End-to-End Certified Graph Colouring

Simon Dold
University of Basel, Switzerland

George Katsirelos
MIA Paris, INRAE, Université Paris-Saclay, France
Université de Toulouse, ANITI, France

Wietze Koops
Lund University, Sweden
University of Copenhagen, Denmark

Magnus O. Myreen
Chalmers University of Technology, Gothenburg, Sweden
University of Gothenburg, Sweden

Jakob Nordström
University of Copenhagen, Denmark
Lund University, Sweden

Andy Oertel
Lund University, Sweden
University of Copenhagen, Denmark

Yong Kiam Tan
Nanyang Technological University, Singapore
Institute for Infocomm Research (I^2R), A*STAR, Singapore

Abstract

Applied combinatorial optimization has witnessed a revolution in performance since the turn of the millennium, but the complexity of modern solvers is making bugs an ever more serious concern. The most promising remedy is to make solvers *certifying*, so that they use *proof logging* to generate machine-verifiable proofs of correctness. We present the first example of state-of-the-art certified graph colouring by equipping the solver ZykovColor with VeriPB proof logging. Combined with the formally verified CakePB checker, this provides end-to-end formally certified results. An experimental evaluation shows excellent results with only moderate overhead for proof logging and checking.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Logic and verification

Keywords and phrases graph colouring, proof logging, certifying algorithms

Digital Object Identifier 10.4230/LIPIcs.CP.2026.56

Supplementary Material *Dataset (Experiments)*: `https://doi.org/10.5281/zenodo.20035765`
Software (CertifyingZykovColor): `https://gitlab.com/MIAOresearch/software/CertifyingZykovColor`
archived at `swh:1:rev:3fcb93d52068cb936d3cfe3eb965df7879ea104`
Software (VeriPB): `https://gitlab.com/MIAOresearch/software/VeriPB`
archived at `swh:1:rev:3a539583463b250c799f7b78f29ce31088a12261`
Software (CakePB): `https://gitlab.com/MIAOresearch/software/cakepb`
archived at `swh:1:rev:a7593ef22de2fc0b47a688f2d4f08e6b742735af`

Funding *Simon Dold*: Swiss National Science Foundation (SNSF), project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA)
George Katsirelos: Funded by the AI Interdisciplinary Institute ANITI. ANITI is funded by the French "Investing for the Future – PIA3" program under the Grant agreement n°ANR-23-IACL-0002
Wietze Koops: Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation
Magnus O. Myreen: Swedish Research Council grant 2021-05165.
Jakob Nordström: Swedish Research Council grant 2024-05801 and Independent Research Fund Denmark grant 9040-00389B
Andy Oertel: Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation
Yong Kiam Tan: Singapore NRF Fellowship Programme NRF-NRFF16-2024-0002

Acknowledgements We would like to thank the anonymous reviewers of *CP* for their useful comments. We are grateful to Benjamin Bogø, Ciaran McCreesh, and Shuo Pang for helpful discussions. Also, we are thankful for stimulating conversations with and useful feedback from participants of the Dagstuhl workshop 25231 *Certifying Algorithms for Automated Reasoning* and the *2nd International Workshop on Highlights in Organizing and Optimizing Proof-logging Systems (WHOOPS '25)* at Institut Pascal in Paris. The third, fifth, and sixth authors wish to acknowledge that they have benefited greatly from being part of the Basic Algorithms Research Copenhagen (BARC) environment financed by the Villum Investigator grant 54451. Our computational experiments used resources provided by LUNARC at Lund University.

1 Introduction

The *graph colouring* problem is to assign colours to the vertices of a graph so that for every edge the two endpoints have different colours. Determining the minimum number of colours needed—the so-called *chromatic number*—is one of the foundational NP-hard problems [51], and is arguably one of the most well-studied problems in theoretical computer science, with strong evidence of exponential-time hardness even for solving the problem approximately (see, e.g., [3, 4, 18, 61, 72, 84]). Graph colouring is also important for many practical applications, including compiler optimization [16], numerical analysis [34, 43], and scheduling [57], and impressive progress in applied research [11, 12, 29, 32, 44, 45, 49] has led to solvers that can find optimal colourings even for very large graphs in spite of the theoretical hardness results. However, this leap in performance has come at the cost of increasing solver complexity, which makes it challenging to avoid errors in algorithm design and implementation. Due to the use of reductions and reformulations, it can even be tricky to recover solutions to the original problem. This means that graph colouring is afflicted by the same concerns as in other areas of combinatorial optimization, where even mature solvers sometimes erroneously claim optimality or return "solutions" which fail to satisfy all constraints [1, 17, 33, 35, 60, 78].

