The 3-Decomposition Conjecture: A SAT-Based Approach with Specialized Propagators

Tianwei Zhang
Algorithms and Complexity Group, TU Wien, Austria

Stefan Szeider
Algorithms and Complexity Group, TU Wien, Austria

Abstract

We investigate the 3-decomposition conjecture, which states that every connected cubic graph can be decomposed into a spanning tree, a collection of cycles, and a matching. Using a SAT-based approach enhanced with specialized propagators, we verify the conjecture for all relevant graphs up to 28 vertices. Our method extends the Satisfiability Modulo Symmetries (SMS) framework with specialized propagators that exploit theoretical properties of minimal counterexamples (counterexamples with the minimal number of vertices), enabling efficient pruning. We demonstrate that graphs containing certain substructures cannot be minimal counterexamples to the conjecture, allowing us to exclude these patterns during the search dynamically. Our experimental results quantify the impact of different propagator configurations and forbidden subgraph constraints on solving efficiency, showing significant performance improvements when leveraging these techniques. The approach scales effectively to graphs of 28 vertices. Our work illustrates how combining SAT solving with specialized constraint propagation techniques can successfully address challenging combinatorial problems in contemporary graph theory.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Mathematics of computing → Graph theory; Theory of computation → Logic and verification; Mathematics of computing → Combinatorics; Theory of computation → Automated reasoning; Mathematics of computing → Graph algorithms

Keywords and phrases SAT, Symmetry Breaking, Subgraphs, Propagators, Combinatorics

Digital Object Identifier 10.4230/LIPIcs.CP.2025.39

Supplementary Material *Other*: `https://doi.org/10.5281/zenodo.15623903`

Funding The project leading to this publication has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement No. 101034440 and was supported by the Austrian Science Fund (FWF) within the project 10.55776/P36688.

1 Introduction

Over the last few years, computational methods have played an important role in mathematical discovery. On the one hand, machine-learning guided heuristic search has successfully established new results on the existence of certain combinatorial objects [22, 23]. On the

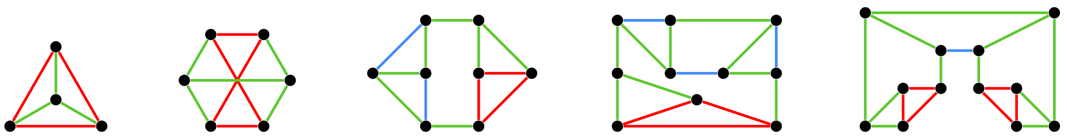

Figure 1 Examples of 3-decompositions of cubic graphs of order 4, 6, 8, 10, and 12. Cubic graphs only have an even number of vertices since the number of edges in a cubic graph with n vertices is $3n/2$. Note that it is possible to have an empty matching, as is shown in the first two examples.

other hand, for non-existence or optimality results (like what is the smallest object with a certain property), where the search space must be explored rigorously, constraint-based methods have been successful [5, 7, 8, 14, 17, 25].

In this paper, we tackle the *3-Decomposition Conjecture (3DC)* with constraint-based methods. The conjecture, formulated by Hoffmann-Ostenhof [6] in 2009, is an open problem in discrete mathematics that attracted a lot of research interest [9]. The conjecture states that every connected cubic graph (each vertex has exactly three neighbors) can be partitioned into three edge-disjoint subgraphs: a spanning tree, a collection of cycles, and a matching. Figure 1 shows several small cubic graphs with the 3-decomposition indicated by colors: green edges represent the spanning tree, red edges the cycles, and blue edges the matching. The conjecture has been verified for connected cubic graphs that are planar, Hamiltonian, traceable, claw-free, or 4-chordal, and for connected cubic graphs of treewidth ≤ 3 [4, 15, 2, 19, 1]. However, these results on special cases only cover a small fraction of all connected cubic graphs for which, despite considerable efforts, the conjecture remains open.

Our main results are as follows.

(1) We verify the 3DC for all connected cubic graphs up to 28 vertices with constraint-based methods.

(2) We systematically compute a list of graphs with up to 18 vertices such that a minimal counterexample to the 3DC cannot contain any graph from the list as a subgraph.

(3) We make an experimental study on how much the use of the graphs from (2) speeds up the search for a 3DC counterexample.

Result (1) is a significant improvement over the known bound of 20 which was established by listing all connected cubic graphs up to $n = 20$ vertices, modulo isomorphism, and checking for each of them whether it has a 3-decomposition [4]. There are $f(20) = 510.489$ such graphs [11]; hence, this generate-and-test approach is feasible. However, for larger n, this becomes quickly infeasible as we have $f(22) > 7.3 \times 10^6$, $f(24) > 1.1 \times 10^8$, $f(26) > 2.0 \times 10^9$, and $f(28) > 4.0 \times 10^{10}$. Our approach circumvents this obstacle by integrating search and test with a combination of SAT encodings with specialized propagators.

Result (2) yields a complete list of all *reducible templates*, a specific type of subcubic graphs, up to index 6 and order 18, from which we deduce the list of subgraphs that a minimal counterexample to the 3DC cannot contain. Some reducible templates have already been identified by Bachtler et al. [4], but our search yields new instances. Observe that Result (2) has a direct effect on Result (1): if we know the forbidden subgraphs, then we can use them to prune the search tree for a 3DC counterexample and thus make the search for (1) more efficient. While Result (1) confirms the conjecture for a finite number of graphs (modulo isomorphism), Result (2) applies to minimal counterexamples of any size, i.e., to an infinite number of graphs (modulo isomorphism), hence can be considered stronger from a mathematical point of view.

Excluding subgraphs during the search requires repeated subgraph testing, known to be NP-complete. We efficiently accomplish this by tightly integrating the Glasgow Subgraph Solver [21] into our method. However, as these tests are frequent, they become computationally costly. Hence, we face a tradeoff: when is the subgraph testing too costly to outweigh the speedup achieved by search space pruning? Result (3) gives a systematic insight into this question. We provide rigorous experimental data that allows us to pinpoint a good balance between the two conflicting factors. Our analysis shows that incorporating forbidden subgraph constraints derived from smaller cases can drastically reduce search times for larger cases. We report on the specific performance improvements for graphs of various sizes.

Our work illustrates how SAT and CP techniques can be combined to tackle challenging combinatorial problems. Our dynamic forbidden subgraph approach naturally extends to many graph conjecture verification tasks where "minimality" can be meaningfully defined. Rather than pre-computing constraints, our method discovers forbidden patterns during the search process itself, creating increasingly powerful pruning rules as exploration progresses. This approach is directly applicable to coloring problems, extremal graph theory conjectures, and Ramsey-type problems, where minimal counterexamples must satisfy specific structural properties. The key insight is that local patterns discovered in one search branch can inform constraints for the entire remaining search space. This "learn-as-you-go" paradigm is particularly valuable for problems where theoretical analysis alone cannot easily identify all forbidden patterns, potentially transforming how we approach computational verification across combinatorial mathematics.

2 Preliminaries

For positive integers $k < \ell$, we write $[k] = \{1, 2, \ldots, k\}$, and $[k, \ell] = \{k, \ldots, \ell\}$.

CNF formulas. A *literal* is a (propositional) variable x or a negated variable $\overline{x}$. A *clause* is a disjunction of a finite set of literals. A *formula* in *conjunctive normal form (CNF)* or *clausal normal form* is a conjunction of a finite set of clauses.

