Breaking Symmetries in Quantified Graph Search: A Comparative Study

Mikoláš Janota[1], **Markus Kirchweger**[2], **Tomáš Peitl**[2], **Stefan Szeider**[2]

[1] Czech Technical University in Prague
[2] Algorithms and Complexity Group, TU Wien, Austria
mikolas.janota@gmail.com, mk@ac.tuwien.ac.at, peitl@ac.tuwein.ac.at, sz@ac.tuwien.ac.at

Abstract

Graph generation and enumeration problems often require handling equivalent graphs—those that differ only in vertex labeling. We study how to extend SAT Modulo Symmetries (SMS), a framework for eliminating such redundant graphs, to handle more complex constraints. While SMS was originally designed for constraints in propositional logic (in NP), we now extend it to handle quantified Boolean formulas (QBF), allowing for more expressive specifications like non-3-colorability (a coNP-complete property). We develop two approaches: a static QBF encoding and a dynamic method integrating SMS into QBF solvers. Our analysis reveals that while specialized approaches can be faster, QBF-based methods offer easier implementation and formal verification capabilities.

1 Introduction

Generating and enumerating graphs can become inefficient when the same graph structure appears multiple times with different vertex labels. A graph's isomorphism class contains all such equivalent labelings, of which we can designate one as canonical—typically the one with lexicographically smallest adjacency matrix. SAT Modulo Symmetries (SMS; Kirchweger and Szeider 2024) builds on this idea to eliminate redundant graph generation. It extends a conflict-driven (CDCL) SAT solver with a custom propagator that enforces canonicity: when a partial graph assignment cannot be extended to a canonical solution, the solver backtracks. While static symmetry breaking through additional constraints is also possible, no polynomial-size encoding is known for complete symmetry breaking of general graphs.

The original SMS framework handled constraints expressible in propositional logic. However, many interesting graph properties require quantified constraints. For example, proving a graph is not 3-colorable means showing that no assignment of three colors to vertices avoids adjacent vertices sharing a color - a coNP-complete property. To handle such properties, SMS was extended with co-certificate learning (CCL Kirchweger, Peitl, and Szeider 2023a). CCL uses two solvers: one generates candidate graphs, while another tries to refute them by finding certificates like valid colorings. When a certificate is found, CCL strengthens the constraints for future candidate generation.

While effective, CCL requires custom implementations for each new graph property - both the refutation algorithm and the constraint strengthening mechanism must be built from scratch. This makes CCL tedious to implement and error-prone for complex properties. Quantified Boolean formulas (QBFs) offer a more systematic approach. A QBF solver can automatically handle the interplay between graph generation and certificate checking based on a formal problem specification. This brings three key benefits: simpler implementation through standard encodings, formal verification through proof generation, and automatic incorporation of future advances in QBF solving technology.

In this paper, we develop this QBF-based approach into a complete framework for quantified graph search problems. Our contributions are:

1. a complete dynamic symmetry-breaking method Q-SMS that extends existing QBF solvers with SMS-style pruning;
2. a complete static symmetry-breaking method Q-static using universal variables to encode graph canonicity;
3. implementations of several quantified graph search problems demonstrating the framework's applicability.

Table 1 shows the main attributes of the three paradigms for quantified graph search. In Section 3, we explain the implementation details for both Q-static, and Q-SMS, describe the QBF solvers we modified and the production of formal proofs, and give an overview of CCL.

For benchmarking, we selected several classes of quantified graph search problems composed of applications that appeared in earlier SMS papers and other prominent problems in graph theory. We describe them in Section 4.

In Section 5, we discuss the results of our evaluation. We conclude that for simpler problems, CCL is usually the fastest, but it cannot provide formal proofs and is generally more error-prone than the other methods. Q-static is the simplest to implement but scales poorly. Q-SMS strikes the best balance overall, providing good performance, formally verifiable proofs, and the expressiveness of general QBF.

Among the QBF solvers themselves, our results show that our newly implemented 2-QBF solver, 2Qiss, outperforms the other QBF solvers. This is a surprise, as our primary

	CCL	Q-static	Q-SMS
	(Kirchw. et al. 2023a)	this paper	
speed	✓		✓
proofs		✓	✓
ease-of-use		✓	✓

Table 1: Pros and cons of the approaches studied in this paper. CCL is typically the fastest but hard to implement for complicated problems. Q-static is straightforward to implement (one encoding can be reused) and can be used with any QBF solver, but turns out to be significantly slow. Encodings for Q-SMS are just as easy as with Q-static, but run much faster; the only price to pay is the need for a specialized SMS-endowed solver (such as those presented in this paper). Both Q-static and Q-SMS can produce independently verifiable proofs, which, combined with uniform (and hence less error-prone) problem encodings, provide stronger trust than the slightly faster CCL.

motivation for putting SMS into QBF solvers was to obtain a performance boost from using established solvers.

2 Preliminaries

For a positive integer n, we write $[n] = \{1, 2, \dots, n\}$. We assume familiarity with fundamental notions of propositional logic (Prestwich 2021). Below, we review some basic notions from graph theory.

Graphs. All considered graphs are undirected and simple (i.e., without parallel edges or self-loops). A *graph* G consists of 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. The *order* of a graph G is the number of its vertices, $|V(G)|$. We write $\mathcal{G}_n$ to denote the class of all graphs with $V(G) = [n]$. The *adjacency matrix* of a graph $G \in \mathcal{G}_n$, denoted by A_G, is the $n \times n$ matrix where the element at row v and column u, denoted by $A_G(v, u)$, is 1 if $vu \in E$ and 0 otherwise.

Coloring. A *proper k-coloring* of a graph G is a mapping $c : V(G) \to [k]$ such that $uv \in E(G)$ implies $c(u) \neq c(v)$. The *chromatic number* of a graph G is the smallest integer k, for which a proper k-coloring exists. A *k-edge-coloring* of a graph G is a mapping $c : E(G) \to [k]$. A k-edge-coloring of a graph G is *proper* if incident edges have different colors.

