost of ac is incurred, resulting in cost 3. The greedy counting of two independent triangles would expect a cost of at least 4.

nodes cannot be part of the same cluster in any subsequent state on that search branch. These operations follow the recurrence relation of Zykov [18]. Figure 3 shows a small example of how two nodes of a graph are contracted when they are decided to be in the same cluster. Two nodes b and c are merged into one node bc, together with their edges, which have as weights the sum of weights of edges from both nodes to another node.



a
$+1$
$+1$
b
c
$+1$
(a) Triangle structure before contraction.
a
$+2$
bc
(b) Triangle structure after contraction.

Figure 3: Contraction of nodes a and b, since they both had a positive edge to c the weight of ab to c is now $+2$.

4 Graph Propagator

The propagator is separated into two parts, propagating transitivity and propagating bounds. Both types of propagation are based on an internal graph structure synchronized with the solver. Whenever the solver assigns a variable to true, two nodes are merged, and whenever the solver assigns a variable to false, two nodes are marked as separated. Each node in this graph structure initially represents only one vertex. However, using the Zykov recurrence, any node might represent an arbitrary number of vertices later in the solving process.

4.1 Propagating transitivity

When we mark two nodes as separated or merge them in the graph structure, we can also easily reason what other assignments must follow. Nodes in the graph structure have a list of vertices they represent, and thus which variables are co-clustered. If two nodes are merged, we need to set all the variables corresponding to the edges between two merged nodes to true, propagating that all vertices are in the same cluster to the solver. If we separate two nodes, a symmetric procedure is followed. More precisely, we need to propagate that all variables with one vertex in the first cluster and the second vertex in the second cluster are separated. Both types of transitive propagation can fail if the solver has already assigned one of the variables to the opposite value. In this case, a conflict is raised, where the conflicting edges are given as an explanation.

By enforcing the transitivity constraints using the propagator, we make the transitivity constraints in the MaxSAT problem encoding redundant. Therefore, a lazy encoding is used with only the soft constraints for each edge, reducing our worst case on the number of constraints in the problem definition to $\mathcal{O}(E)$ where E is the number of edges. This encoding can be considered lazy since the transitivity constraints are generated from the learned clauses by the solver only when necessary, as opposed to creating all constraints in advance.

4.2 Propagating Bounds

To more efficiently prune the search space, we compute a lower bound by counting triangle substructures in the graph. These bounds are then used to stop the search for the solution in the current variable assignments if the current state can not improve on the best solution found so far. Thus, when the lower bound of the current state of decisions is higher than the global upper bound from the best-found solution so far, a conflict can be derived.

This lower bound is calculated based on the triangles as discussed in subsection 3.4. There are three challenges in this propagation. Firstly, finding the best bound is itself a hard problem. Secondly, the graph keeps changing during the solving process, meaning that previously counted triangles might no longer exist, or a better bound might have become available. The last difficulty is providing an explanation to the solver for the conflict raised.

The first problem is tackled by using an approximate greedy approach. This algorithm finds bound in $\mathcal{O}(N^3)$ where N is the number of nodes. It works by first finding all cliques of size 3 in the internal graph. If this clique follows the structure of one of the bounding triangles, we add the minimum incurred cost to the lower bound and mark the edges to avoid overestimation. The approximate bound is the sum of all costs implied by the counted triangles.

The second problem is solved by exploiting the fact that the point when the bounds need to be updated is right after a change in the graph structure, which also happens after propagating transitivity. Thus, checking the bounds immediately after transitivity is propagated ensures the bounds are checked as often as possible without recalculating the bounds on the same graph.

Providing a good explanation of why the decisions can not result in an optimal solution can help the solver learn from the conflict. In our propagator, this explanation consists of all decisions made before this point. This part of the search space does not need to be explored anymore. However, it does not provide more precise information as to which subset of decisions caused this higher bound to the solver.

5 Empirical Study

This section provides the details of the empirical study carried out to assess the performance of our implementation.

5.1 Experimental goal

The empirical study aims to determine the influence of the novel propagation techniques on the performance of the MaxSAT solver. To this end, several key aspects of the behavior of the different versions of the solver are compared. The main research question we aim to answer is:

> [**RQ**] Does a propagator with dynamic bounding using the addition-contraction recurrence of Zykov improve the performance of exact solving using hybrid SAT/CP of correlation clustering?