Important work on eliminating errors in combinatorial solvers has been done using testing techniques such as *fuzzing* and *delta debugging* [13, 14, 52, 69, 71], but testing alone is inherently unable to establish the absence of bugs. Formal methods can deliver correct-by-construction algorithms, but although there is impressive work on, e.g., Boolean satisfiability (SAT) solving [31], these techniques just cannot scale to the complexity of modern solvers. The most successful way to ensure correctness is instead to make solvers *certifying* [59], so that they use *proof logging* to emit machine-verifiable proofs of correctness. For SAT solving, a large number of proof systems for the conjunctive normal form (CNF) format have been developed [7, 42, 47, 81, 83], and proof logging has become a standard requirement. The success of SAT proof logging can be explained by three aspects. Firstly, it is easy to implement for most techniques. Secondly, it is efficient, with the overhead of proof generation usually staying within 10% of solving time, and proof checking taking at most a factor 10 longer than solving. Finally, each proof step is straightforward to check, making it possible to develop formally verified proof checkers [20, 21, 55, 76].

There has also been work on proof logging in, e.g., maximum satisfiability (MaxSAT) solving [9, 56, 66, 73, 74, 79], constraint programming [24, 70, 80], mixed integer programming [17, 25], and automated planning [27, 28, 67], but with much more limited success. Most of these approaches either struggle with supporting the full range of solver techniques, or introduce specialized rules for different techniques, complicating proof checking; moreover, sometimes proof steps cannot be verified efficiently and instead require trusting the solver.

In recent years, so-called *pseudo-Boolean (PB) proof logging*, based on reasoning with 0–1

linear inequalities, has changed this situation. The VeriPB proof system [8] has been shown to provide a unified proof logging method not only for the most advanced techniques in SAT solving [2, 8, 41] as well as for pseudo-Boolean optimization [36, 53], but also for MaxSAT solving [5, 6, 48], subgraph solving [37, 38, 39, 54], constraint programming [26, 40, 62, 63, 64], automated planning [23], and dynamic programming and decision diagrams [22]. This is even more striking since the proof system has no concept of graphs, non-Boolean variables, or planning tasks, but can still capture reasoning about such concepts using 0–1 linear inequalities. It should be pointed out, however, that the overhead for proof generation and checking has mostly been significantly larger for paradigms beyond SAT.

1.1 Our Contribution

In this work, we present for the first time a state-of-the-art graph colouring solver with proof logging, namely a version of ZykovColor [11] emitting VeriPB proofs.

ZykovColor is based on the Zykov recurrence [85] combined with SAT solving, using *reified constraints* to define new variables. To compute chromatic number lower bounds the solver uses cliques but also detects *Mycielski graphs* [68], for which our proof logging uses and extends work by Yolcu et al. [82]. The only technique for which we currently cannot provide proof logging is the fractional chromatic number bound computed via LP duality and column generation [46, 58, 65], since VeriPB cannot reason about rational values, but this feature has only a small impact on solver performance [11]. It has been shown previously that proof logging can serve as a powerful debugging tool even for mature solvers [6, 25, 37, 53, 54, 78], and our work on making ZykovColor certifying provides yet another example of this in that it helped fix a bug where a colouring with too many colours could be returned.

VeriPB proofs reason in terms of a 0–1 integer linear encoding of the graph colouring problem instance. Though this encoding is quite straightforward, our proof checking workflow includes formal verification of the translation by a CakePB frontend. Since CakePB also has a formally verified backend for checking VeriPB proofs, we achieve end-to-end formally certified results (similar to prior work [38]). An additional optimization is that an initial upper bound on the chromatic number can be used to shrink encoding size substantially.

Our experimental evaluation shows excellent performance overall, with average proof logging overhead of under 14% and average checking time of 1.4× solving time, but there are also some problematic instances. For graphs with large cliques, the proofs need to rederive for every colour that at most one vertex in the clique can take this colour (since all variables are Boolean), while the solver automatically knows that this holds for any colour. This causes an extra linear overhead for proof generation, similar to phenomena that have been observed in proof logging for constraint programming [62, 63, 64]. We also incur slowdowns for Mycielski graphs of high order, since the pseudo-Boolean proofs for their chromatic number formalize a rather subtle recursive argument that has to be explicitly written out and checked, whereas the solver can just use that the theorem can be trusted to be true.

We should emphasize that our performance statistics were achieved only after several rounds of careful optimization of the proof logging. For instance, it was crucial to generate constraints as lazily as possible, and previous proofs for Mycielski graphs [82] were made more efficient. For cliques with large intersections, it was important to recycle subderivations for these intersections. This shows that while pseudo-Boolean reasoning makes efficient proof logging possible even for advanced combinatorial algorithms, getting the low-level details right to achieve good performance in practice is arguably still a bit too complicated for comfort.

The proof checking performance also hinges on a large number of low-level optimizations, many of which do not seem to be related to graph colouring. Since writing an efficient proof

checker, including a formally verified backend, is immensely challenging, this provides an additional argument in favour of a simple, unified proof logging format such as VeriPB, that can be combined with different formally verified frontends for a wide range of problems.