Isomorphisms. For a permutation $\pi : [n] \to [n]$, $\pi(G)$ denotes the graph obtained from $G \in \mathcal{G}_n$ by the permutation π, where $V(\pi(G)) = V(G) = [n]$ and $E(\pi(G)) = \{\, \pi(u)\pi(v) \mid uv \in E(G) \,\}$. Two graphs $G_1, G_2 \in \mathcal{G}_n$ are *isomorphic* if there is a permutation $\pi : [n] \to [n]$ such that $\pi(G_1) = G_2$; in this case G_2 is an *isomorphic copy* of G_1.

Partially defined graphs. As defined by Kirchweger and Szeider (2021), a *partially defined graph* is a graph G where $E(G)$ is split into two disjoint sets $D(G)$ and $U(G)$. $D(G)$ contains the *defined* edges, $U(G)$ contains the *undefined* edges. A (*fully defined*) graph is a partially defined graph G with $U(G) = \emptyset$. A partially defined graph G can be *extended* to a graph H if $D(G) \subseteq E(H) \subseteq D(G) \cup U(G)$.

SAT Modulo Symmetries (SMS). SMS is a framework that augments a CDCL SAT solver (Fichte et al. 2023; Marques-Silva, Lynce, and Malik 2021) with a custom propagator that can reason about symmetries, allowing to search modulo isomorphisms for graphs in $\mathcal{G}_n$ which satisfy constraints described by a propositional formula. During search the SMS propagator can trigger additional conflicts on top of ordinary CDCL and consequently learn *symmetry-breaking clauses*, which exclude isomorphic copies of graphs. More precisely, only those copies are kept which are lexicographically minimal (*canonical*) when considering the rows of the adjacency matrix concatenated into a single vector. A key component is a minimality check, which decides whether a partially defined graph can be extended to a minimal graph; if it cannot, a corresponding clause is learned. For a full description of SMS, we refer to the original work where the framework was introduced (Kirchweger and Szeider 2021, 2024). SMS has been successfully applied to a wide range of combinatorial problems (Fazekas et al. 2023; Kirchweger, Scheucher, and Szeider 2022; Kirchweger, Peitl, and Szeider 2023a,b; Kirchweger, Scheucher, and Szeider 2023; Zhang, Peitl, and Szeider 2024; Zhang and Szeider 2023).

Quantified Boolean Formulas

Quantified Boolean formulas (QBF) generalize propositional logic with quantification. We consider *closed* formulas in *prenex* form, i.e., ones where all quantifiers are in the front in the quantifier *prefix*, and the rest—the *matrix*—is a propositional formula. An example of a QBF is

$$\exists x \exists y \forall z \big((x \wedge \neg y) \vee z \big).$$

Given a formula ϕ and an assignment α of some variables, then $\phi[\alpha]$ is the formula resulting by replacing x with $\top$ if $\alpha(x) = 1$ and replacing it with $\bot$ if $\alpha(x) = 0$. The semantics of a QBF can be defined recursively. The formula $\exists x \phi$ is true if $\phi[\{x \mapsto 0\}] \vee \phi[\{x \mapsto 1\}]$ is true. The formula $\forall x \phi$ is true if $\phi[\{x \mapsto 0\}] \wedge \phi[\{x \mapsto 1\}]$ is true.

Let $X = \{x_1, \dots, x_n\}$. If $Q \in \{\exists, \forall\}$, we write $QX\phi$ for $Qx_1 \dots Qx_n \phi$. Further, $\alpha|_X$ gives the restriction of the function only to elements in X. A simple CEGAR (counterexample-guided abstraction refinement) algorithm for 2-QBF of the form $\exists X \forall Y \phi$ is given in Algorithm 1. The SAT solver used as a subroutine returns a model of the propositional formula if satisfiable, otherwise NULL. The idea behind Algorithm 1 is the basis of many QBF solvers (Janota et al. 2012; Rabe and Tentrup 2015; Hecking-Harbusch and Tentrup 2018; Janota 2018a,b).

Algorithm 1 can be seen as a game. One player tries to assign existential variables to satisfy the matrix ϕ; the other tries to find universal counter-moves to falsify ϕ. In response to a successful counter-move β, the first player must find a move α that does not lose to β (or any other previous move).

3 Combining SMS with QBF

In this section we expound the approaches listed in Table 1. We start with the idea of checking the canonical form with

Algorithm 1: A CEGAR-based 2-QBF solver

Input: A closed 2-QBF $\Phi = \exists X \forall Y \phi$.
Output: True if Φ is true, false otherwise.

```
1: φ' ← ⊤
2: loop forever
3:    α ← SAT(φ')                          {Get a model of φ'}
4:    if α = NULL then
5:       return False                      {No model}
6:    β ← SAT(¬φ[α|_X])                    {Compute a counterexample}
7:    if β = NULL then
8:       return True
9:    φ' ← φ' ∧ φ[β|_Y]                    {Refine}
```

universal variables (Q-static), and move on to the integration of SMS in existing QBF solvers.

Quantified Static Symmetry Breaking (Q-static)

While it is unknown whether a polynomially sized complete symmetry break is expressible in propositional logic, it is easy to achieve in QBF. The idea is to encode that no permutation of the vertices results in a lexicographically smaller adjacency matrix for the graph, using universal variables to represent the permutations.

We first construct a formula expressing that a permutation leads to a lexicographically smaller graph for a fixed number of vertices n. Negating the formula leads to the encoding of minimality. Since the adjacency matrix is symmetric, it is sufficient to consider only the upper triangle. We write $P_n := \{ (i,j) \mid i,j \in [n], i < j \}$, and use the variables $e_{i,j}$ for $(i,j) \in P_n$ to encode the adjacency matrix and $p_{i,j}$ for $i,j \in [n]$ to encode the permutation $\pi : [n] \to [n]$, i.e., $p_{i,j}$ is true if and only if $\pi(i) = j$. We use the following formula to ensure that the variables $p_{i,j}$ indeed represent a permutation, i.e., that π is total and injective:

$$isPerm = \bigwedge_{i \in [n]} \bigvee_{j \in [n]} p_{i,j} \wedge \bigwedge_{i<j} \bigwedge_{k \in [n]} (\neg p_{i,k} \vee \neg p_{j,k}).$$

Finally, we define variables $pe_{i,j}$ to hold the permuted adjacency matrix: $pe_{i,j}$ is true if, and only if $A_{\pi(G)}(i,j) = 1$.

$$pe_{i,j} = \bigvee_{(i',j') \in P_n} (e_{i',j'} \wedge p_{i',i} \wedge p_{j',j}).$$

This allows us to construct a formula that is true if the permutation leads to a lexicographically smaller graph:

$$nonMin = \bigvee_{(i,j) \in P_n} \bigwedge_{\substack{(i',j') \in P_n, \\ (i',j') \prec_{lex} (i,j)}} (e_{i',j'} \vee \neg pe_{i',j'}) \wedge e_{i,j} \wedge \neg pe_{i,j},$$

where $\prec_{lex}$ is the lexicographic order on vertex pairs, i.e., $(i,j) \prec_{lex} (i',j')$ if (i) $i < i'$ or (ii) $i = i'$ and $j < j'$.