To limit the scope of this study, we split the main question into three sub-questions:

1. *How does the novel propagator differ when using a core-guided optimization approach instead of linear search?*
2. *How does the novel propagator influence the solution quality of the underlying MaxSAT solver?*
3. *How does the novel propagator influence the number of decisions made by the underlying MaxSAT solver?*

5.2 Benchmark Instances

We used two synthetically generated graph instances, which we refer to as `synth_N`, respective to their sizes. To generate these graphs, a clique of size N is generated. From that clique, edges are split between positive and negative edges according to some specified Bernoulli distribution parameter p_{split}. After the split, each edge is deleted from the graph according to a different Bernoulli parameter p_{delete} such that the (approximate) desired graph density is obtained. For these experiments p_{split} and p_{delete} are fixed to 0.5 and 0.8, respectively.

In addition, we used five real-life datasets, which we transformed to graph instances. These were all obtained from a database provided by NTU Singapore [1]. Since some instances were too large for the limited memory available within this research, we reduced the size by only selecting a subset of data points. Table 1 lists some relevant properties for each of the benchmark sets.

Name	Nodes	Classes
`glass`	214	6
`iris`	150	3
`vehicle`	400	4
`vowel`	250	11
`wine`	178	3

Table 1: The used real-life datasets. The number of classes gives insight in the expected result of the clustering.

5.3 Experimental Protocol

This subsection describes the evaluation strategies and quantifiers used to assess the performance of the propagator implementation.

[1]`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

5.3.1 Metrics

Objective value. The objective value of the final and intermediate solutions produced by the solver is an essential descriptor of its quality, when constrained by time. This metric can only be used when at least one of the solvers in the comparison did not find and prove optimality.

Number of decisions. The goal of the bounding techniques is to prune the search space such that decisions that do not lead to optimality are avoided. Evaluating solution quality as a function of the number of decisions made by solver helps decouple the algorithmic performance of the solver from its implementation.

Number of conflicts from bounding. We use the number of conflicts our propagator raises as an indicator of the frequency with which the solver exploits the information from our technique. This reveals how effective the bounds are and helps to determine how efficient the propagator needs to be as to not bottleneck the solver.

5.3.2 Evaluation strategy

The metrics are obtained by sampling solver runs on the graph instances. To generate results with a higher statistical significance, each instance is independently solved 5 times using different seeds and the results are aggregated. The timeout was set to 60 minutes for the real-world problems and to 5 minutes for the synthetic instances.

The compared solver configurations are: (1) the transitive encoding of Berg Järvisalo [3] with the standard version of the solver, (2) the lazy transitive encoding with the version of the solver that integrates our custom propagator and no bounding techniques, and (3) the lazy transitive encoding with the solver including the custom propagator and the triangle bounds. We refer to these versions as `trans`, `prop`, and `prop + bounds`, respectively. All experiments were executed once with a full linear search based approach and once for a full core-guided search approach.

All instances of the solver have been implemented in C++ and compiled using `g++ (GCC) 12.1.0`. The solver used is Pumpkin, an unreleased SAT/MaxSAT solver created by Emir Demirović. All experiments have been carried out on a Manjaro 21.2.6 machine running on an AMD Ryzen 7 5800H CPU at 3.20GHz, with 16GB of RAM.

Instance	trans	prop	prop_bounds
`glass`	t/o	t/o	t/o
`iris`	t/o	t/o	t/o
`wine`	2.07	t/o	t/o
`vehicle`	8.32	t/o	t/o
`vowel`	22.57	t/o	t/o
`synth_20`	0.90	5.56	2.03
`synth_25`	t/o	t/o	163.62

Table 2: Time in CPU seconds required by linear search to prove optimality. t/o indicates the configuration timed out.

Instance	trans	prop	prop_bounds
`glass`	16.42	t/o	t/o
`iris`	1.64	t/o	t/o
`wine`	2.05	t/o	t/o
`vehicle`	7.71	t/o	t/o
`vowel`	4.63	t/o	t/o
`synth_20`	0.01	0.40	n/a
`synth_25`	0.11	2.76	n/a

Table 3: Time in CPU seconds required by core-guided search to prove optimality. t/o indicates the configuration timed out and n/a means the configuration crashed.

6 Results

This section contains the analysis of the results of the empirical study.

6.1 Core guided search

To capture the difference between the two solving paradigms, we performed all experiments using core guided and linear search, respectively. The results averaged over 5 independent runs are given in Table 2 and Table 3.