Simple graphs. A *simple graph* is a graph without parallel edges or self-loops. In this paper, the term *graphs* refers to *simple graphs* unless otherwise specified. A simple graph G consists of a set $V(G)$ of vertices and a set $E(G)$ of edges; we denote the edge between vertices $u, v \in V(G)$ by uv or, equivalently, vu. We thus write (V, E) to denote a graph, where V is the vertex set and E the edge set. Two vertices $u, v \in V(G)$ are *adjacent* if $uv \in E(G)$. The *degree* of a vertex $v \in V(G)$, denoted as $\deg v$, is the number of vertices in G that it is adjacent to. For $k \geq 0$, a k-regular graph is a graph where each vertex has exactly degree k. A 3-regular graph is also called a *cubic* graph. A graph where each vertex has at most 3 neighbors is a *subcubic* graph. A graph is *connected* if there is a path between any distinct pair of vertices. A *cycle* is a connected 2-regular graph. A *matching* is a graph where no two edges share a common vertex. A graph is a *tree* if there is a unique path between any pair of distinct vertices. A graph T is a *spanning tree* of a graph G if T is a tree, $V(T) = V(G)$, and $E(T) \subseteq E(G)$. A graph H is a *subgraph* of G if there is an injective mapping $\varphi : V(H) \to V(G)$ such that for any $u, v \in V(H)$, if $uv \in E(H)$ then $\varphi(u)\varphi(v) \in E(G)$. H is an *induced subgraph* of G if the mapping above additionally satisfies that for any $u, v \in V(H)$, $uv \in E(H)$ if $\varphi(u)\varphi(v) \in E(G)$. If two graphs are induced subgraphs of each other, then we say the two graphs are *isomorphic*, and we call the witnessing mappings *isomorphisms*. Given $E \subseteq E(G)$, the subgraph of G *induced* by E is the graph H, where $V(H) := \bigcup_{uv \in E}\{u, v\}$ and $E(H) := E$. The *adjacency matrix* A of a graph G with $|V(G)| = n$ is the $n \times n$ $\{0, 1\}$-matrix where the element at row v and column u, denoted by $A(v, u)$, is 1 if and only if $vu \in E(G)$. Let $e(i, j)$ denote the propositional *edge variable* for the possible edge between vertices i and j. If $e(i, j) = 1$, then the possible edge is present; otherwise, the possible edge is absent.

SAT modulo symmetries (SMS). SMS [18] is a framework that augments a CDCL (conflict-driven clause learning) SAT solver [10, 20]. The framework is specifically designed for enumerating graphs that satisfy given properties expressed in the form of a formula in propositional logic. Usually, properties of graphs that we take into consideration are invariant under isomorphisms, and SMS benefits from only considering one adjacency matrix out of the numerous different adjacency matrices that represent the same graph (modulo isomorphism). SMS implements this idea by connecting a CDCL SAT solver to an external propagator, which checks from time to time whether the current assignment can be extended to a lexicographically minimal (*canonical*) matrix (more precisely, only those copies are kept that are when considering the rows of the adjacency matrix concatenated into a single vector), and gives feedback to the CDCL solver in the form of *symmetry breaking clauses* accordingly. This synergy between the CDCL solver and the external propagator is not restricted to generating canonical matrices. In fact, customized external propagators can be added to learn clauses and trim the search tree to suit the specific task at hand. As of today, SMS has incorporated many commonly sought-after functionalities, such as generating graphs of certain connectivity, graphs without certain subgraphs, and so on. It is possible to specify a partition of the vertex set and restrict the symmetry breaking to those permutations that preserve the partition. For a full description of SMS, we refer to the original work where the framework was introduced [18].

Partially defined graphs. The notion of partially defined graphs was introduced in the context of SMS [17] to capture the combinatorial object represented by a partial truth assignment over the edge variables of a graph. A *partially defined graph* is a graph where some edges are undefined in the sense that their presence in the graph is open. Formally, a partially defined graph G is a graph whose edge set $E(G)$ is split into disjoint sets $E_d(G)$ and $E_n(G)$, the sets of defined edges and defined non-edges, respectively. G is *fully defined* if $uv \in E_d(G) \cup E_n(G)$ for any $u, v \in V(G)$ such that $u \neq v$.

Partially defined subgraphs occur naturally in the process of SAT solving, and many graph properties can already be shown when we only know for sure some edges exist (or do not exist) in a given graph, e.g., the existence of certain (induced) subgraphs. Thus, the benefits of considering partially defined graphs during SAT-based graph enumeration are at least twofold. First, if we can falsify the desired property in a partially defined graph, then we can block the corresponding branch of the search tree early on and speed up the search. Second, properties that are not in the class NP are usually hard to express concisely in CNF, and therefore can introduce enormous time overhead for the solver due to the large size of the formula. An external propagator, therefore, serves as a way to circumvent the clumsy encoding because it only gives the clauses for the desired properties to the solver when these clauses become useful in blocking the search path.

3 Verifying the conjecture up to $n = 28$

Let's first give a formal statement of the conjecture.

▶ **Conjecture 1** (3-Decomposition Conjecture, 3DC). *Every connected cubic graph can be decomposed into a spanning tree T, a collection of cycles C, and a (possibly empty) matching M.*

We verify the conjecture for all connected cubic graphs with n vertices up to $n = 28$. We achieve this by performing an exhaustive search for counterexamples to the conjecture through all cubic graphs with n vertices, starting with $n = 4$ and increasing by 2 each time since all cubic graphs have an even number of vertices, until we reach $n = 28$. Thus, we can safely assume that if a counterexample is found, it is a minimal counterexample. Given a

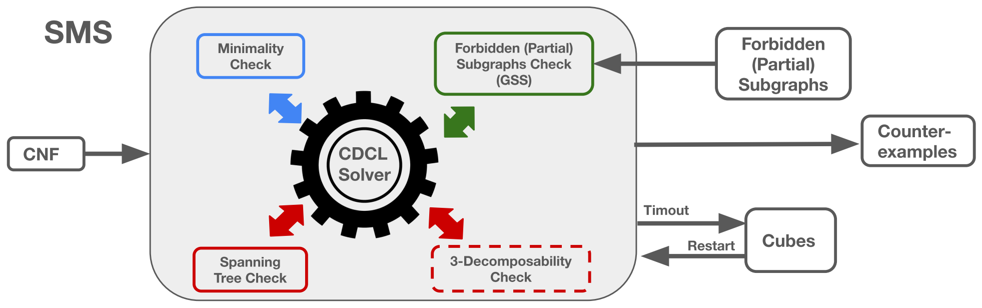

Figure 2 A roadmap of the process for verifying the 3-decomposition conjecture up to $n = 28$.

fixed n, we give the following CNF formula to SMS, augmented with specialized propagators.

$$F_{\text{cubic}}(n) := \bigwedge_{i\in[n]} \sum_{j\in[n],j\neq i} e(i,j) = 3.$$

$F_{\text{cubic}}(n)$ describes the constraints for a graph with n vertices to be cubic. The summation sign here represents a cardinality constraint, which specifies how many variables in a given set are set to true. We express cardinality constraints using cardinality networks [3]. The CNF formula and the propagators together ensure that SMS outputs all and only counterexamples, i.e., cubic graphs with n vertices that are without a 3-decomposition. This way, if SMS outputs no graphs, then we know that the conjecture holds for all graphs with n vertices. For larger cases, a timeout of one day is set. When this timeout is reached, the solving stops and various *cubes*, which are sets of literals that represent subproblems split from the original problem [13], are generated. We then restart the solver with each cube as a set of assumptions. We do this iteratively until all cubes are solved before the timeout. This process is visualized in Figure 2.

3.1 Propagators

As is shown in Figure 2, we use in total four external propagators in combination with the CDCL SAT solver. The propagator that checks whether the current graph has a 3-decomposition is only activated for checking fully defined graphs, and is enclosed in dashed lines in the figure. The other three propagators are activated for checking partially defined graphs, to which fully defined graphs also belong, and are enclosed in solid lines. The minimality check, shown in blue, is the default propagator in SMS and is responsible for filtering out non-canonical adjacency matrices as early as possible in the process of generation. The other three propagators are either modified or newly added to SMS, and we will now explain them in detail.

Forbidden (partial) subgraphs check. The forbidden (partial) subgraph check, shown in green in Figure 2, uses the Glasgow Subgraph Solver (GSS) to forbid the generation of graphs containing certain subgraphs. This functionality has already been realized in previous iterations of SMS [16]. We modified it to allow partially-defined graphs to be forbidden as subgraphs. Recall that a graph H is a *subgraph* of G if there is an injective mapping $\varphi : V(H) \to V(G)$ such that for any $u, v \in V(H)$, if $uv \in E(H)$ then $\varphi(u)\varphi(v) \in E(G)$. If H is a subgraph to be forbidden, then when the forbidden subgraphs check detects such a mapping φ, it sends the clause $\{\,\overline{\varphi(u)\varphi(v)} : uv \in E(V)\,\}$ to the SAT solver.