If the formula $nonMin \wedge isPerm$ is satisfied, then there is a permutation π described by the assignment to $p_{i,j}$ and $(i^*, j^*) \in P_n$ such that $A_G(i',j') \geq A_{\pi(G)}(i',j')$ for all $(i',j') \prec_{lex} (i^*,j^*)$ and $A_G(i^*,j^*) > A_{\pi(G)}(i^*,j^*)$. This implies that G is not minimal, and consequently the formula

$$\forall \{ p_{i,j} \mid i,j \in [n] \} \quad \neg(nonMin \wedge isPerm)$$

encodes lexicographic minimality.

As we will see in the experiments, trying to enforce lexicographic minimality using a QBF encoding scales poorly in terms of performance. This does not come as a big surprise for CEGAR-based approaches using a second SAT solver to check whether the universal property is satisfied. The reason is that there is a "hidden" pigeon-hole principle. This is easiest explained in an example. Let $G \in \mathcal{G}_n$ and let the first vertex have degree δ with $(1,i) \in E(G)$ for $i \in [n] \setminus [n-\delta]$. Let another vertex u have degree $\delta + 1$. When mapping the vertex u to the first vertex, the solver tries to assign the $\delta + 1$ neighbors of u to vertices in $[n] \setminus [n-\delta]$. In other words, one tries to injectively map $\delta + 1$ elements to δ elements. SAT solvers are known not to perform well on these types of instances.

Independently of the chosen ordering, a further disadvantage of the minimality encoding is that, for CEGAR-based approaches, the minimality is only checked when all existential variables are assigned, i.e., the graph is fully defined. One can often detect that a partially defined graph with only a few edge variables assigned already cannot be extended to a canonical one, and thus branches of the search tree can be pruned much earlier. An extreme case is provided by instances where the existential part is unsatisfiable: the static symmetry-breaking constraint is not evaluated at all.

QBF Modulo Symmetries (Q-SMS)

In contrast to encoding the SMS minimality check into QBF, as described above, we describe here how to integrate SMS with an external minimality check (similarly to the SAT-based SMS) into three different circuit-based QBF solvers, Qfun (Janota 2018b), CQesto (Janota 2018a) and Qute (Peitl, Slivovsky, and Szeider 2019). The main reason for only considering circuit-based solvers is to avoid an additional quantifier alternation resulting from transforming quantified circuits into a QCNF.

Qfun and CQesto are both based on CEGAR. Unlike its precursor, the RAReQS algorithm (Janota and Silva 2011; Janota et al. 2012, 2016), **Qfun** learns functions using decision trees. Instead of adding $\phi[\beta|_Y]$ to the formula in Algorithm 1 line 9, the universal variables are occasionally replaced by formulas/functions only depending on existential variables. For example, a universal variable could be substituted by the negation of an existential variable but also with more complex functions. The functions are constructed based on previous models and counter models.

The **CQesto** algorithm has been designed to be more lightweight than RAReQS, which scales poorly beyond 3 levels. Consider the following example. Let $g \equiv (x \vee y)$ in $\exists x \exists y \forall u \, (g \vee u) \wedge (\neg g \vee \neg u)$ and the assignment $x = y = 1$ to which the universal player responds with $u = 1$, falsifying the second conjunct. Now, the solver could add the constraint $\neg x \vee \neg y$ because setting both x and y to true is losing for the existential player. However, this is a weak constraint because the same move will beat any assignment that satisfies gate g. Instead, CQesto *propagates* the assignment and learns the constraint $\neg g$. For the purpose of this paper, the solver is run repeatedly to obtain many different solutions, and we observed that value propagation incurs a non-

negligible time overhead. Therefore, we have changed its implementation to be bottom-up, instead of top-down, and to evaluate gates lazily. This has improved the performance slightly, but propagation still takes considerable time.

Note that the solvers are not restricted to 2-QBF, but for the sake of simplicity, we restricted the presentation to this simpler case. For the generalization to an arbitrary number of quantifier alternations, we refer to the literature.

From a conceptual standpoint, it is straightforward to incorporate SMS into the two CEGAR-based solvers Qfun and CQesto. Using an SMS solver for calls in Line 3 ensures that non-canonical graphs are excluded. For all other SAT calls, a standard SAT solver is used.

For graph problems, often not only the existence of a graph with a certain property is of interest but also to enumerate all graphs up to isomorphism with the wished-for property. We also extend the solvers to allow enumerating solutions. We propose to use free variables to indicate which variables are relevant for enumeration, i.e., only the assignments of the free variables are part of the output. Free variables are already part of the QCIR input format (Jordan, Klieber, and Seidl 2016).

Qute is a solver based on Quantified Conflict-Driven Constraint Learning (QCDCL; Zhang and Malik 2002). As such, Qute does not use SAT solvers, so we cannot simply replace one component. It is reasonably straightforward to call the minimality check at the appropriate place in the QCDCL loop (when unit propagation reaches a fixed point). However, the learning of symmetry-breaking clauses poses some challenges that we try to explain briefly.

QCDCL, as implemented in Qute, maintains, in addition to the usual set of input and learned clauses like in CDCL, another set of *cubes*. A cube is a conjunction of literals. The cube set maintained by Qute is a DNF (disjunctive normal form, i.e., a disjunction of cubes) representation of the input formula, together with further learned cubes (just like the clause set is the original formula plus learned clauses).

Conjoining a (symmetry-breaking or solution-blocking) clause to the clause set is trivial and fast; one just appends it. However, conjoining a clause to the cube set, by De Morgan's laws, potentially changes every cube, including learned ones, and that is expensive when done repeatedly. We can do better if we understand the structure of the cubes.