The results indicate that the `trans` encoding performs significantly better when using core-guided search. When compared to linear search, the core-guided configuration is able to prove optimality in three more instances and the optimal results are always found more quickly. This is especially noticeable on the `vowel` instance, where core-guided is on average 80% faster.

`prop` displays a similar pattern to `trans` for the synthetic instances. Using core-guided search, it takes an order of magnitude less time for it to prove optimality for `synth_20` than in the linear search setting. In addition, this configuration is only able to solve `synth_25` when using core-guided search. Due to implementation errors, core-guided search could not be tested with the `prop + bounds` version on the synthetic problems. Because both propagator configurations time out on real world problems, no decisive comparison can be made between the two.

The comparisons indicate that core-guided search is superior to linear search when the goal is to find the optimal solution to a CC problem. However our bounding technique does not provide any improvement, no conflicts are being raised from it. However, if the goal is to find as good a solution as possible within a fixed time budget that does not allow optimality to be proven, linear search is the superior option because it can generate valid intermediate solutions. In this case, our bounding technique does raise conflicts, as can be seen in Table 4.

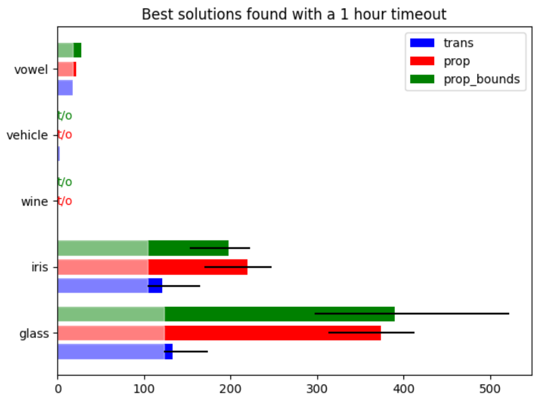


Figure 4: The mean of the best solutions found for the real-life datasets. The lighter area shows the optimal solution. t/o indicates that the solver timed out before any solution was found. The black lines show the range from best to worst solution found.

6.2 Final solution quality

To capture the influence of the novel propagator, we consider the solution quality obtained within the time budget when using linear search. The results for this can be found in Table 2. For the synthetic datasets, the optimal solution is found using linear search with all techniques, but optimality is only proven consistently for `synth_20`.

Instance	Mean no. of conflicts
`glass`	4407
`iris`	46023
`vowel`	725
`vehicle`	0
`wine`	0
`synth_20`	10397
`synth_25`	144615

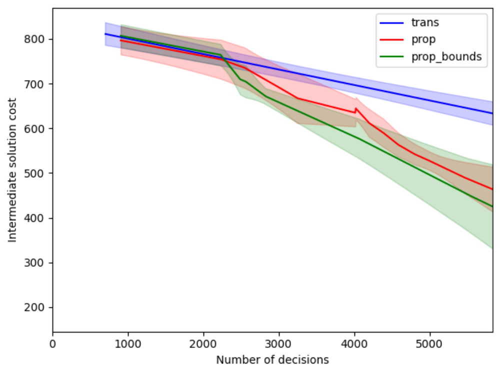
Table 4: Mean number of conflicts based on the lower bound found by our bounding technique, rounded to the closest integer for linear search. For core-guided search, there were never any conflicts found.

This instance is solved significantly faster by the transitive encoding, with `trans` being on average twice as fast as `prop + bounds` and six times faster than `prop`. These results indicate that linear search may benefit from the bounds provided by the novel propagator on small problems, but the implementation is too slow for it to keep up with the performance of `trans` on real world instances. For `synthetic_25`, `trans` and `prop + bounds` find the optimal solution in all runs, while `prop` only finds it in 2 of 5 runs. However, optimality is only proven when using the propagator with bounds. This suggests that the bounding technique might provide a speed-up for proving optimality, even though it does not find the solution faster. This insight is reinforced by the number of bound conflicts that were raised, presented in Table 4. For the `synth_25` instance, the bounds prune the search space significantly more often in comparison to the others.