For our purposes, we define a relation similar to the subgraph relation for partially defined graphs. Given two partially defined graphs G and H, we say that G is a *partial subgraph* of H if there is a mapping $\varphi : V(G) \to V(H)$ such that for any $u, v \in V(G)$ we have that $\varphi(u)\varphi(v) \in E_d(H)$ if $uv \in E_d(G)$ and $\varphi(u)\varphi(v) \in E_n(H)$ if $uv \in E_n(G)$. Note that a fully defined graph G can also be seen as a simple graph with $E(G) = E_d(G)$. Given two fully defined graphs G and H, the subgraph relations differ when they are seen as simple graphs and when they are seen as partially defined graphs. In fact, G is a partial subgraph of H if and only if G is an induced subgraph of H. The reason for this definition is that we want it to be the case that G is a partial subgraph of H witnessed by φ if and only if given any way to assign the undefined edges in H to derive H', there is a way to assign the undefined edges in G to derive G', such that G' is an induced subgraph of H' witnessed by φ. If H is a partial subgraph to be forbidden, then when the forbidden partial subgraphs check detects such a mapping φ, it sends the clause $\{\, \overline{\varphi(u)\varphi(v)} : uv \in E_d(V) \,\} \cup \{\, \varphi(u)\varphi(v) : uv \in E_u(V) \,\}$ to the SAT solver.

In practice, the set of forbidden (partial) subgraphs we use are transformed from results by Bachtler et al. [4]. The details of the transformation are explained in Section 4.3.

Spanning tree check and 3-decomposability check. The two red propagators in Figure 2 are new additions to SMS, and both are specifically designed to filter out graphs that have a 3-decomposition. Both propagators are based on the following Theorem 2.

▶ **Theorem 2.** *Let G be a counterexample to the 3-decomposition conjecture, and T be a spanning tree of it. Then there is an edge in $E(G) \setminus E(T)$ that connects a vertex of degree 1 and another of degree 2 in T. In other words,*

$$B(T) := \{\, uv : u, v \in V(T), uv \notin E(T), deg_T(u) = 1, deg_T(v) = 2 \,\} \neq \emptyset.$$

Proof. Assume the contrary. Let H be a simple graph such that $V(H) := \{v \in V(G) : \deg_T(v) < 3\}$ and $E(H) := E(G)/E(T)$. By definition, for all $v \in V(H)$, we have $\deg_H(v) = 3 - \deg_T(v)$. Since we assume that for all $uv \in E(H)$, either $\deg_T(u) = \deg_T(v) = 1$ or $\deg_T(u) = \deg_T(v) = 2$, it follows that $\deg_H(u) = \deg_H(v) = 1$ or $\deg_H(u) = \deg_H(v) = 2$. This means that H is a union of a 2-regular graph and a 1-regular graph, and therefore, we have a 3-decomposition. This contradicts our assumption. Therefore, $B(T) \neq \emptyset$. ◀

Given a spanning tree T, define the formula

$$\mathrm{OneTwo}(T) := \left(\bigwedge_{uv \in E(T)} e(u, v) \right) \to \left(\bigvee_{uv \in B(T)} e(u, v) \right).$$

OneTwo(T) states that if all edges of the spanning tree T are present in the graph, then at least one edge that connects a vertex of the tree of degree one and another of degree two must also be present in the graph. The spanning tree check works with partially defined graphs. Given a partially defined graph G, it searches for a spanning tree T consisting of defined edges. If it finds one, and $B(T) \cap E(G) = \emptyset$, it sends the solver the formula OneTwo(T); otherwise, it does nothing. The 3-decomposition check works only with fully defined graphs, and searches for a 3-decomposition for a fully defined graph exhaustively. If it finds a decomposition (T, C, M) (note that it is necessary that $B(T) \cap E(G) = \emptyset$ due to Theorem 2), then it sends to the solver OneTwo(T); otherwise, it does nothing.

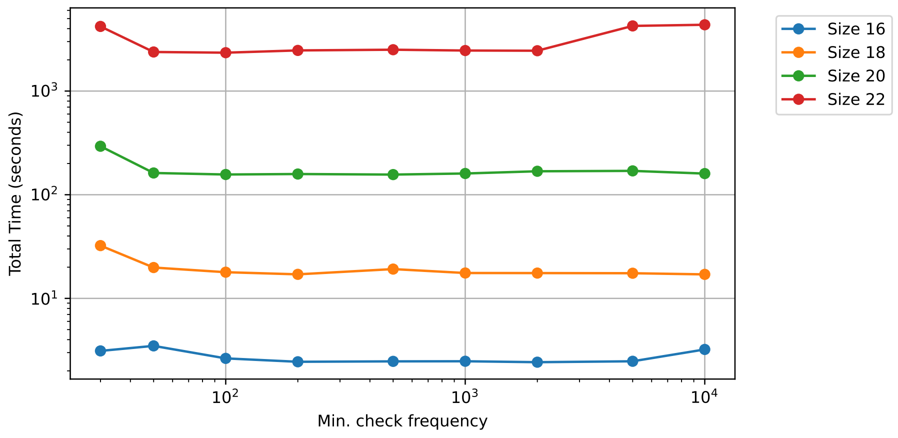

Figure 3 Time spent in solving the case of $n \in \{16, 18, 20, 22\}$ with the frequency of minimality check $f \in \{30, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}$.

3.2 Results

We implemented the approach with the four propagators as mentioned above and verified the conjecture for every even $4 \leq n \leq 28$. The time spent in each n is shown in Table 1[1]. The frequency for the minimality check for all cases in the table is the default value of 30. In Table 2, we provide statistics of the propagators for the bigger cases. In figure 3, we compare the time spent in solving the middle cases with the minimality check invoked with different frequencies. As we can see, in general, the time first decreases and then increases as we invoke the minimality check more seldom, but the difference is not significant.

Table 1 Time spent verifying that all connected cubic graphs with n vertices have a 3-decomposition.

n	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**	**26**	**28**
time	< 1ms	2.6ms	6.1ms	45ms	0.23s	0.92s	3.1s	32s	4.9m	1.2h	20h	26d	4.0y

▶ **Theorem 3.** *The 3DC holds for all connected cubic graphs up to 28 vertices.*

4 Search for forbidden (partial) subgraphs

In this section, we search for patterns that cannot appear in a minimal counterexample to the conjecture. This is useful because if we know certain patterns cannot appear in a counterexample, we can close off the search branch once we detect the subgraph in the partially defined graphs and thereby shorten the search time. We search for particular pairs

[1] The computational tasks throughout the paper were carried out on a Sun Grid Engine (SGE) cluster consisting of heterogeneous machines running Ubuntu 18.04.6 LTS. The cluster contains nodes with the following architectures: 2× Intel Xeon E5540 with 2.53 GHz Quad Core, 2× Intel Xeon E5649 with 2.53 GHz 6-core, 2× Intel Xeon E5-2630 v2 with 2.60GHz 6-core, 2× Intel Xeon E5-2640 v4 with 2.40GHz 10-core and 2× AMD EPYC 7402 with 2.80GHz 24-core. All relevant scripts and code can be found at `https://doi.org/10.5281/zenodo.15623903`.

Table 2 Statistics of the propagators for the cases $n \in \{18, 20, 22, 24, 26, 28\}$ of Table 1. min. is short for minimality check, forb. is short for forbidden subgraph check, span. is short for spanning tree check, and 3-dec. is short for 3-decomposition check. For each n, three columns of data are given. The first column shows the time each propagator uses. The second column shows the percentage that time takes up in the total solving time. The third column shows the percentage of the propagator calls that yield learned clauses.

n	**18**			**20**			**22**		
min.	11s	35%	67%	1.5m	29%	52%	15m	21%	35%
forb.	10s	32%	29%	2.0m	41%	15%	28m	40%	5.0%
span.	93ms	2.9‰	1.9‰	1.7s	5.6‰	< 1‰	34s	8.0‰	< 1‰
3-dec.	0.28s	8.7‰	100%	7.9s	2.7%	100%	4.2m	6.1%	100%
n	**24**			**26**			**28**		
min.	2.4h	12%	21%	3.0d	11%	11%	98d	6.8%	9.9%
forb.	4.7h	23%	1.2%	2.1d	7.8%	1.2%	80d	5.5%	9.9‰
span.	10m	8.5‰	< 1‰	2.5h	3.8‰	< 1‰	3.8d	2.6‰	< 1‰
3-dec.	2.4h	12%	100%	8.5d	32%	100%	2.9y	73%	100%

of patterns called *reducible extensions*, which are pairs of subcubic graphs X and Y with $|V(X)| < |V(Y)|$, such that if we have a counterexample containing Y, then we can always obtain a smaller counterexample by replacing Y with X. This way, Y cannot appear as a subgraph in a minimal counterexample to the 3-decomposition conjecture.

We start with reviewing some existing concepts and definitions that we built our concepts on. In particular, we introduce the concept of *templates*, *reducible extensions*, and *reducible templates*, and a sufficient condition for automatically testing whether a pair of subcubic graphs belongs to the latter. Then, we show that the said condition can be relaxed. Finally, we implement a search procedure using both the relaxed condition and SMS and obtain all reducible templates up to a certain size measure.