1.2 Outline of This Paper

The rest of this manuscript is structured as follows. After reviewing preliminaries for pseudo-Boolean reasoning and graphs in Section 2 and the ZykovColor graph colouring solver in Section 3, we present our proof logging approach in Section 4. The special case of Mycielski graphs is discussed in Section 5. We explain our extensions to the formally verified proof checker CakePB in Section 6. In Section 7, we present the results of our experimental evaluation, after which we conclude in Section 8 with some final remarks.

2 Preliminaries

We start by discussing pseudo-Boolean reasoning (referring the reader to [8, 15] for more details), and then show how graph colouring can be encoded as a pseudo-Boolean problem. We use the standard notation $\mathbb{N}^+$ for positive integers and write $[n] = \{1, 2, \dots, n\}$ for $n \in \mathbb{N}^+$.

2.1 Pseudo-Boolean Reasoning with VeriPB

A *literal* ℓ over a *Boolean variable* x (which takes values in $\{0,1\} = \{\mathit{false}, \mathit{true}\}$) is either x itself or its negation $\overline{x} = 1 - x$. A *pseudo-Boolean (PB) constraint* is a 0–1 integer linear inequality $C \doteq \sum_i a_i \ell_i \geq A$, (where $\doteq$ denotes syntactic equivalence), which we can assume without loss of generality to be in *normalized* form, i.e., with coefficients $a_i \in \mathbb{N}^+$ and literals ℓ_i over distinct variables. The *negation* of C is the constraint $\neg C \doteq \sum_i a_i \overline{\ell}_i \geq -A + 1 + \sum_i a_i$. A *pseudo-Boolean formula* $F \doteq \bigwedge_{j=1}^{m} C_j$ is a conjunction of pseudo-Boolean constraints, which we can equivalently view as a set $F = \{C_j \mid j \in [m]\}$.

A *(partial) assignment* ρ is a (partial) function $x \mapsto \rho(x)$ from Boolean variables to $\{0,1\}$, which we extend to literals by defining $\rho(\overline{x}) = 1 - \rho(x)$. We say that ρ *satisfies* $\sum_i a_i \ell_i \geq A$ if it holds that $\sum_{\rho(\ell_i)=1} a_i \geq A$ (assuming normalized form), and ρ satisfies F if it satisfies all $C \in F$, in which case it is a *solution*. A *substitution* ω maps variables to $\{0,1\}$ or literals. Applying ω to C yields $C{\restriction_\omega} \doteq \sum_i a_i \omega(\ell_i) \geq A$ (collecting constants on the right and normalizing), and this notation is extended to formulas by applying ω to each of its constraints i.e., $F{\restriction_\omega} \doteq \{C{\restriction_\omega} \mid C \in F\}$. The *pseudo-Boolean optimization (PBO) problem* (F, f) is to minimize the *objective* $f = \sum_i a_i \ell_i$ (with $a_i \in \mathbb{N}^+$) over all solutions to F.

A VeriPB proof for an optimization problem (F, f) is used to establish upper and lower bounds on the minimal value of f subject to the constraints in F. An upper bound UB is shown by exhibiting a solution ρ to F such that $f(\rho) = UB$ and a lower bound $LB \leq UB$ is proven by deriving the PB constraint $f \geq LB$ with the VeriPB rules explained next.

VeriPB is based on the *cutting planes proof system* [19] with slight modifications as described in [15]. If $C \doteq \sum_i a_i \ell_i \geq A$ and $D \doteq \sum_i b_i \ell_i \geq B$ are *input axioms* in F or *literal axioms* $\ell_i \geq 0$, or if these constraints have previously been derived, then we can

- *sum* C and D to get $C + D \doteq \sum_i (a_i + b_i)\ell_i \geq A + B$ (in normalized form);
- *multiply* C by $m \in \mathbb{N}^+$ to get $m \cdot C \doteq \sum_i (m \cdot a_i)\ell_i \geq m \cdot A$;
- *divide* C by $d \in \mathbb{N}^+$ and round up to get $\mathsf{div}(C, d) \doteq \sum_i \lceil a_i/d \rceil \ell_i \geq \lceil A/d \rceil$;
- *saturate* C to get $\mathsf{sat}(C) \doteq \sum_i \min\{a_i, A\} \cdot \ell_i \geq A$ (which assumes normalized form for C).

A constraint C *propagates* a literal ℓ under a partial assignment ρ if there is no way C can be satisfied if ρ is extended to assign $\ell = 0$. We say that C is implied by *reverse unit*

propagation (RUP) [42, 26] from a set of constraints F if starting from the empty assignment and iteratively adding all literals propagated by $F \cup \{\neg C\}$ leads to some constraint being falsified. In such a case, it is allowed as a convenient shorthand to infer C from F by a single RUP step (which is sound since F implies C). We use $F \vdash C$ to denote that we have a derivation with the rules above of C from F, and write $F \vdash F'$ if $F \vdash C$ for all $C \in F'$.