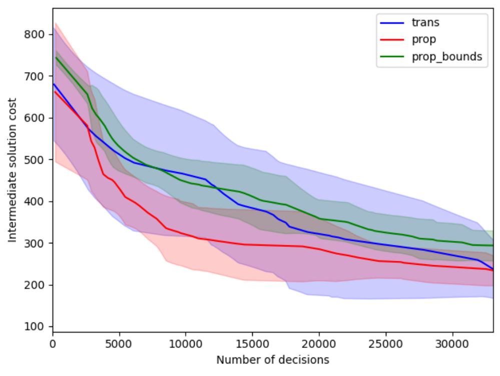
The results of the final solution cost for the real world datasets can be found in Figure 4. The transitive baseline always finds a better solution quality when given a time-based budget. For the `vehicle` and `wine` instances, it is the only configuration to find any solution. There is no significant difference between the quality of the solution found with or without bounding.

6.3 Convergence of solution

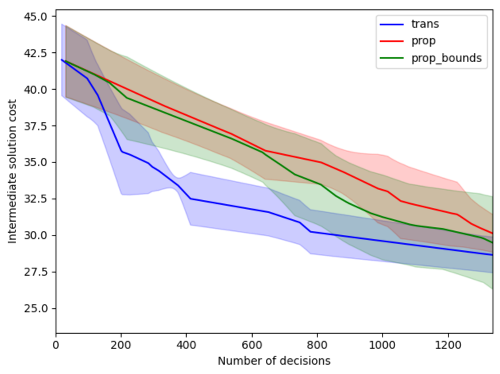
To be able to analyze how the propagator and bounds affect the number of decisions required to get to a solution of a certain quality, we measured the convergence of the intermediate solutions over the number of decisions. The results for this can be seen in Figure 5. For the



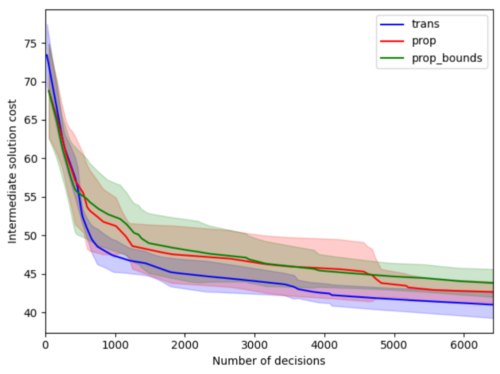
(a) Glass



(b) Iris



(c) Synthetic dataset of size 20.



(d) Synthetic dataset of size 25.

Figure 5: The convergence of solution quality over number of decisions for each of the tested configurations. The line shows the mean value of the 5 executions, with a region colored with the height of a standard deviation in both directions, until the minimal number of decisions reached over all executions.

larger datasets, the propagator was too slow to find any useful solutions within the time budget. For the `glass` dataset, the propagator with bounds finds lower cost solutions with fewer decisions. For the `iris` dataset, the propagator without bounds seems to perform better in the first 10,000 decisions, but the difference between solvers shrinks over time. For `synth_20`, a similar pattern emerges for `trans`. For `synth_25`, `trans` performs best within the number of decisions, and the other two solvers perform similarly. Overall, there is no clear preference for any of the 3 techniques when using a time-out based on the number of decisions, based on the tested datasets.

7 Conclusion

We have introduced a novel transitivity-based propagator for solving the correlation clustering problem exactly in a hybrid MaxSAT/CP framework, which uses dynamic triangle bounds to prune the search space for the underlying solver. We implemented our propagator using an intermediate representation of the underlying graph within Pumpkin, a SAT/MaxSAT solver created by Emir Demirović.

To assess the performance of our propagator, we conducted an empirical study on a dataset consisting of five real-world datasets and two small stochastically generated problem instances. We compared two versions of our implementation against a different encoding without a custom propagator. The results indicate that the combination of core-guided search and the baseline transitive encoding performs best. The propagator does not provide any added benefit, but merely adds unnecessary computational overhead when using core-guided search.

When using linear search, the propagator suffers from a large overhead, which negatively impacts solution quality for most instances in comparison to the baseline. The only clear advantage of the propagator configurations is the much smaller size of its encoding. Further, our research indicates that the bounding does not significantly improve performance. For two of the instances, the newly introduced bounds are tight enough to prune the search space and improve efficiency. This might indicate that an improved version of the propagator could be beneficial in cases where memory is essential.

8 Future work

This section outlines some of the directions that future improvements and extensions of our work could focus on.