4.1 Templates, replacement, coloring, and possible behaviors

In this section, we review some useful definitions and theorems from Bachtler et al. [4]. Some of the definitions have been slightly modified to better formulate the concepts we work with.

Templates. A *template* is a graph X whose vertex set is partitioned into a set of *inner vertices* $I(X)$ and a set of *outer vertices* $O(X)$, such that all inner vertices have 3 neighbors and all outer vertices have 1 neighbor. For any $x \in O(X)$, we write $a_X(x)$ to denote the unique vertex in $I(X)$ that x is adjacent to. The *order* of a template is defined as its number of vertices, and the *index* of a template is its number of outer vertices. Let X and Y be two templates with $O(X) = O(Y)$. Let G be a cubic graph. We say template X is in G or G contains X as a template if there exists a mapping $\varphi : V(X) \to V(G)$ such that for any $x, y \in V(X)$, we have if $x \in I(X)$, then $\varphi(x) \neq \varphi(y)$ and $xy \in E(X)$ if and only if $\varphi(x)\varphi(y) \in E(G)$. We call such a mapping a *witness*.

Intuitively, a template is intended as a representation of "a piece of" a cubic graph where the inner vertices together with the edges among them form an induced subgraph of the entire graph, and the outer vertices correspond to all the other vertices that the vertices in this induced subgraph are adjacent to. We have defined what it means to be "a piece of" a cubic graph, and now we define what it means to replace one piece of a cubic graph with another.

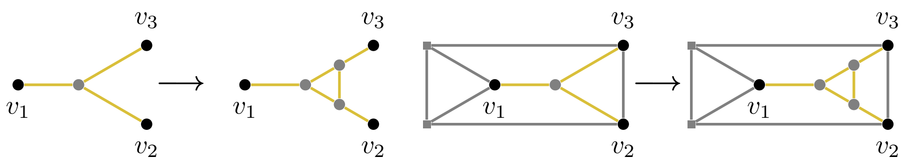

Figure 4 Examples of templates and replacement. On the left are two templates with the same set of outer vertices. The outer vertices are in black and the inner vertices gray. On the right, we show the result of replacing the first template with the second in a cubic graph of original size 6. The vertices that are not part of the templates are shown as gray squares.

Replacement. Suppose G contains X as a template. Then we can replace X in G with Y and get $G_\varphi[X \to Y]$ where $V(G_\varphi[X \to Y]) := (V(G)/\varphi(I(X))) \cup I(Y)$ and $E(G_\varphi[X \to Y]) := \{ uv : u,v \in V(G)/\varphi(I(X)), uv \in E(G) \} \cup \{ xy : x,y \in I(Y), xy \in E(Y) \} \cup \{ \varphi(x)y : x \in O(Y), y \in I(Y), xy \in E(Y) \}$. When it is not necessary to spell out the mapping φ, we also write $G[X \to Y]$ for $G_\varphi[X \to Y]$. A pair (X,Y) of templates with $O(X) = O(Y)$ is called an (X,Y)-*transformation*. Transformations are called *extensions* if $|V(X)| < |V(Y)|$. The *order* and *index* of an extension (X,Y) is defined as the order and index of Y.

Figure 4 contains an example for the concepts of template and replacement. To reformulate the goal mentioned at the beginning of this section, we want to find pairs of templates X and Y such that if we have a counterexample containing Y, then we can always obtain a smaller counterexample by replacing Y with X. Now, we rigorously define such pairs of templates in the notion of *reducible extensions.*

Reducible extensions. An (X,Y)-extension is *3-compatible* if for every cubic graph G with a 3-decomposition and for every extension $H := G[X \to Y]$ of G, H also has a 3-decomposition. Given a (Y,X)-reduction, a mapping $\varphi : V(Y) \to V(G)$ is called *permissible* if $\varphi(x) = \varphi(y)$ implies $a_X(x) \neq a_X(y)$ for all $x,y \in O(Y)$. A 3-compatible (X,Y)-extension is called a *reducible extension* if, for any connected cubic graph G containing Y as a template witnessed by a permissible mapping for (Y,X), we have that $G_\varphi[Y \to X]$ is also a connected cubic graph. Such a template Y is called *reducible.* The reason why we can forbid reducible templates in our search for counterexamples to the 3DC is formally stated in the following Theorem 4.

▶ **Theorem 4.** *Let (X,Y) be a reducible extension and G be a connected cubic graph that contains Y witnessed by a permissible mapping for (Y,X). Then G cannot be a minimal counterexample to the 3-decomposition conjecture.*

Theorem 4 states why reducible extensions are useful, but we still lack a way to compute them, since a template can appear in infinitely many cubic graphs and thus it is impossible to test if two templates stand in the relation of reducible extensions by enumerating all cubic graphs they appear in. This problem is solved in Bachtler et al. in the following way. First, note the following proposition that allows us to filter out reducible extensions from 3-compatible extensions.

▶ **Proposition 5.** *Let (X,Y) be a 3-compatible extension. If X is connected, then (X,Y) is a reducible extension.*

Now, our task is shifted to finding 3-compatible extensions. Bachtler et al. introduced a sufficient condition for 3-compatible extension that only requires analysing the structure of the templates themselves. The idea is that even though a template X can appear in infinitely

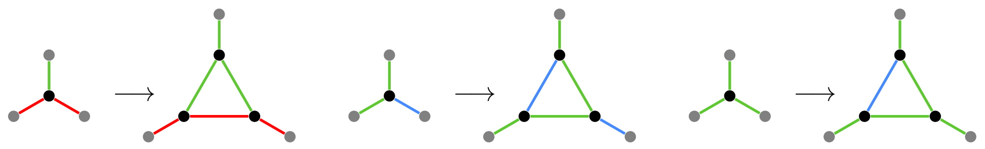

Figure 5 An example of applying Theorem 8 to determine membership of 3-compatibility. We use the same two templates as in Figure 4. We list all three possible behaviors of the smaller template (modulo reordering of the outer vertices), and each of the corresponding possible behaviors of the big template is given on the right.

many cubic graphs, there are only finitely many ways a template X can be decomposed as a part of a 3-decomposition. We call these ways *possible behaviors* of X. So if for each the possible behavior of X, there is a corresponding possible behavior of Y, such that the 3-decomposition still holds after replacement, then we can safely say that (X, Y) is a 3-compatible extension. This reasoning is rigorously formulated in Proposition 6, Corollary 7, and Theorem 8. We write RGB for the set $\{\text{red}, \text{green}, \text{blue}\}$. A *3-coloring* of a graph G is a mapping $cl : E(G) \to \text{RGB}$. A 3-decomposition can be seen as a 3-coloring, where the green edges represent the edges of the spanning tree, the red edges the cycles, and the blue edges the matching. We have the following proposition.

▶ **Proposition 6.** *A 3-coloring of a cubic graph is a 3-decomposition if and only if*

1. *The list of colors of the 3 incident edges of every vertex are either (green, green, green), (blue, green, green), and (green, red, red), and*
2. *the green edges form a spanning tree over the cubic graph.*

Proposition 6 gives us the following corollary.

▶ **Corollary 7.** *Given a template X and a 3-coloring over $E(X)$, the following conditions are necessary for the coloring to be a 3-decomposition over some cubic supergraph of X, restricted to $E(X)$.*

1. *The list of colors of the 3 incident edges of every $v \in I(X)$ are either [green, green, green], [blue, green, green] or [red, red, green], modulo reordering.*
2. *The green edges form a spanning forest, and each tree in the spanning forest contains at least one vertex from $O(X)$.*

Possible behaviors. We call any 3-coloring of X that satisfies the conditions in Corollary 7 a *possible behavior* of X. Let X, Y be two templates with $O(X) = O(Y)$ and $|X| < |Y|$. For any $v \in O(X) = O(Y)$, write $c(v)$ for the color of the unique edge incident to v in the coloring c. For a possible behavior c of X, define a partition P_c of $O(X)$ in which two vertices in $O(X)$ are in the same equivalence class if they are in the same tree of the spanning forest given c.