The rules discussed so far can only derive implied constraints that preserve the set of solutions. The *redundance-based strengthening* rule, or just *strengthening*, allows to infer C from F by exhibiting an explicit substitution ω such that the cutting planes derivations

$$F \cup \{\neg C\} \vdash (F \cup C)\restriction_\omega \cup \{f \geq f\restriction_\omega\}\,. \tag{1}$$

can be shown to exist. If ρ satisfies F but falsifies C, it follows from (1) that the composed assignment $\rho \circ \omega$ satisfies $F \cup \{C\}$ and achieves objective value $f(\rho \circ \omega) \leq f(\rho)$. If ρ is a solution to F, an *objective-improving constraint* $f \leq -1 + f(\rho)$ (but in normalized form) can be inferred to force the search for strictly better solutions or make it possible to derive contradiction if the solution ρ is optimal.

2.2 Graph Colouring

We write $G = (V, E)$ for an undirected graph G with vertices $V(G) = V$ and edges $E(G) = E \subseteq \{\{u, v\} \mid \{u, v\} \subseteq V\}$, and say that $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. We write $N_G(v) = \{u \mid \{u, v\} \in E\}$ to denote the neighbours of a vertex v in G. A *k-colouring* of G is a function $f\colon V \to [k]$ for $k \in \mathbb{N}^+$, which is assumed to be *proper* unless otherwise stated, i.e., such that $f(u) \neq f(v)$ for all $\{u, v\} \in E$. The *chromatic number* $\chi(G)$ of G is the smallest k for which a proper k-colouring of G exists.

To encode the graph colouring problem for $G = (V, E)$ with some known upper bound $n \leq |V|$ on $\chi(G)$ in pseudo-Boolean form, we use vertex colour variables $x_{v,c}$ with the intended meaning "$f(v) = c$" and colour usage variables u_c to mean "colour c is in the image of f", where $v \in V$ and $c \in [n]$. It is not hard to see that the 0–1 integer linear program

$$\min \quad \sum_{c \in [n]} u_c \tag{2a}$$

$$A_v^1 \doteq \quad \sum_{c \in [n]} x_{v,c} \geq 1 \qquad \text{for each } v \in V \tag{2b}$$

$$A_v^2 \doteq \quad \sum_{c \in [n]} -x_{v,c} \geq -1 \qquad \text{for each } v \in V \tag{2c}$$

$$A_c^3 \doteq \quad |V| \cdot u_c + \sum_{v \in V} \overline{x_{v,c}} \geq |V| \qquad \text{for each } c \in [n] \tag{2d}$$

$$A_c^4 \doteq \quad \overline{u_c} + \sum_{v \in V} x_{v,c} \geq 1 \qquad \text{for each } c \in [n] \tag{2e}$$

$$A_{u,v,c}^5 \doteq \quad \overline{x_{u,c}} + \overline{x_{v,c}} \geq 1 \qquad \text{for each } \{u, v\} \in E,\ c \in [n] \tag{2f}$$

has the chromatic number $\chi(G)$ of G as its optimal value. By way of a brief argument, the constraints (2b) and (2c) enforce that the variables $x_{u,c}$ encode a function from V to $[n]$, i.e., that each $v \in V$ is mapped to exactly one $c \in [n]$. The constraints (2d) and (2e) force variables u_c to true precisely when c is in the range of this function. The *edge constraints* (2f) ensure that the function is a proper colouring. Minimizing the objective (2a) thus means finding a proper colouring with as few colours as possible, i.e., determining the chromatic number (and an important part of our certified graph colouring workflow is a formally machine-verified proof of this fact, so that there is no need to trust this proof sketch).

3 High-Level Description of the ZykovColor Graph Colouring Solver

The ZykovColor solver is based on SAT solving with user propagators [30], using ideas previously proposed by Hebrard and Katsirelos [45]. Rather than exploring the space of colourings, it argues by case analysis over pairs of non-neighbours u and v in $G = (V, E)$ that in any colouring either u and v are given the same colour, in which case they can be merged into a single vertex uv, or else an edge $\{u, v\}$ can be added, since their colours are different. This means that the chromatic number of G can be computed by the *Zykov recurrence* [85]

$$\chi(G) = \min\{\chi(G/(uv)), \chi(G + \{u, v\})\}\,, \tag{3}$$

where $G/(uv)$ denotes the graph with u and v merged into a single vertex uv with neighbours $N(uv) = N(u) \cup N(v)$, and $G + \{u, v\}$ is the graph with the edge $\{u, v\}$ added to E. The base cases of this recurrence are cliques whose size is the size of the colouring found.