Suppose we are solving a QBF whose matrix is the circuit F. Qute initializes its cube set with DNF(F), which is obtained with Tseitin's well-known translation procedure (Tseitin 1968).[1] DNF(F) consists of two components, DNF$^*(F)$, which encodes the structure of the circuit as a DNF, and the unit cube (F_{out}), which says that the circuit should evaluate to true. Now, suppose we already have DNF(F) = DNF$^*(F) \vee (F_{out})$, and we want to obtain DNF($F \wedge C$) for a newly added clause C. Observe that much of the structure of the circuits F and $F \wedge C$ is identical. It is not too hard to see that in fact DNF($F \wedge C$) $\equiv$ DNF$^*(F) \vee$ DNF$^*(C) \vee (F_{out} \wedge C_{out})$. Thus, we can conjoin the clause C by encoding it into DNF, appending to the cube set, and replacing the unit cube (F_{out}) with the new output cube $(F_{out} \wedge C_{out})$.

After this replacement, every cube derived using (F_{out}) might be invalid. Such cubes we call *tainted*, and we extended Qute to keep track of whether a cube is tainted. After adding a clause, all tainted cubes must be removed.

Three technical aspects of clause addition appear to be important. First, delete tainted cubes lazily: mark them for deletion, but do not actually clean them from memory (a periodical cleanup is performed by the solver). Second, when a cube shares a literal with the newly added clause, the cube can be kept even if it is tainted (a formal proof of this is an easy exercise; we leave it out to save space). Third, to add a clause, the solver must backtrack to a consistent state. We backtrack all the way; a finer yet complicated analysis might allow keeping some of the search state.

Proofs

We have implemented basic proof logging with SMS in Qute. We picked Qute for this as it is the only solver that supports proof logging. Our proof framework certifies correctness of the obtained solutions (that the graphs have the right properties), and the fact that all solutions have been found (unsatisfiability at the end). To certify correctness of the symmetry-breaking clauses generated by SMS, we can use the same approach as for propositional SMS (Kirchweger and Szeider 2024), as the validity of the symmetry clauses does not depend on the encoding.

Qute outputs proofs in *long-distance Q-resolution* (LDQ-resolution, for short; Zhang and Malik 2002; Balabanov and Jiang 2012), a clausal proof system (and its counterpart LDQ-consensus which operates dually on cubes). LDQ-resolution and consensus can prove a QBF in CNF/DNF false and true, respectively. When solving a QBF with the circuit matrix F, the proof starts from CNF(F)/DNF(F). An extra step outside of the proof system, which we have not implemented, would be necessary to certify the correctness of the circuit-to-CNF/DNF translation.

A single LDQ-resolution/consensus proof can certify one solution or prove that there are no solutions. In order to certify an enumeration problem, several proofs are needed. All the necessary proofs can be extracted from the solver trace (the chronological log of all learned clauses and cubes). We have adapted the extractor `qrp2rup` (Peitl, Slivovsky, and Szeider 2018) to handle traces containing multiple proofs.

In a proof trace, all variables and axioms are usually introduced right at the beginning. In SMS, axioms and auxiliary variables can be introduced on the fly as well. We adapted `qrp2rup` to handle such cases. When new axioms are introduced in the middle of the proof, any tainted cubes must be forgotten (see above). We do not see how the proof checker could check this without understanding the entire setup, including the circuit structure outside of the clausal proofs. It appears to us that a dedicated proof system for enumeration would be necessary to correctly capture what happens when a solution is found and blocked for subsequent search. We leave the design of such a proof system to future work.

[1] The translation into DNF (Disjunctive Normal Form) is the negation of the better known translation into CNF (Conjunctive Normal Form), and uses universal auxiliary variables.

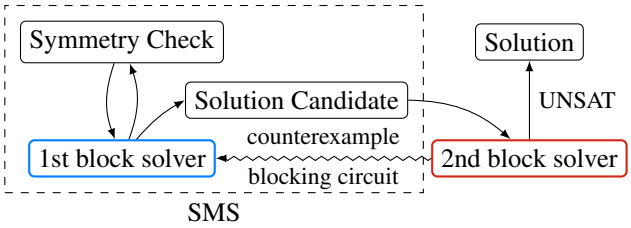

Figure 1: CCL. CEGAR-based QBF solving with an SMS blackbox follows a similar pattern.

On the other hand, `qrp2rup` can, on top of verifying a proof, extract strategies for the winning player and DRAT proofs for them, and verify them with DRAT-trim (Wetzler, Heule, and Hunt 2014), and can even extract GRAT annotations directly and verify the strategies with the formally verified checker `gratchk` (Lammich 2017). We made sure this feature continues to work even with multiple proofs.

A New 2-QBF Solver

Since in preliminary experiments, existing QBF solvers (extended with SMS) performed worse than CCL, we decided to write our own 2-QBF solver 2Qiss, which implements Algorithm 1. We use CaDiCaL (Biere et al. 2020) as the underlying SAT solver. We create two instances of the solver, one for calls at Line 3, the other for calls at Line 6. The first solver is responsible for the existential part, the second for the universal. The first solver is initialized with the formula $\phi\,[\{\, y \mapsto 0 \mid y \in Y \,\}]$ and the second with $\neg\phi$. For the first solver, we use the incremental interface to add additional constraints at Line 9 (incremental calls preserve learned clauses). For the second, we use assumptions to fix the assignment to the existential variables in each call.

Like the other QBF solvers mentioned above, we use QCIR as input format, i.e., the matrix of the formula is not restricted to CNF. Nevertheless, constraints must be transformed into CNF before they are added to the underlying SAT solvers. This is done using the Tseitin transformation (Tseitin 1968). New variables are introduced for subformulas and constraints added to ensure that the variables are equivalent to the subformulas. To avoid encoding identical subformulas several times, we use a technique called *gate/structural hashing* (Balabanov et al. 2016). For each solver, a hash map is created which maps each subformula to its corresponding variable.

Before being passed to a solver, each formula is simplified by removing true (false) inputs from conjunctions (disjunctions), replacing empty conjunctions (disjunctions) with true (false), and replacing unary gates by their only input.