▶ **Theorem 8.** *Let X and Y be templates with $O(X) = O(Y) = O$ and $|V(X)| < |V(Y)|$. (X, Y) is a 3-compatible extension if for any possible behavior c of X, there is a possible behavior c' of Y satisfying the following conditions.*

1. *For any $v \in O$, $c'(v) = c(v)$.*
2. *$P_{c'} = P_c$.*

An example of applying Theorem 8 to determine membership of 3-compatibility is shown in Figure 5

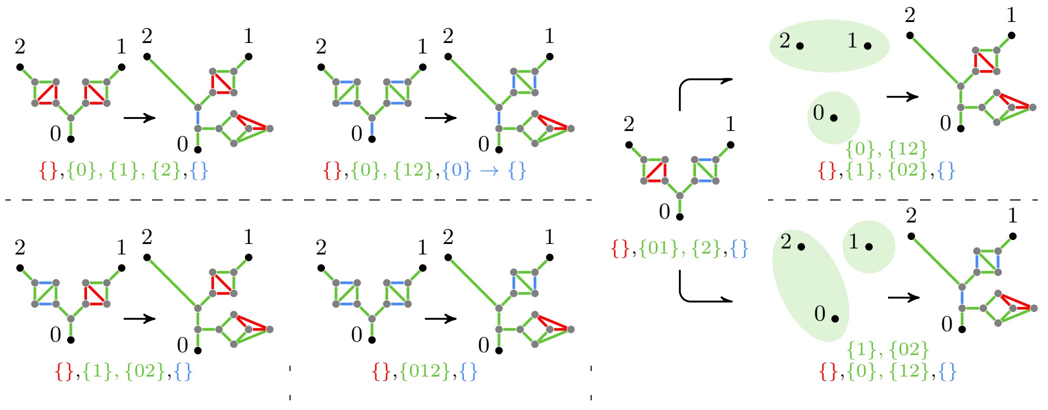

Figure 6 An example showing how Theorem 13 can be applied to affirm a reducible extension. All possible behaviors of the smaller template (on the left) come down to five different configurations in terms of the set of outer vertices adjacent to a red edge (displayed in red), the partition the green forest gives rise to over the outer vertices (displayed in green), and the set of vertices adjacent to a blue edge (displayed in blue). The larger template (on the right) has a possible behavior for four of the configurations with a spanning forest that gives rise to the same partition over the outer vertices. For the last configuration, the larger template does not have this, but instead has a corresponding possible behavior for each possibility of how the rest of the spanning tree can be.

4.2 Relaxing the Conditions

In this section, we show that the condition in Theorem 8 can be further relaxed. The key observation is as follows. Given templates X and Y with $O(X) = O(Y)$ and $|V(x)| < |V(Y)|$ and a possible behavior c of X, (X, Y) can still be a 3-compatible extension even if Y does not have a possible behavior c' such that $P_c = P_{c'}$. In fact, Y only needs to have a corresponding possible behavior for each of the possibilities that the rest of the spanning tree can be. We introduce Lemmas 9 and 11 that set the foundation for Definition 12, in which we rigorously characterize the description above. Then, we end this section with the relaxed condition and proof of its validity. Due to space limits, the proofs for this section are shown in Appendix A.

▶ **Lemma 9.** *Given two trees T_1 and T_2 with $E(T_1) \cap E(T_2) = \emptyset$, the combined graph $G := (V(T_1) \cup V(T_2), E(T_1) \cup E(T_2))$ is a tree if and only if $|V(T_1) \cap V(T_2)| = 1$.*

▶ **Definition 10.** *Let $\mathcal{S} = S_1, S_2, ..., S_n$ be a finite collection of sets. $\mathcal{S}$ is* mergeable *if there exists a permutation a of $[n]$ such that for each $i \in [n-1]$, $|(\bigcup_{j=1}^{i} S_{a(j)}) \cap S_{a(i+1)}| = 1$.*

▶ **Lemma 11.** *Let S be a set of vertices and $T_1, T_2, ..., T_n$ a set of trees such that $V(T_i) \cap S \neq \emptyset$ and $V(T_i) \cap V(T_j) \subseteq S$ for distinct $1 \leq i, j \leq n$. Define the combined graph $G := (\bigcup_{i=1}^{n} V(T_i), \bigcup_{i=1}^{n} E(T_i))$. Then G is a tree if and only if $\mathcal{S} := \{ S_i := V(T_i) \cap S : i \in [n] \}$ is mergebale.*

▶ **Definition 12.** *Given a partition $\mathcal{S}$ of a set S, we write $Comp(\mathcal{S})$ to denote the set of partitions $\mathcal{S}'$ of S such that $\mathcal{S} \cup \mathcal{S}'$ is mergeable.*

Now we present the following Theorem 13, which relaxes Theorem 8.

▶ **Theorem 13.** *Let X and Y be templates with $O(X) = O(Y) = O$ and $|V(X)| < |V(Y)|$. (X, Y) is a 3-compatible extension if for any possible behavior c of X and partition $P \in Comp(P_c)$, there is a possible behavior c' of Y satisfying the following conditions.*

1. *For any $v \in O$, $c'(v) =$ green if $c(v) =$ green; $c'(v) =$ red if and only if $c(v) =$ red.*
2. *$P_{c'} \in Comp(P)$.*

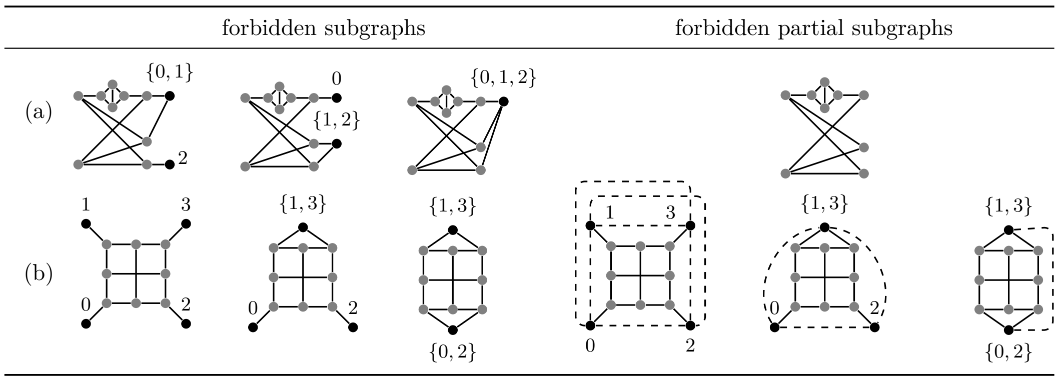

Figure 7 Two examples of the set of forbidden (partial) subgraphs we obtain from reducible extensions. Both of the reducible extensions are new discoveries, and are shown in Figure 9.

Theorem 13 and Proposition 5 give the criterion by which we search for reducible extensions using a C++ program. The program itself and the related toolchain will be explained in Section 4.4. An example of applying Theorem 13 to affirm a reducible extension is shown in Figure 6.

4.3 Converting reducible extensions to forbidden (partial) subgraphs

In this section, we explain how to convert reducible extensions into forbidden subgraphs and partial subgraphs. Recall that templates are intended as a representation of "a piece of" a graph where the inner vertices together with the edges among them form an induced subgraph of the entire graph, and the outer vertices correspond to all the other vertices that the vertices in this induced subgraph are adjacent to. The restriction that each outer vertex is adjacent to a unique inner vertex and none of the outer vertices allows us to consider fewer cases when enumeration such "pieces of a graph", but this also means that we need to consider the possibility that multiple outer vertices can represent the same vertex in the entire graph. We now describe in detail how a reducible extension is converted to a set of forbidden subgraphs and forbidden partial subgraphs, respectively.

For any template Z, let $\mathrm{Part}(Z)$ be the partition of $O(Z)$ such that two outer vertices are in the same equivalence class if and only if the inner vertices that they are uniquely adjacent to are the same. Define $\overline{\mathrm{Part}}(Z)$ to be the set obtained by removing singletons from $\mathrm{Part}(Z)$, and write $\overline{O}(X)$ to denote $\bigcup \overline{\mathrm{Part}}(Z)$. For both cases, we start with a reducible extension (X, Y) with $O(X) = O(Y)$. Note that if distinct $u, v \in O(X)$ are in the same equivalence class of $Part(X)$ and G is a graph containing Y where u and v correspond to the same vertex in G, then $G[Y \to X]$ is not a simple graph, since it might contain parallel edges. In other words, the existence of the (partial) subgraph obtained by mapping u and v to the same vertex in a minimal counterexample is not ruled out by (X, Y).