To use the Zykov recurrence, ZykovColor considers only *equality* variables e_{uv} for $\{u, v\} \notin E$ with the intended meaning "$f(u) = f(v)$". A complete assignment to these variables corresponds to a leaf of the Zykov recurrence tree. Additionally, ZykovColor uses constraint propagators to enforce

$$e_{uv} \wedge e_{vw} \rightarrow e_{uw} \qquad \text{for all } u, v, w \in V\,, \tag{4}$$

i.e., that equality is transitive. Representing this in CNF would require $\Theta(n^3)$ clauses. ZykovColor instead uses a propagator to achieve the same propagation in linear time. More importantly, the propagator also computes a lower bound on the chromatic number of the current graph G_c obtained by contracting all pairs of vertices u, v for which e_{uv} is true and adding edges $\{u, v\}$ when e_{uv} is false. This bound is inferred by identifying two kinds of subgraphs H of G_c for which the chromatic number is readily determined:

- *Clique bounds:* If G_c contains as a subgraph a clique K_s of size s, then clearly $\chi(G_c) \geq s$.
- *Mycielski bounds:* Starting from a clique K_s detected in the previous step, the solver tries to find larger subgraphs H of G_c by iteratively applying a slight modification of the Mycielski graph construction (see Section 5). If successful, this yields a lower bound of $s + k$, where s is the size of the initial clique and k is the number of Mycielskian iterations.

When a lower bound computed by one of these methods exceeds the current upper bound, this triggers a failure, which the propagator explains with a clause $\bigvee_{\{u,v\}\notin E(G),\{u,v\}\in E(G_c)} e_{uv}$. The VeriPB proof then has to show that the corresponding pseudo-Boolean constraint $\sum_{\{u,v\}\notin E(G),\{u,v\}\in E(G_c)} e_{uv} \geq 1$ can be derived from the PB encoding in Section 2.2.

The ZykovColor solver tries to find colourings with an increasing number of colours $i = 2, 3, 4, \ldots$ until it reaches the chromatic number $\chi(G)$ of G. The current number of colours i serves as an upper bound for the bound propagators. In order to maintain soundness, the solver adds to each constraint an activation variable b_i encoding that it was inferred under the assumption that at most i colours are available. These activation variables can in turn be introduced in the VeriPB proof by constraints

$$(n - i) \cdot \overline{b_i} + \sum_{c\in[n]} \overline{u_c} \geq n - i \tag{5}$$

inferred by redundance-based strengthening.

4 Pseudo-Boolean Proof Logging for Graph Colouring

We now describe how to implement proof logging for the graph colouring techniques used by ZykovColor. Since our pseudo-Boolean encoding of the graph colouring problem

(as described in Section 2.2) does not have the equality variables e_{uv} that are used by ZykovColor, we have to introduce them in the proof. For any constraint $C \doteq \sum_i a_i \ell_i \geq A$ in normalized form and any fresh variable r, the *reified constraints* $r \Rightarrow C$ and $r \Leftarrow C$ expressing that r should be true if and only if C is satisfied can be derived by the redundance-based strengthening rule [41]. Syntactically, these reified constraints are given by $r \Rightarrow C \doteq A\overline{r} + \sum_i a_i \ell_i \geq A$ (as in (5)) and $r \Leftarrow C \doteq (\sum_i a_i - A + 1)\, r + \sum_i a_i \overline{\ell_i} \geq \sum_i a_i - A + 1$. We also use $r \Leftrightarrow C$ as a shorthand for the two reified constraints $r \Rightarrow C$ and $r \Leftarrow C$.

The meaning of e_{uv} can be written in terms of the colour variables as $\bigvee_{c\in[n]}(x_{u,c} \wedge x_{v,c})$, but this is not expressible as a single pseudo-Boolean constraint. In the proof, we therefore first define temporary variables $q_{uvc} \Leftrightarrow (x_{u,c} \wedge x_{v,c})$ for $c \in [n]$, using which we can introduce $e_{uv} \Leftrightarrow \bigvee_{c\in[n]} q_{uvc}$. Note that $x_{u,c} \wedge x_{v,c}$ and $\bigvee_{c\in[n]} q_{uvc}$ are equivalent to the pseudo-Boolean constraints $x_{u,c} + x_{v,c} \geq 2$ and $\sum_{c\in[n]} q_{uvc} \geq 1$, respectively. We then derive the constraints

$$E^1_{u,v,c} \doteq \overline{x_{v,c}} \Rightarrow (\overline{e_{uv}} + \overline{x_{u,c}} \geq 1) \doteq x_{v,c} + \overline{e_{uv}} + \overline{x_{u,c}} \geq 1\,, \tag{6}$$

$$E^2_{u,v,c} \doteq \overline{e_{uv}} \Rightarrow (\overline{x_{u,c}} + \overline{x_{v,c}} \geq 1) \doteq e_{uv} + \overline{x_{u,c}} + \overline{x_{v,c}} \geq 1 \tag{7}$$

(noting that $E^2_{u,v,c} \doteq E^2_{v,u,c}$) by reverse unit propagation (RUP) as described in Section 2.1. This allows us to infer the transitivity constraints $e_{uv} \wedge e_{vw} \to e_{uw}$, or in pseudo-Boolean form $\overline{e_{uv}} + \overline{e_{vw}} + e_{uw} \geq 1$, by first adding $E^1_{u,v,c}$, $E^1_{v,w,c}$, and $E^2_{u,w,c}$ and saturating to get $\overline{x_{u,c}} + \overline{e_{uv}} + \overline{e_{vw}} + e_{uw} \geq 1$, and then adding up these constraints for all $c \in [n]$ together with the at-least-one-colour constraint A^1_u and saturating to obtain the desired constraint.