Core-guided search The empirical results suggest a strong reduction when using a core-guided search strategy, and our propagator currently does not provide any benefit in this case. We were unable to find the underlying reason why the lower bound occurs in the time frame of this work. Research to deepen the understanding of the combination of core-guided search and the substructures might allow adapting the bounding in a way that improves core-guided search. This would be essential for this method to potentially improve upon the current state-of-the-art.

Tighter Bounds Our work extensively uses the triangle bounding technique, which only considers 3-cliques and the minimum clustering cost they necessitate. However, there are cases where more sophisticated bounds can be employed. We present one such example in Figure 2. Using the simple triangle bounding techniques, either of the `<x, a, b>` and `<y, a, b>` triangles would be found, and a bound of 2 would be computed for either of its positive edges. The remaining triangle would be discarded from the bound computation in this scenario. A tighter bound can be obtained when considering that the edge `(a, b)` is shared between the triangles: if `a` and `b` are clustered together, this can be seen as both triangles incurring half the cost of the `(a, b)` edge. Using this approach, a tighter bound of 3 would be computed, which might improve the solver's performance. However, there are also advantages to this approach. Since edges can be shared between an arbitrary number of triangles, the computation might be too expensive to be practically viable within a solver. Moreover, it is possible that such structures are not common enough in real-world applications to make a significant difference.

Propagating other encodings The transitive encoding was chosen for this project for the relative ease of its implementation compared to the alternatives. However, Berg Järvisalo [3] found through their empirical that the more sophisticated binary encoding performs better on real world problems. While implementing a propagator for this encoding would be far more complex due to the number of constraints involved and possible optimizations that they might be subjected to, this might prove more efficient than the transitive implementation.

Disconnected graphs Our graph representation is currently only used for finding better lower bounds. However, there is more potential in having this representation available, for example, by looking at disconnected graphs. Two independent problems can be solved in parallel when the graph is disconnected. This might result in more efficient solving.

Better bound explanations The clauses learned from our bounding techniques can be very long and contain many decisions that do not influence the incurred or future cost we computed. Generating explanations based on the incurred cost and the assignments of the found triangle structures would be more complicated and possibly more expensive. However, we believe it could also lead to better conflict-driven learning and reduce the number of decisions.

Implementation efficiency For this work, the focus was on looking into the potential of this method. The efficiency of implementation was not a priority, which influences the runtime results. For a more complete insight in this potential way to reduce the search space, this could be optimized in future work, to see the minimal overhead required for identifying the substructures. It would also be interesting to consider not recomputing the bounds after every decision, but incrementally keep track of changes to the bound.

References

[1] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008.

[2] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1):89–113, 2004.

[3] Jeremias Berg and Matti Järvisalo. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artificial Intelligence*, 244:110–142, 2017.

[4] Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Tao Jiang. On the approximation of correlation clustering and consensus clustering. *Journal of Computer and System Sciences*, 74(5):671–696, 2008.

[5] Erik D Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006.

[6] Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. *arXiv preprint cs/0504023*, 2005.

[7] Tias Guns, Siegfried Nijssen, and Luc De Raedt. k-pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*, 25(2):402–418, 2011.

[8] Emmanuel Hébrard and George Katsirelos. Constraint and satisfiability reasoning for graph coloring. *Journal of Artificial Intelligence Research*, 69:33–65, 2020.

[9] Maxim Marchal. Exact machine learning: Improving space and speed of maxsat solvers for correlation clustering. Master's thesis, TU Delft, 2021.

[10] Atsushi Miyauchi, Tomohiro Sonobe, and Noriyoshi Sukegawa. Exact clustering via integer programming and maximum satisfiability. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] Jan Mycielski. Sur le coloriage des graphes. In *Colloq. Math*, volume 3, page 9, 1955.

[12] Siegfried Nijssen, Tias Guns, and Luc De Raedt. Correlated itemset mining in roc space: a constraint programming approach. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 647–656, 2009.

[13] Jurgen Van Gael and Xiaojin Zhu. Correlation clustering for crosslingual link detection. In *IJCAI*, pages 1744–1749, 2007.

[14] Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. *AAAI/IAAI*, 1097:577–584, 2000.

[15] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[16] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[17] Arthur Zimek. Correlation clustering. *ACM SIGKDD Explorations Newsletter*, 11(1):53–54, 2009.

[18] Alexander Aleksandrovich Zykov. On some properties of linear complexes. *Matematicheskii sbornik*, 66(2):163–188, 1949.