To deduce the set of forbidden subgraphs from (X, Y), we enumerate all partitions P of $O(X)$ such that if distinct $u, v \in O(X)$ are in the same equivalence class in $\mathrm{Part}(X)$, then u and v are not in the same equivalence class of P. For each P, we construct a simple graph G_P where $V(G_P) := P \cup I(Y)$ and $E(G_P) := \{ uv : u, v \in I(Y), uv \in E(Y) \} \cup \{ pv : p \in P, v \in I(Y), \exists u \in p.\, uv \in E(Y) \}$.

To deduce the set of forbidden partial subgraphs from (X, Y), we enumerate all partitions $\overline{P}$ of $\overline{O}(X)$ such that if distinct $u, v \in \overline{O}(X)$ are in the same equivalent class in $\overline{\mathrm{Part}}(X)$, then u and v are not in the same equivalent class of $\overline{P}$. For each $\overline{P}$, we construct a partially defined

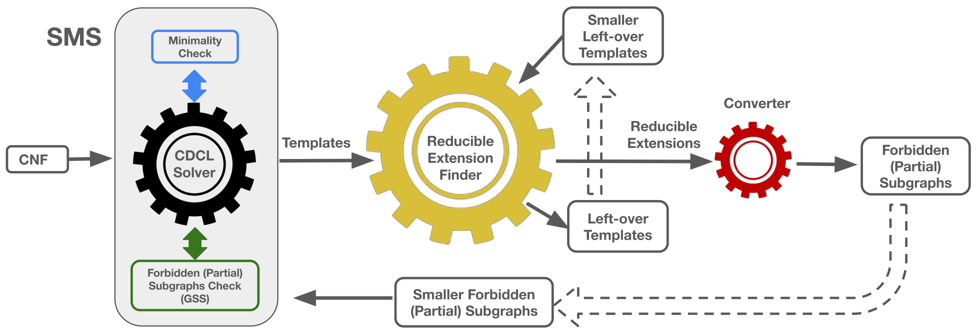
SMS
Minimality Check
CDCL Solver
Forbidden (Partial) Subgraphs Check (GSS)
CNF
Templates
Reducible Extension Finder
Smaller Left-over Templates
Left-over Templates
Reducible Extensions
Converter
Forbidden (Partial) Subgraphs
Smaller Forbidden (Partial) Subgraphs

Figure 8 The toolchain for searching for forbidden (partial) subgraphs.

graph $\overline{G_P}$ where $V(\overline{G_P}) := \overline{P} \cup I(Y)$, $E_d(\overline{G_P}) := \{\, uv : u, v \in I(Y), uv \in Y \,\} \cup \{\, pv : p \in \overline{P}, v \in I(Y), \exists u \in p.\, uv \in E(Y) \,\}$, and $E_n(\overline{G_P}) := \{\, uv : u, v \in I(Y), uv \notin E(Y) \,\} \cup \{\, pv : p \in \overline{P}, v \in I(Y), \forall u \in p.\, uv \notin E(Y), \forall x \in O(X)/\overline{O}(X).\, xv \notin E(Y) \,\}$.

We implement a toolchain for computing forbidden (partial) subgraphs and explain it in detail in the next section. For now, we give two examples for the set of forbidden (partial) subgraphs we obtain from newly discovered reducible extensions in Figure 7.

4.4 Computing reducible templates

In this section, we explain the toolchain we built for computing reducible extensions. The toolchain is visualized in Figure 8. It is run iteratively, each time searching for reducible extensions of a fixed pair of order and index. The precise sequence in which the iterations are carried out is shown in Table 3. The reason for this sequence is to get reducible templates with as few vertices and edges as possible. In total, three tools are used.

The first tool is SMS with the minimality check and the forbidden (partial) subgraph check activated. We set SMS to enumerate all graphs with n vertices and give SMS the following CNF formula.

$$F_{\text{temp}}(k, n) := \left(\bigwedge_{i \in [k]} \sum_{j \in [k], j \neq i} \neg e(i,j) \right) \wedge \left(\bigwedge_{i \in [k]} \sum_{j \in [k+1,n]} e(i,j) = 1 \right) \wedge \left(\bigwedge_{i \in [k+1,n]} \sum_{j \in [n], j \neq i} e(i,j) = 3 \right).$$

Here we postulate that vertices $v \in [k]$ are the k outer vertices and the rest are inner vertices. $F_{\text{temp}}(k, n)$ states the conditions for a graph to be a template with index k and order n.

The second tool is a *reducible extensions finder*, which is a separate C++ program that takes in templates and finds reducible extensions according to Theorem 13. Here, smaller templates that are connected (see Proposition 5) and that are not the bigger graph for any reducible extensions found in previous iterations (shortened as *left-overs* in the figure) are used as candidates for the smaller template, and the templates generated by SMS are used as candidates for the bigger graph.

The last tool is a *converter* that converts reducible extensions found into forbidden (partial) subgraphs, which we have explained in Section 4.3.

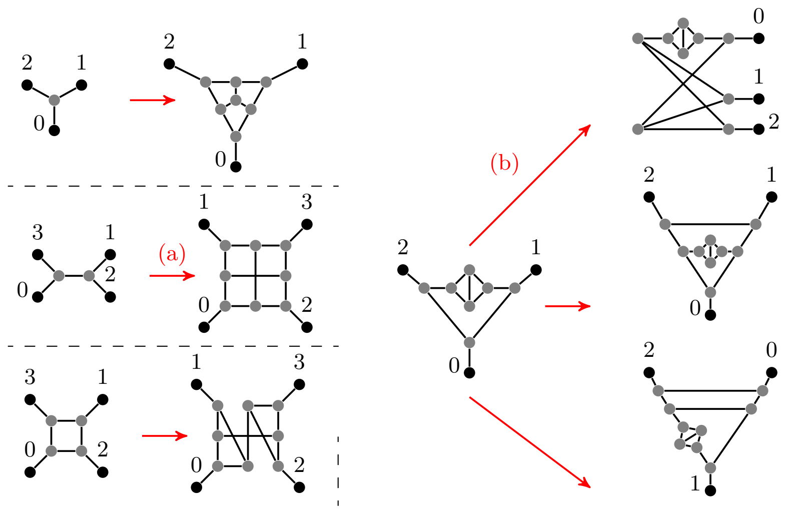

Figure 9 Newly discovered reducible extensions up to order 12. Reducible extensions with index < 3 are not shown, since all reducible extensions (X, Y) we found up to order 18 of index 2 have the unique smallest template of index 2 as X, and those with index 1 have the unique smallest template with index 1 as X.

By running this setup, we can determine all reducible templates we were looking for, and we arrive at the following main result of this section.

▶ **Theorem 14.** *The numbers of reducible templates of various indices and orders up to index* 6 *and order* 18 *are as shown in Table 3.*

Figure 9 shows the newly discovered reducible extensions up to order 12.

Table 3 The results of exhaustive search for reducible templates Y with index ≤ 6, up to order 18. Each row represents a different value for the number of vertices of the larger graph. Each row is divided into six groups, each group representing a different number of outer vertices. There are three numbers in each group. The first is the number of reducible extensions Y found. The second is the number of leftover templates. The third is the total number of templates generated by SMS. This table also shows the order of computation: on the same row, the groups are computed from left to right; across rows, the groups are computed from top to bottom. All graphs in the table can be found in the supplementary material.

$\lvert V(Y)\rvert$	$\lvert O(Y)\rvert$																	
	6			**5**			**4**			**3**			**2**			**1**		
4		-			-			-		0	1	1		-			-	
6		-			-		0	1	1	1	0	1	0	1	1	0	1	1
8		-		0	1	1	0	1	1	1	1	3	2	1	3	0	0	0
10	0	2	2	0	2	3	2	2	6	1	2	4	3	0	4	1	0	2
12	0	5	8	0	6	9	3	6	12	4	2	11	2	0	5	1	0	2
14	0	20	28	10	18	34	9	7	25	4	0	10	1	0	5	0	0	3
16	3	84	108	48	24	92	15	2	35	2	2	16	1	0	7	0	0	2
18	217	138	414	33	28	111	12	7	53	4	0	21	0	0	7	0	0	4

5 Speeding up the search with forbidden (partial) subgraphs

In this section, we test the effectiveness of forbidding (partial) subgraphs on shortening the search time for $n \in \{16, 18, 20, 22, 24\}$. We first gather all forbidden subgraphs (forbidden partial subgraphs, respectively) and order them in ascending order of the number of vertices. We then take the first $5 \cdot x$ forbidden (partial) subgraphs in the ordered list for $x \in [0, 25]$ and forbid them during our search. We also set the frequency at which the forbidden (partial) subgraphs check is called to freq $\in \{30, 100, 200, 500, 1000, 2000, 5000, 100000\}$. The forbidden (partial) subgraph check is called once per freq many edge variable decisions probabilistically. The time spent in search in each case is shown in Figure 10. The figure tells us the following.