Many natural 2-QBF problems can be encoded in the form $\phi = \exists X \forall Y\,(F(X) \wedge G(X,Y))$, i.e., with a top-level conjunct that uses only existential variables. After adding $\phi\,[\{\, y \mapsto 0 \mid y \in Y \,\}]$ to the first SAT solver, we delete the existential part $F(X)$ from the formula (if there are more such existential conjuncts, we remove all of them).

Co-certificate learning (CCL)

CCL follows the pattern of Algorithm 1, depicted graphically in Figure 1. The 2nd block solver can be any algorithm: a SAT solver (with a suitable formula), or even a custom domain-specific solver. Whereas a CEGAR QBF solver computes a strengthening on Line 9 from the input QBF and a counter-move β, in CCL, the 2nd-block solver is responsible for returning an appropriate strengthening. In some cases, like graph coloring, this is easy: when the 2nd solver finds a coloring of the candidate graph proposed by the 1st solver, it can return a clause that says at least one edge with endpoints of the same color should be present in future candidate graphs. For more involved problems, though, CCL can get complex and error-prone. Since the 2nd-block solver can be an arbitrary algorithm, it is also hard to provide independently verifiable proofs for CCL.

4 Benchmark Problems and Encoding

In this section we introduce the graph search problems and present QBF encodings on which we evaluated our solvers. For most problems we only sketch the encoding, as the main focus is on comparing the solving approaches. See Section 5 for a link to generator scripts and details of the formulas. All encodings are of the form $\exists X \forall Y (F(X) \wedge \neg H(X,Y))$. We call F the $\exists$-encoding and H the $\forall$-encoding.

Coloring Triangle-Free Graphs

If a graph contains the k-clique as a subgraph, then its chromatic number must be at least k. The opposite is not true: Mycielski (1955) explicitly constructed triangle-free graphs (without K_3 as a subgraph) with unbounded chromatic number. Erdős (1967) asked about the values $f(k)$, which denote the smallest number of vertices in a triangle-free non-$(k-1)$-colorable graph. Mycielski's construction provides upper bounds on $f(k)$, and these are tight up to $k = 4$; for $k = 5$, minimal graphs are also known (Goedgebeur 2020), but none of them is a *Mycielskian*. The cases $k \geq 6$ are open.

Task: Compute a triangle-free graph with n vertices and chromatic number at least k.

$\exists$-Encoding: The existential part ensures that the graph is triangle-free. Without loss of generality, we further restrict the search to *maximal triangle-free* graphs (triangle-free and adding any edge creates a triangle).

$\forall$-Encoding: Universal variables $c_{v,i}, v \in [n], i \in [k-1]$ and straightforward constraints enforce non-$(k-1)$-colorability.

Folkman Graphs

Folkman graphs, named after the mathematician Folkman (and not to be confused with the specific graph also named after him), play an important role in generalized Ramsey theory (Folkman 1970). The *Ramsey number* $R(x,y)$ is the least integer such that for every 2-edge-coloring (red-blue) of the complete graph $K_{R(x,y)}$, one can find a red K_x or a blue K_y subgraph (Ramsey 1930; Radziszowski 2021).

Folkman numbers generalize this idea, by looking for the existence of monochromatic complete subgraphs in edge colorings of arbitrary graphs. The *Folkman number* $F(x, y; k)$ is the least integer such that there exists an $(x, y; k)$-*Folkman graph*: a K_k-free graph such that any 2-edge-coloring contains either a red K_x or a blue K_y.

Task: Given two integers k and n, output a $(3, 3; k)$-Folkman graph with n vertices.

$\exists$**-Encoding:** K_k-free: enumerate all k-tuples of vertices, and in each require at least one edge to be missing.

$\forall$**-Encoding:** We use universal variables for encoding edge-colorings, and require that for each edge coloring there be some monochromatic triangle.

We focus on the Folkman number $F(3, 3; 4)$, for which the best bounds are $21 \leq F(3, 3; 4) \leq 786$ (Bikov and Nenov 2020; Lange, Radziszowski, and Xu 2012).

Domination Number of Cubic Graphs

A *dominating set* of a graph G is a subset $S \subseteq V(G)$ such that each vertex is in S or has a neighbor in S. The *domination number* $\gamma(G)$ is the size of a smallest dominating set of G. Reed (1996) conjectured that the domination number of every *cubic* (each vertex has degree 3) connected graph G is $\leq \lceil |V(G)|/3 \rceil$. This conjecture turned out to be false (Kostochka and Stodolsky 2005; Kelmans 2006), but restricted variants remain open. A graph is k-*connected* if it cannot be made disconnected by removing fewer than k vertices. The *girth* of a graph is the length of its shortest cycle.

Conjecture 1. *Let G be a cubic graph. If*

i) G is 3-connected, or
ii) G is bipartite, or
iii) G has girth $g \geq 6$,

then $\gamma(G) \leq \lceil |V(G)|/3 \rceil$.

Task: Find a counterexample to Conjecture 1 with n vertices.

$\exists$**-Encoding:** We use sequential counters (Sinz 2005) to enforce cubicity. Instead of a full encoding of 3-connectedness, we only require connectedness, and postprocess the solutions. For girth $g \geq 6$ we use a compact encoding girth_6^n due to Kirchweger and Szeider (2024). To encode bipartiteness, we introduce variables $b_i, i \in [n]$ to guess the bipartition.

$\forall$**-Encoding:** Checking whether a graph does not have a dominating set of size $\leq k$ is coNP-complete. Universal variables d_i, $i \in [n]$ and straightforward constraints describe a dominating set of size $\leq k$.

Löwenstein and Rautenbach (2008) proved Reed's conjecture for graphs of girth ≥ 83; Verstraëte (Dorbec and Henning 2024) conjectured it holds for girth ≥ 6 (condition *iii*).

Treewidth

Treewidth is a well-studied graph invariant that measures how much a graph resembles a tree. The standard definition of treewidth uses a *tree decomposition* (Bodlaender 1993); here we recall the equivalent definition in terms of *elimination orderings*, which, for a graph G, is a permutation $\pi = v_1, \ldots, v_n$ of $V(G)$. The *width* $w_G(\pi)$ of π for G is $\max(\deg_G(v_1), w_{G^*_{v_1}}(v_2, \ldots, v_n))$, where $G^*_{v_1}$ is obtained from G by removing v_1 and completing its neighbors to a clique. *Treewidth* $\text{tw}(G)$ is the minimum width of an elimination ordering. Checking if $tw(G) \leq k$ is NP-complete.