4.1 Clique Bounds

A crucial technique in ZykovColor is to find cliques to infer lower bounds on the chromatic number, which in our encoding means establishing lower bound on the objective $\sum_{c\in[n]} u_c$.

Let us first show how to derive the constraint $\sum_{c\in[n]} u_c \geq |S|$ given a clique S in the original input graph G, which we refer to as an *unconditional clique*. Our proof starts by recovering the cardinality constraints $C_{S,c} \doteq \sum_{u\in S} \overline{x_{u,c}} \geq |S| - 1$, which is done by iteratively deriving the constraints $C_{T,c} \doteq \sum_{u\in T} \overline{x_{u,c}} \geq |T| - 1$ for subsets $T \subseteq S$ of increasing size. For the base case $|T| = 2$ we have that $C_{T,c}$ is an edge constraint (2f). Given $C_{T,c}$, we can derive $C_{T\cup\{v\},c}$ by multiplying $C_{T,c}$ by $|T| - 1$, adding all edge constraints $A^5_{u,v,c}$ for $u \in T$, and dividing by $|T|$.

As the next step, we observe that the colour usage constraint A^3_c implies the inequality $|S| \cdot u_c + \sum_{v\in S} \overline{x_{v,c}} \geq |S|$, which can be inferred formally by *weakening*, i.e., adding literal axioms $x_{v,c} \geq 0$, for all variables over vertices v that do not appear in the clique. Adding $|S| - 1$ times the constraint $C_{S,c}$ yields $|S| \cdot u_c + |S| \sum_{v\in S} \overline{x_{v,c}} \geq (|S| - 1)^2 + |S|$, which after division by $|S|$ results in the constraint $u_c + \sum_{v\in S} \overline{x_{v,c}} \geq |S|$.

Finally, summing $u_c + \sum_{v\in S} \overline{x_{v,c}} \geq |S|$ for each $c \in [n]$ with the at-least-one-colour constraints A^1_u for $u \in S$ produces the constraint $\sum_{c\in[n]} u_c \geq |S|$ as desired. By adding this constraint to the definition of the activation variable $b_{|S|-1}$ in (5) and saturating, it is also possible to derive the unit constraint $\overline{b_{|S|-1}} \geq 1$.

Let us next consider the scenario when the solver finds a clique in the current graph G_c after branching on some equality variables (i.e., after merging vertices and adding edges). In this case we can make a similar approach work, except that we do not always have access to original edge constraints $A^5_{u,v,c}$, but instead need to use the *reified* edge constraints $E^2_{u,v,c} \doteq \overline{e_{uv}} \Rightarrow A^5_{u,v,c}$ introduced by the Zykov recurrence. If we run the same derivation as before on these reified constraints, we obtain the same conclusion constraint but prefixed with all reification variables (as shown in [22]). Concretely for our case, this means that the very final step infers the clausal constraint $\overline{b_{|S|-1}} + \sum_{\{u,v\}\notin E(G),\{u,v\}\in E(G_c)} e_{uv} \geq 1$.

4.2 Proof Logging Optimizations

There are a few optimizations that are necessary to achieve acceptable overhead for the proof logging. First, and most obviously, we never use equality variables e_{uv} for edges $\{u, v\}$ that exist in the input graph G, since they are always false. Secondly, we delay introducing other equality variables e_{uv} until they are actually needed, rather than adding all such variables straight away. This is especially important since we also need to generate transitivity constraints for all introduced equality variables.

The third and most interesting optimization is related to the clique bound clauses. The most expensive part in the derivation of such clauses is the recovery of the cardinality constraints, which takes $\Theta(|S|^2)$ cutting planes steps per colour $c \in [n]$. However, Zykov-Color computes an initial large clique before the search starts and finds other unconditional cliques when it returns to the root node. The cardinality constraints derived for an unconditional clique S^r depend only on edges in G and is not reified on any equality variables. Therefore, when the solver finds another clique S' with $S' \cap S^r \neq \emptyset$, we can reuse part of the cardinality constraints derived for S^r, by weakening away variables $x_{u,c}$ for every $u \in S^r \setminus S'$ and every colour c from the cardinality constraints. This leaves a constraint $\sum_{u \in S' \cap S^r} x_{u,c} \geq |S' \cap S^r| - 1$, which we only need to extend to a cardinality constraint including the vertices $S' \setminus S^r$. Unfortunately, since these are the longest derivations, this technique only pays off when the intersection is large. The technique also does not work when the cardinality constraints are conditioned on some equality variables, because when such equality variables are weakened away the entire constraint becomes too weak.