1. Forbidding (partial) subgraphs generally reduces the search time, with the effect stronger as n increases.
2. The search time first decreases, then increases as the number of forbidden (partial) subgraphs grows. For subgraphs, the optimal number to forbid is around 25, and for partial subgraphs is 5.
3. Fixing the number of forbidden structures, the search with subgraphs forbidden is, in general, faster than the search with partial subgraphs forbidden. This is the result of two competing forces. On the one hand, partial subgraphs are more concise, with a forbidden partial subgraph more likely to rule out a partial assignment than a forbidden subgraph. On the other hand, partial subgraphs are represented as edge-labeled graphs in the Glasgow Subgraph Solver, which might lead to longer subgraph isomorphism testing times compared to unlabeled graphs. Apparently, the latter force overpowers the former in practice.
4. Except for $n \in \{16, 18\}$ with forbidden partial subgraphs, where the frequent call of the forbidden partial subgraph propagators visibly drags down the search speed, there is no uniform pattern between search time and the frequencies of subgraph isomorphism testing. This might be because, while subgraph isomorphism testing is time-consuming, frequent testing allows for earlier pruning of the search tree.

6 Conclusion and future work

In this work, we developed a SAT-based method enhanced by constraint programming techniques to verify the 3-Decomposition Conjecture for connected cubic graphs up to 28 vertices. We combined specialized propagators with forbidden subgraph checks to effectively prune the search space and reduce computation time. Our experiments provided clear insights into the trade-offs between the cost of constraint propagation and the benefits of early search pruning.

For future research, implementing a systematic and automated feature to gather and verify proofs and certificates generated by different tools in the SMS framework would be interesting. We explain here how it could work with respect to the tool chain presented in this paper.

The SMS uses CaDiCaL as the SAT solver, which is amenable to the production of DRAT [12]. To take into account the learned clauses produced by various specialized propagators, we can first run SMS with the propagators while collecting all learned clauses and then run CaDiCaL a second time with proof logging, feeding it the original formula together with all the recorded learned clauses. The DRAT proof produced this way can then be independently checked using verified proof checkers like DRAT-trim [24].

Besides generating a DRAT proof for the SAT solving process, we also need to verify that the propagators work the way intended and learn correct clauses. This is also doable given the SMS framework. The propagators are essentially approximation algorithms for NP problems

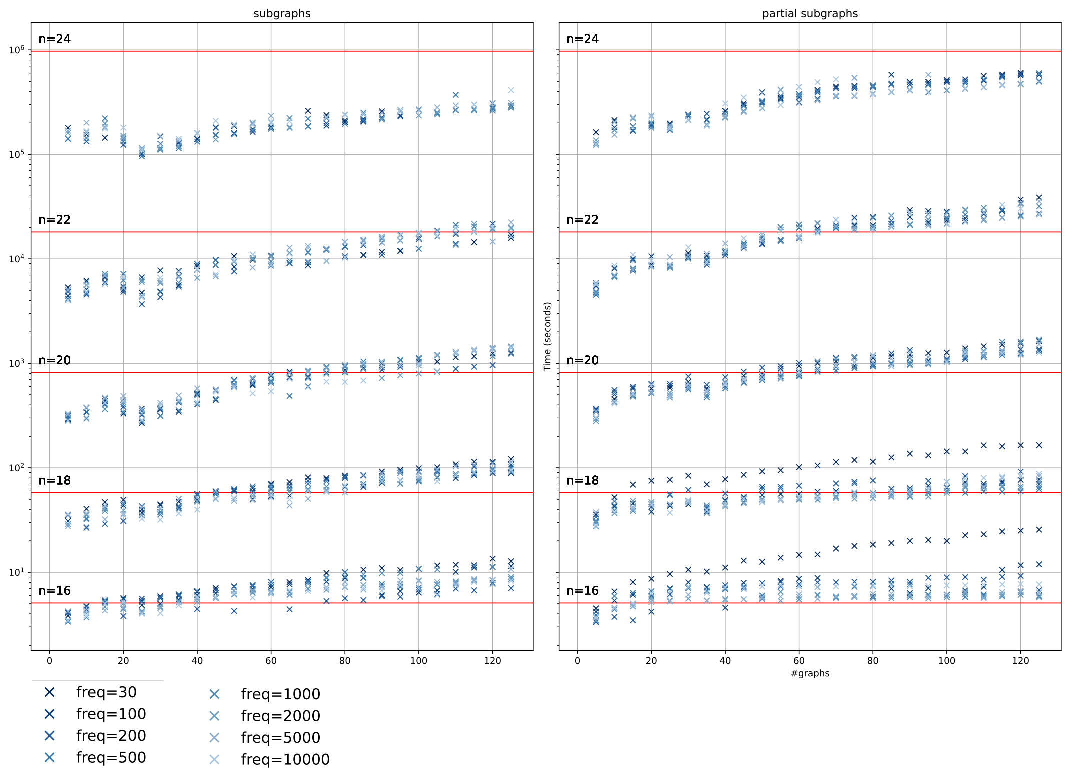

Figure 10 Time spent in the search for $n \in \{16, 18, 20, 22, 24\}$ forbidding different numbers of (partial) subgraphs with the forbidden (partial) subgraph check triggered with different frequencies. The data from forbidding subgraphs is shown on the left, and the data for forbidding partial subgraphs is shown on the right. The horizontal axis corresponds to the number of (partial) subgraphs forbidden, and the vertical axis corresponds to the time spent in search. Data points are marked as 'x', with the darker ones corresponding to the cases where the propagator is called more often. For each n, a red horizontal line is drawn at the height corresponding to the time spent in search without forbidding any (partial) subgraphs.

and are reliable when the answer is "yes" – they only learn clauses under that condition. Hence, a certificate can be generated each time a clause is learned, and these can be checked with a separate program. Note that NP certificates are simple to verify. Specifically for the tool chain in this paper, the minimality check can produce "nc-certificates" (see [18]); the forbidden (partial) subgraph check can output the mapping that witnesses subgraph isomorphism; the spanning tree check can output the spanning tree found that has $B(T) = \emptyset$; the 3-decompositon check can output 3-decompositions. In general, each propagator used in SMS can be equipped with a certificate-generating option and an accompanying verifier for verifying the certificates it produces.

Realizing a systematic way of verification for the SMS framework requires significant engineering that we leave for future work.

References

1 Elham Aboomahigir, Milad Ahanjideh, and Saieed Akbari. Decomposing claw-free subcubic graphs and 4-chordal subcubic graphs. *Discret. Appl. Math.*, 296:52–55, 2021. doi:10.1016/J.DAM.2020.01.016.

2 Saieed Akbari, Tommy R. Jensen, and Mark H. Siggers. Decompositions of graphs into trees, forests, and regular subgraphs. *Discret. Math.*, 338(8):1322–1327, 2015. doi:10.1016/J.DISC.2015.02.021.

3 Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 167–180. Springer, 2009. doi:10.1007/978-3-642-02777-2_18.

4 Oliver Bachtler and Irene Heinrich. Reductions for the 3-decomposition conjecture. *Procedia Computer Science*, 223:96–103, 2023. XII Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS 2023). doi:10.1016/j.procs.2023.08.218.

5 Curtis Bright, Kevin K. H. Cheung, Brett Stevens, Ilias S. Kotsireas, and Vijay Ganesh. A sat-based resolution of Lam's problem. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3669–3676. AAAI Press, 2021. doi:10.1609/AAAI.V35I5.16483.

6 Peter J. Cameron. Research problems from the BCC22. *Discrete Mathematics*, 311(13):1074–1083, 2011. Selected Papers from the 22nd British Combinatorial Conference. doi:10.1016/j.disc.2011.02.024.

7 Michael Codish, Michael Frank, Avraham Itzhakov, and Alice Miller. Computing the Ramsey number R(4, 3, 3) using abstraction and symmetry breaking. *Constraints An Int. J.*, 21(3):375–393, 2016. doi:10.1007/S10601-016-9240-3.

8 Michael Codish, Alice Miller, Patrick Prosser, and Peter J. Stuckey. Constraints for symmetry breaking in graph representation. *Constraints*, 24(1):1–24, 2019. doi:10.1007/s10601-018-9294-5.