It is well known that if G contains a graph H as minor (H is obtained from G by deleting vertices and edges and by contracting edges), then $\text{tw}(H) \leq \text{tw}(G)$, i.e., the class of graphs of treewidth $\leq k$ is *minor-closed*. A famous theorem of Robertson and Seymour (2004) states that every minor-closed family of graphs is definable by a finite set of forbidden minors, but since the proof is not constructive, finding these finite *obstruction sets* is an open challenge. The minimal obstruction set for the class of graphs of treewidth $\leq k$ consists of *treewidth-critical* graphs; whose any minor has strictly smaller treewidth.

Task: Find all treewidth-k-critical graphs with n vertices.

$\exists$**-Encoding:** For showing that a graph's treewidth is at most k, SAT encodings based on elimination orderings are known (Samer and Veith 2009).

$\forall$**-Encoding:** Negating the encoding by Samer and Veith allows us to express that the treewidth must be at least k.

In principle, one can encode treewidth-criticality by checking that any edge deletion or contraction results in a graph with treewidth $< k$. We opted for a more compact encoding, only requiring that the graph itself has treewidth $\leq k$, and postprocessing to filter out non-critical graphs.

Snarks

A *snark* is a non-3-edge-colorable cubic graph (Gardner (1976) took the name, a portmanteau of 'snake' and 'shark,' from Lewis Carroll's poem *The Hunting of the Snark*). Similarly to vertex coloring, edge coloring is NP-hard. To avoid trivial cases, it is usually required that a snark have girth ≥ 5 and be *cyclically* 4*-edge-connected* (deleting any 3 edges does not create two connected components both containing a cycle).

Task: Enumerate all snarks with n vertices.

$\exists$**-Encoding:** We use sequential counters to enforce cubicity. We forbid 3 and 4-cycles by explicitly enumerating them and requiring at least one edge from each to be absent. Instead of enforcing cyclical 4-edge-connectedness, we only enforce at least 2(-vertex)-connectedness.

$\forall$**-Encoding:** Universal variables c^l_{ij} for $i < j \in [n]$, $l \in [3]$, and straightforward constraints describe the 3-edge-coloring and ensure it is not proper.

Snarks were introduced in the context of a conjecture now known to be true as the four color theorem: that every planar graph is 4-vertex-colorable. One equivalent statement of the four color theorem is that planar snarks do not exist. Snarks continue to be relevant today; for many important conjectures in graph theory (such as the famous Cycle Double Cover Conjecture; Szekeres 1973; Seymour 1979; Jaeger 1985) it is known that the smallest counterexamples, if they

exist, must be snarks. There is already a wealth of work on enumerating small snarks; for example, all snarks with up to 36 vertices are known (Brinkmann and Goedgebeur 2017).

Kochen-Specker Graphs

Kochen-Specker (KS) vector systems are special sets of vectors in at least 3-dimensional space that form the basis of the Bell-Kochen-Specker Theorem, demonstrating quantum mechanics' conflict with classical models due to contextuality (Budroni et al. 2022). Kochen and Specker (1967) originally came up with a 3D KS vector system of size 117. The smallest known system (in 3D) has 31 vectors (Peres 1991), while the best lower bound is 24 (Kirchweger, Peitl, and Szeider 2023a; Li, Bright, and Ganesh 2023). These lower bounds were obtained with computer search for KS candidate graphs, which are *non-010-colorable* graphs with additional restrictions. A graph is 010-colorable if its vertices can be colored red and blue such that no two adjacent vertices are both red and no triangle is all blue.

Task: Enumerate all KS candidates with n vertices.

$\exists$**-Encoding:** For the full list of constraints and the encoding, see (Kirchweger, Peitl, and Szeider 2023a).

$\forall$**-Encoding:** Universal variables c_v, $v \in [n]$, and straightforward constraints enforce non-010-colorability.

5 Results

We evaluated all solvers on all benchmark problems, on a cluster of machines with different processors[2], running Ubuntu 18.04 on Linux 4.15.[3]

Table 2 shows the performance of the three main approaches on problem instances from Section 4, with a time limit of 4 hours for each instance. All solvers are run with a single thread. Table 3 shows the number of solved instances by each solver. 'minqbf' is the static encoding of lexicographic minimality, 'minqbf-co' is colexicographic minimality using 2Qiss as the underlying solver.

Note that our experimental setup is intended to compare different approaches, rather to improve on any of the mentioned problems. Making progress would likely require significantly more CPU time and extensive parallelization.

The experiments show that the QBF encoding of minimality performs poorly on almost all instances, independently of the chosen ordering. Also, Qute does not seem to be well-suited for graph problems. Although CCL, CQesto, Qfun, and 2Qiss are conceptually similar, there is variability in performance. Among the QBF solvers, 2Qiss performs best, but CCL, while solving the same instances of applicable problems, solves them 2–3 times faster. For the largest solved n from Table 2, CCL needed 6, 63, and 34 minutes, respectively, while 2Qiss needed 18, 122, and 58 minutes for the same instances. For hard combinatorial graph problems requiring multiple CPU years, using CCL might be advisable.

[2]Intel Xeon {E5540, E5649, E5-2630 v2, E5-2640 v4}@ at most 2.60 GHz, AMD EPYC 7402@2.80GHz

[3]Solvers and benchmark generators are included in supplementary material in the versions used in this paper (Janota et al. 2025). For detailed instructions on how to create the encodings and use the solvers, as well as for up-to-date version of solvers, visit https://sat-modulo-symmetries.readthedocs.io/en/latest/applications#qbf.