5 Proof Logging for Mycielski Graphs

We now turn to a specific technique to prove lower bounds on the chromatic number, namely detecting *(generalized) Mycielski graphs* [68]. When applied repeatedly to a triangle-free starting graph, the original Mycielski construction yields triangle-free graphs with arbitrarily large chromatic number (for which studying cliques as described in Section 4.1 will not be very helpful). In our slightly generalized setting, a Mycielski graph consists of any base graph, a set of shadow vertices, and a top vertex. For each base graph vertex v, there is a shadow vertex v' adjacent to all neighbours of v in the base graph, and the top vertex is in turn adjacent to all shadow vertices. Let us write this down formally.

▶ **Definition 1** (Generalized Mycielski graph). *A graph $M = (V_M, E_M)$ is a* (generalized) Mycielski graph *with* base graph $B = (V_B, E_B)$ *if there exists a* shadow function $s\colon V_B \to V_M$ *and a* top vertex $w \in V_M$ *such that*

1. *B is a subgraph of M,*
2. *for all $v \in V_B$ it holds that $N_B(v) \subseteq N_M(s(v))$, and*
3. *for the image* $\mathrm{Im}(s)$ *of the function s it holds that* $\mathrm{Im}(s) \subseteq N_M(w)$.

A Mycielski graph is pure *if s is injective and V_B,* $\mathrm{Im}(s)$*, and $\{w\}$ are pairwise disjoint.*

We call $v' = s(v)$ the shadow vertex *of v. If a vertex v in the base graph is a neighbour of the top vertex w, then it is valid to set $s(v) = v$, in which case we say that v* self-shadows*. In case a vertex is a shadow of multiple vertices, we refer to this vertex as a* shared shadow*.*

For any Mycielski graph M with base graph $B = (V_B, E_B)$, we can generate a corresponding purified Mycielski graph $\widehat{M} = \left(V_{\widehat{M}}, E_{\widehat{M}}\right)$ with vertex and edge sets

$$V_{\widehat{M}} = V_B \cup \{v' \mid v \in V_B\} \cup \{\widehat{w}\}\,, \tag{8a}$$

$$E_{\widehat{M}} = E_B \cup \{\{u, v'\} \mid u, v \in V_B, \{u, v\} \in E_B\} \cup \{\{v', \widehat{w}\} \mid v \in V_B\}\,. \tag{8b}$$

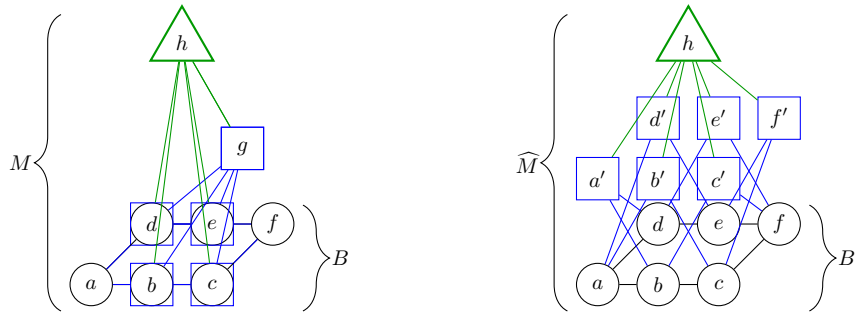


Figure 1 A Mycielski graph M (left) and its corresponding purified graph $\widehat{M}$ (right). The circles denote base vertices, squares denote shadow vertices, and the triangle denotes the top vertex.

Note that for each edge $\{u, v\} \in E_B$, the edge set $E_{\widehat{M}}$ in (8b) contains two additional edges, namely $\{u', v\}$ and $\{u, v'\}$. Hence, we have $\left|V_{\widehat{M}}\right| = 2 \cdot |V_B| + 1$ and $\left|E_{\widehat{M}}\right| = 3 \cdot |E_B| + |V_B|$.

▶ **Example 2.** Figure 1 gives an example of a Mycielski graph M whose base graph is a cycle of length 6 on the vertices a–f and whose top vertex is h. The vertices b–e self-shadow, while g is a shared shadow of a and f, and so M is not a pure Mycielski graph.

To construct the corresponding purified Mycielski graph $\widehat{M}$, we first duplicate vertices b–e to create the shadows b'–e'. We also duplicate g, where one copy serves as the shadow of a and the other serves as the shadow of f. Finally, we remove all edges that do not belong to the pure Mycielski graph construction, such as edges from h to the base graph.

By taking an entire Mycielski graph M with base graph B as a new base graph B' in Definition 1, we get a *level-2 Mycielski graph*, and iterating this k times yields the *level-k Mycielski graph* with base graph B. The crucial property of these graphs is that their chromatic number grows with k: if M_k is a level-k Mycielski graph with base graph B, then $\chi(M_k) \geq \chi(B) + k$. The ZykovColor solver detects subgraphs that are level-k Mycielski graphs with a clique as base graph. To show that $\chi(M_k) \geq \chi(B) + k$, it is sufficient to focus on a single iteration of the Mycielski construction and then argue by induction. We describe next how to show that for a (level-1) Mycielski graph M with base graph B, it holds that $\chi(M) \geq \chi(B) + 1$. We start with the mathematical argument in natural language and then translate it into a VeriPB proof (which is needed since the proof checker does not know any graph theory, or even what a graph is, only how to manipulate 0–1 linear inequalities).