9 Genghua Fan and Chuixiang Zhou. Hoffmann-Ostenhof's 3-decomposition conjecture. *Discrete Mathematics*, 348(7):114454, 2025. doi:10.1016/j.disc.2025.114454.

10 Johannes K. Fichte, Markus Hecher, Daniel Le Berre, and Stefan Szeider. The silent (r)evolution of SAT. *Communications of the ACM*, 66(6):64–72, June 2023. doi:10.1145/3560469.

11 The OEIS Foundation. Number of connected cubic unlabeled graphs with n nodes. https://oeis.org/A002851.

12 Marijn Heule, Warren Hunt, Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving*, pages 269–284, Cham, 2017. Springer International Publishing.

13 Marijn J. H. Heule, Oliver Kullmann, and Armin Biere. Cube-and-conquer for satisfiability. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 31–59. Springer, 2018. doi:10.1007/978-3-319-63516-3_2.

14 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean Triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016. doi:10.1007/978-3-319-40970-2_15.

15 Arthur Hoffmann-Ostenhof, Tomáš Kaiser, and Kenta Ozeki. Decomposing planar cubic graphs. *Journal of Graph Theory*, 88(4):631–640, 2018. doi:10.1002/jgt.22234.

16 Markus Kirchweger, Tomáš Peitl, and Stefan Szeider. Co-certificate learning with SAT modulo symmetries. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 1944–1953. ijcai.org, 2023. Main Track. doi:10.24963/IJCAI.2023/216.

17 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, LIPIcs, pages 39:1–39:17. Dagstuhl, 2021. doi:10.4230/LIPIcs.CP.2021.34.

18 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation and enumeration. *ACM Trans. Comput. Log.*, 25(3):1–30, 2024. doi:10.1145/3670405.

19 Panpan Li and Wenzhong Liu. Decompositions of cubic traceable graphs. *Discuss. Math. Graph Theory*, 40(1):35–49, 2020. doi:10.7151/DMGT.2132.

20 João P. Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 131–153. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-131.

21 Ciaran McCreesh, Patrick Prosser, and James Trimble. The Glasgow subgraph solver: Using constraint programming to tackle hard subgraph isomorphism problem variants. In Fabio Gadducci and Timo Kehrer, editors, *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*, volume 12150 of *Lecture Notes in Computer Science*, pages 316–324. Springer Verlag, 2020. doi:10.1007/978-3-030-51372-6_19.

22 Bernardino Romera-Paredes et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi:10.1038/S41586-023-06924-6.

23 Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024. doi:10.1038/S41586-023-06747-5.

24 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer Verlag, 2014. doi:10.1007/978-3-319-09284-3_31.

25 Tianwei Zhang and Stefan Szeider. Searching for smallest universal graphs and tournaments with SAT. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pages 39:1–39:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CP.2023.39.

A Proofs for Section 4.2

Proof of Lemma 9.

Proof. We distinguish the following three cases. Suppose $|V(T_1) \cap V(T_2)| = 0$. Then G is clearly a forest consisting of two disconnected trees. Suppose $|V(T_1) \cap V(T_2)| > 1$, then there exists distinct $u, v \in V(T_1) \cap V(T_2)$. Since T_1 and T_2 are trees, it follows that there is a unique path between u and v in both T_1 and T_2. Given that $E(T_1) \cap E(T_2) = \emptyset$, these two paths are distinct and therefore together form a cycle. Suppose $V(T_1) \cap V(T_2) = \{s\}$. It suffices to show that for any distinct $u, v \in V(G)$, there is a unique path between u and v. Without loss of generality, assume $u \in V(T_1)$. We distinguish the following two cases.

1. $v \notin V(T_1)$. Since $\{s\}$ is separating in G, any path between u and v consists of a path between u and s in T_1 and a path between v and s in T_2. Since both of the latter exist and are unique, it follows that there is a unique path between u and v in G.
2. $v \in V(T_1)$. Then there exists a unique path between u and v in T_1. Suppose there is a different path between u and v in G. Then this path must contain at least one edge from $E(T_2)$ and therefore a vertex $w \in V(T_2)/\{s\}$. This means that this path consists of a path p_1 between u and w and a path p_2 between v and w. Since $\{s\}$ is separating in G, it follows that s is on both paths, and therefore s is visited twice on the path. Contradiction. ◀

Proof of Lemma 11.

Proof. We prove the statement from both directions.

1. Suppose $\mathcal{S}$ is mergeable and a is witnessing permutation of $[n]$. Then we can show by induction on i that $G_i := (\bigcup_{j=1}^{i} V(T_j), \bigcup_{j=1}^{i} E(T_j))$ is a tree. The base case where $i = 1$ is true by definition. The induction step is valid because of Lemma 9. This means that in particular $G = G_n$ is a tree.
2. Suppose G is a tree. We show by induction on the size of non-empty $M \subseteq [n]$ that if $G_M := (\bigcup_{i \in M} V(T_i), \bigcup_{i \in M} E(T_i))$ is a tree, then $\mathcal{S}_M := \{\, S_i : i \in M \,\}$ is mergeable. The base case where $|M| = 1$ is trivially true. Define $I_M^x := V(T_x) \cap (\bigcup_{i \in M \setminus \{x\}} V(T_i))$. For the induction step, we first show that if G_M is a tree and $|M| > 1$, then there is $x \in M$ such that $|I_M^x| = 1$. Since G_M is connected, it follows that $|I_M^x| \geq 1$ for all $x \in M$. Now assume the contrary, that is, $|I_M^x| > 1$ for all $x \in M$. We will see this assumption allows us to find a cycle in G_M, which contradicts the premise that G_M is a tree. To illustrate this, we build sequences $b_1, b_2, ... b_{n+1} \in M$ and $c_1, c_2, ..., c_{n+1}$ where $c_i \in T_{b_i}$ for all $i \in [n+1]$. Start with any $b_1 \in M$, and let $c_1 \in I_M^{b_1}$. Given b_i and c_i, pick $b_{i+1} \neq b_i$ such that $c_i \in T_{b_{i+1}]}$ and $c_{i+1} \in I_M^{b_{i+1}} \setminus \{c_i\}$. According to the pigeonhole principle, there exists $x, y \in [n+1]$, $x < y$ such that $b_x = b_y$. Now we can obtain a cycle in G_M by concatenating the unique path from c_x to c_{x+1} in $T_{b_{x+1}}$, the unique path from c_{x+1} to c_{x+2} in $T_{b_{x+2}}$,..., the unique path from c_{y-2} to c_{y-1} in $T_{b_{y-1}}$ and the unique path from c_{y-1} to c_x in $T_{b_y} = T_{b_x}$. So far, we have established that if G_M is a tree and $|M| > 1$, then there is $x \in M$ such that $|I_M^x| = 1$. This means that $G_{M \setminus \{x\}}$ is a tree. By the induction hypothesis, there is a mapping $a : [|M| - 1] \to M \setminus \{x\}$ that witness the fact that $\mathcal{S}_{M \setminus \{x\}}$ is mergable. Now it is easy to see that $a' : [|M|] \to M$ defined as $a'(i) := a(i)$ for all $1 \leq i < |M|$ and $a'(|M|) := x$ witnesses the fact that $\mathcal{S}_M$ is mergable. Now we have established with induction that if $G_M := (\bigcup_{i \in M} V(T_i), \bigcup_{i \in M} E(T_i))$ is a tree, then $\mathcal{S}_M := \{\, S_i : i \in M \,\}$ is mergeable. In particular, if G is a tree, then $\mathcal{S}$ is mergeable.

◀

Proof of Theorem 13.

Proof. Suppose X is a subgraph of a connected cubic graph G with a 3-decomposition c_G. Then $c := c_G|_{E(X)}$ is a possible behavior of X. Note that the green edges of $c_G|_{E(G/X)}$ form a spanning forest in G/X. Let P be the partition of $O(X)$ that the green edges of c_G in G/X induce. Since c_G is a 3-decomposition, it follows that $P \in \text{Comp}(P_c)$. Now, suppose c' is a possible behavior of Y that satisfies the conditions above. It suffices to show that

$$d : E(G[X \to Y]) \to \text{RGB}, \qquad d := \begin{cases} c'(e) & \text{if } e \in E(Y); \\ c_G(e) & \text{otherwise.} \end{cases}$$

is a 3-decomposition of $G[X \to Y]$, i.e., it satisfies the conditions in Proposition 6. The first condition is satisfied due to Condition 1 of Corollary 7 and Condition 1 of Theorem 13. The second condition is satisfied because $P'_c \in \text{Comp}(P)$. ◀