Task	CCL	Q-static	Q-SMS
Δ-free non-3-colorable	15	13	15
Max-Δ-free non-4-colorable	20	12	20
Kochen-Specker	20	-	20
Domination conjecture (*i*)	-	12	28
Domination conjecture (*ii*)	-	12	34
Domination conjecture (*iii*)	-	12	30
Folkman $(3, 3; 4)$	-	11	15
Folkman $(3, 3; 5)$	-	10	12
Treewidth 4	-	9	9
Treewidth 5	-	9	9
Snarks girth 5	-	14	20

Table 2: The largest number n of vertices solvable with each of the three main approaches, for each benchmark problem, within 4 hours of CPU time. For Q-SMS, the best QBF solver is reported. We implemented CCL only for the first three problems. Kochen-Specker instances start at $n = 15$; Q-static did not solve any.

solver	# solved	solver	# solved
2Qiss	110	qfun	104
cqesto	90	qute	72
minqbf-co	57	minqbf	55

Table 3: Number of solved graph instances by each solver.

We ran Qute with proof logging with a time limit of 30 minutes per instance (for longer runs, the proofs grow too large) and extracted and validated proofs of solved instances. Qute solved 70 instances and produced 1874 proofs, which, compressed with `xz`, total 350MB. All proofs were verified with `qrp2rup` and `gratchk` in under 4 hours.

6 Conclusion

This research explores novel techniques for solving quantified graph search problems with QBF solvers.

Q-SMS, which integrates SMS-style symmetry breaking into QBF solvers, matched CCL performance across multiple prominent graph search problems after a comprehensive evaluation. In contrast, the completely QBF-based method Q-static showed poor scaling beyond small graphs.

Our implementation extends Qute to produce verifiable proofs, a capability missing in CCL approaches. Q-SMS delivers equivalent performance with more straightforward implementation, broader problem applicability, and enables formal proof generation. Future work will expand 2Qiss to handle general QBFs and optimize its core algorithms to improve efficiency on complex graph problems further.

Acknowledgements

This research was funded in part by the Austrian Science Fund (FWF) 10.55776/COE12 and 10.55776/P36688. The

results were supported by the MEYS within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902 and are co-funded by the European Union under the project *ROBOPROX* (reg. no. CZ.02.01.01/00/22_008/0004590).

References

Balabanov, V.; and Jiang, J. R. 2012. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1): 45–65.

Balabanov, V.; Jiang, J. R.; Mishchenko, A.; and Scholl, C. 2016. Clauses Versus Gates in CEGAR-Based 2QBF Solving. In Darwiche, A., ed., *Beyond NP, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 12, 2016*, volume WS-16-05 of *AAAI Technical Report*. AAAI Press.

Biere, A.; Fazekas, K.; Fleury, M.; and Heisinger, M. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020. In Balyo, T.; Froleyks, N.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M., eds., *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, 51–53. University of Helsinki.

Bikov, A.; and Nenov, N. 2020. On the independence number of $(3, 3)$-Ramsey graphs and the Folkman number $F_e(3, 3; 4)$. *Australas. J. Combin.*, 77: 35–50.

Bodlaender, H. 1993. A tourist guide through treewidth. *Acta Cybernetica*, 11(1-2): 1–22.

Brinkmann, G.; and Goedgebeur, J. 2017. Generation of Cubic Graphs and Snarks with Large Girth. *J. Graph Theory*, 86(2): 255–272.

Budroni, C.; Cabello, A.; Gühne, O.; Kleinmann, M.; and Larsson, J.-Å. 2022. Kochen-Specker Contextuality. *Rev. Mod. Phys.*, 94: 045007.

Dorbec, P.; and Henning, M. A. 2024. The 1/3-conjectures for domination in cubic graphs. arXiv:2401.17820.

Erdős, P. 1967. Some remarks on chromatic graphs. *Colloq. Math.*, 16: 253–256.

Fazekas, K.; Niemetz, A.; Preiner, M.; Kirchweger, M.; Szeider, S.; and Biere, A. 2023. IPASIR-UP: User Propagators for CDCL. In Mahajan, M.; and Slivovsky, F., eds., *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. To appear.

Fichte, J. K.; Hecher, M.; Berre, D. L.; and Szeider, S. 2023. The silent (R)evolution of SAT. *Communications of the ACM*, 66(6): 64–72.

Folkman, J. 1970. Graphs with Monochromatic Complete Subgraphs in Every Edge Coloring. *SIAM Journal on Applied Mathematics*, 18(1): 19–24.

Gardner, M. 1976. Snarks, boojums, and other conjectures related to the four-color-map theorem. *Scientific American*, 4(234): 126–130.

Goedgebeur, J. 2020. On minimal triangle-free 6-chromatic graphs. *J. Graph Theory*, 93(1): 34–48.

Hecking-Harbusch, J.; and Tentrup, L. 2018. Solving QBF by Abstraction. *Electronic Proceedings in Theoretical Computer Science*, 277: 88–102.

Jaeger, F. 1985. A Survey of the Cycle Double Cover Conjecture. In Alspach, B.; and Godsil, C., eds., *Annals of Discrete Mathematics (27): Cycles in Graphs*, volume 115 of *North-Holland Mathematics Studies*, 1–12. North-Holland.

Janota, M. 2018a. Circuit-Based Search Space Pruning in QBF. In Beyersdorff, O.; and Wintersteiger, C. M., eds., *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, 187–198. Springer.

Janota, M. 2018b. Towards Generalization in QBF Solving via Machine Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 6607–6614. AAAI Press.

Janota, M.; Kirchweger, M.; Peitl, T.; and Szeider, S. 2025. Supplementary Material for Breaking Symmetries in Quantified Graph Search: A Comparative Study. https://doi.org/10.5281/zenodo.14530908.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2012. Solving QBF with Counterexample Guided Refinement. In Cimatti, A.; and Sebastiani, R., eds., *Theory and Applications of Satisfiability Testing - SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, 114–128. Springer Verlag.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2016. Solving QBF with Counterexample Guided Refinement. *Artificial Intelligence*, 234: 1–25.

Janota, M.; and Silva, J. P. M. 2011. Abstraction-Based Algorithm for 2QBF. In Sakallah, K. A.; and Simon, L., eds., *Theory and Applications of Satisfiability Testing - SAT 2011*, volume 6695, 230–244. Springer Verlag.

Jordan, C.; Klieber, W.; and Seidl, M. 2016. Non-CNF QBF Solving with QCIR. In Darwiche, A., ed., *Beyond NP, Papers from the 2016 AAAI Workshop.*, volume WS-16-05 of *AAAI Workshops*. AAAI Press.

Kelmans, A. 2006. Counterexamples to the Cubic Graph Domination Conjecture. arXiv:math/0607512.

Kirchweger, M.; Peitl, T.; and Szeider, S. 2023a. Co-Certificate Learning with SAT Modulo Symmetries. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence, IJCAI 2023*. AAAI Press/IJCAI.

Kirchweger, M.; Peitl, T.; and Szeider, S. 2023b. A SAT Solver's Opinion on the Erdős-Faber-Lovász Conjecture. In Mahajan, M.; and Slivovsky, F., eds., *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04-08, 2023, Alghero, Italy*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Kirchweger, M.; Scheucher, M.; and Szeider, S. 2022. A SAT Attack on Rota's Basis Conjecture. In *Theory and Applications of Satisfiability Testing - SAT 2022 - 25th International Conference, Haifa, Israel, August 2-5, 2022, Proceedings*.

Kirchweger, M.; Scheucher, M.; and Szeider, S. 2023. SAT-Based Generation of Planar Graphs. In Mahajan, M.; and Slivovsky, F., eds., *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04-08, 2023, Alghero, Italy*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Kirchweger, M.; and Szeider, S. 2021. SAT Modulo Symmetries for Graph Generation. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, LIPIcs, 39:1–39:17. Dagstuhl.

Kirchweger, M.; and Szeider, S. 2024. SAT Modulo Symmetries for Graph Generation and Enumeration. *ACM Trans. Comput. Log.*, 25(3): 1–30.

Kochen, S.; and Specker, E. 1967. The Problem of Hidden Variables in Quantum Mechanics. *J. Math. Mech.*, 17(1): 59–87.

Kostochka, A. V.; and Stodolsky, B. Y. 2005. On domination in connected cubic graphs. *DM*, 304(1-3): 45–50.

Lammich, P. 2017. Efficient Verified (UN)SAT Certificate Checking. In de Moura, L., ed., *Automated Deduction – CADE 26*, 237–254. Springer International Publishing. ISBN 978-3-319-63046-5.

Lange, A.; Radziszowski, S.; and Xu, X. 2012. Use of MAX-CUT for Ramsey arrowing of triangles. *arXiv*.

Li, Z.; Bright, C.; and Ganesh, V. 2023. A SAT Solver and Computer Algebra Attack on the Minimum Kochen-Specker Problem. *CoRR*, abs/2306.13319.

Löwenstein, C.; and Rautenbach, D. 2008. Domination in Graphs of Minimum Degree at least Two and Large Girth. *Graphs Comb.*, 24(1): 37–46.

Marques-Silva, J.; Lynce, I.; and Malik, S. 2021. Conflict-Driven Clause Learning SAT Solvers. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 133–182. IOS Press.

Mycielski, J. 1955. Sur le coloriage des graphs. *Colloquium Mathematicae*, 3(2): 161–162.

Peitl, T.; Slivovsky, F.; and Szeider, S. 2018. Polynomial-Time Validation of QCDCL Certificates. In Beyersdorff, O.; and Wintersteiger, C. M., eds., *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT*, volume 10929 of *Lecture Notes in Computer Science*, 253–269. Springer.

Peitl, T.; Slivovsky, F.; and Szeider, S. 2019. Qute in the QBF Evaluation 2018. *J. Satisf. Boolean Model. Comput.*, 11(1): 261–272.

Peres, A. 1991. Two simple proofs of the Kochen-Specker theorem. *J. Phys. A Math. Theor.*, 24(4): L175.

Prestwich, S. D. 2021. CNF Encodings. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 75–100. IOS Press.

Rabe, M. N.; and Tentrup, L. 2015. CAQE: A Certifying QBF Solver. In Kaivola, R.; and Wahl, T., eds., *Formal Methods in Computer-Aided Design - FMCAD 2015*, 136–143. IEEE Computer Soc.

Radziszowski, S. 2021. Small Ramsey Numbers. *The Electronical Journal of Combinatorics*.

Ramsey, F. P. 1930. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society*, s2-30(1): 264–286.

Reed, B. A. 1996. Paths, Stars and the Number Three. *Combin. Probab. Comput.*, 5: 277–295.

Robertson, N.; and Seymour, P. D. 2004. Graph Minors. XX. Wagner's conjecture. *J. Combin. Theory Ser. B*, 92(2): 325–357.

Samer, M.; and Veith, H. 2009. Encoding Treewidth into SAT. In *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*, 45–50. Springer Verlag.

Seymour, P. D. 1979. Sums of Circuits. In Bondy, J. A.; and Murty, U. S. R., eds., *Graph Theory and Related Topics*, 341–355. New York–London: Academic Press.

Sinz, C. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In van Beek, P., ed., *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, 827–831. Springer Verlag.

Szekeres, G. 1973. Polyhedral decompositions of cubic graphs. *Bulletin of the Australian Mathematical Society*, 8(3): 367–387.

Tseitin, G. S. 1968. On the Complexity of Derivation in Propositional Calculus. *Zap. Nauchn. Sem. Leningrad Otd. Mat. Inst. Akad. Nauk SSSR*, 8: 23–41. Russian. English translation in J. Siekmann and G. Wrightson (eds.) *Automation of Reasoning. Classical Papers on Computer Science 1967–1970*, Springer Verlag, 466–483, 1983.

Wetzler, N.; Heule, M. J. H.; and Hunt, W. A. 2014. 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*, 422–429. Springer Verlag.

Zhang, L.; and Malik, S. 2002. The Quest for Efficient Boolean Satisfiability Solvers. In Brinksma, D.; and Larsen, K. G., eds., *Computer Aided Verification: 14th International Conference (CAV)*, volume 2404, 17–36.

Zhang, T.; Peitl, T.; and Szeider, S. 2024. Small unsatisfiable k-CNFs with bounded literal occurrence. In Chakraborty, S.; and Jiang, J.-H. R., eds., *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 31:1–31:22. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-334-8.

Zhang, T.; and Szeider, S. 2023. Searching for smallest universal graphs and tournaments with SAT. In Yap, R., ed., *Proceedings of CP 2023, the 29th International Conference on Principles and Practice of Constraint Programming*, volume 280 of *LIPIcs*, 39:1–39:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.