We first show that we can focus on the case that the Mycielski graph is pure. Intuitively, we can purify a Mycielski graph by adding a copy of every vertex (with the same neighbourhood) that is a shared shadow or a shadow in the base graph, and then removing any extra edges (as in Example 2). Since the copy v' of a vertex v can receive the same colour as v and removing edges only decreases the chromatic number, the purified graph cannot have a higher chromatic number than the original impure Mycelski graph.

▶ **Proposition 3.** *For a Mycielski graph M and a pure Mycielski graph $\widehat{M}$ with the same base graph B, it holds that $\chi(M) \geq \chi(\widehat{M})$.*

To express this as a pseudo-Boolean constraint, assume that we already have an encoding for the chromatic number of M over the variables $x_{v,c}$ and u_c. To reason about the chromatic number of $\widehat{M}$, we derive the constraints $\widehat{A}^1_v$, $\widehat{A}^2_v$, $\widehat{A}^3_c$ and $\widehat{A}^5_{u,v,c}$ over variables $\widehat{x}_{v,c}$ and $\widehat{u}_c$ corresponding to A^1_v, A^2_v, A^3_c and $A^5_{u,v,c}$. This can be done using redundance-based strengthening with a witness substitution mapping the variables for $\widehat{M}$ to the corresponding

variables for M. Using the same substitution, we can also infer $\sum_{c\in[n]} u_c \geq \sum_{c\in[n]} \widehat{u}_c$, which is the PB encoding of $\chi(M) \geq \chi(\widehat{M})$. As a low-level optimization, we use an activation variable to avoid using this substitution for multiple constraints (see Appendix A for details).

We now proceed with the main part of the argument, namely showing that the chromatic number of a pure Mycielski graph is larger than that of its base graph.

▶ **Proposition 4.** *For a pure Mycielski graph $\widehat{M}$ with base B it holds that $\chi(\widehat{M}) \geq \chi(B)+1$.*

Although Proposition 4 is of course well-known, we present the proof here, since this is what we will need to formalize in the VeriPB proof system.

Proof. We wish to show that without loss of generality we can assume that each base vertex v has the same colour as its shadow vertex v', after which it follows that top vertex will need an extra colour. This is a surprisingly delicate argument, which requires two steps of copying colours first from base vertices to shadows and then in the other direction. Throughout the proof, let c_w be the colour of the top vertex w.

In the first step, we copy colours $c \neq c_w$ from the base graph to the shadows. If a vertex $v \in V_B$ has colour $f(v) = c \neq c_w$, then v' can take the same colour c, since $N_{V_{\widehat{M}}}(v') = N_{V_B}(v) \cup \{w\}$ and both w and the neighbourhood of v have colours different from c. Hence, without loss of generality it holds for any vertex $v \in V_B$ with colour $c \neq c_w$ that v' has the same colour as v.

In the second step, for each base graph vertex $v \in V_B$ with colour c_w we instead copy the colour $f(v') = c' \neq c_w$ of its shadow v' to v. Fix such a base vertex v with $f(v) = c_w$. Since $N_{V_B}(v) \subseteq N_{V_{\widehat{M}}}(v')$, the neighbours of v in the base graph do not have colour c', and they also cannot have colour c_w since they are neighbours of v. Therefore, after the first step above they have the same colour as their shadows. Since no base or shadow neighbour of v has colour c', this is a valid colour for v.

These two steps show that without loss of generality it holds for $\widehat{M}$ that vertices and their shadows have the same colours. Since the vertices in V_B jointly use at least $\chi(B)$ different colours, the same holds for the shadow vertices. The top vertex requires an additional colour since it is adjacent to all shadow vertices, from which we conclude that $\chi(\widehat{M}) \geq \chi(B)+1$. ◀

We now show how to carry out this reasoning as a sequence of VeriPB proof steps (deferring to Appendix A details such as exactly how certain constraints can be inferred by strengthening):

1. Introduce constraints
$$K^1_{v,b} \doteq \qquad \overline{\widehat{x}_{v,b}} + \widehat{x}_{v',b} + \widehat{x}_{\widehat{w},b} \geq 1 \qquad \text{for each } v \in V_B,\ b \in [n]\,, \tag{9}$$
using strengthening with witness substitution $\{\widehat{x}_{v',c} \mapsto \widehat{x}_{v,c} \mid c \in [n]\}$. This says that without loss of generality each shadow vertex v' has the colour of its corresponding base vertex v, unless this base vertex colour coincides with the colour of the top vertex.
2. Introduce constraints
$$K^2_{v,b} \doteq \qquad \overline{\widehat{x}_{v,b}} + \widehat{x}_{v',b} \geq 1 \qquad \text{for each } v \in V_B,\ b \in [n]\,, \tag{10}$$
with witness substitution $\{\widehat{x}_{v,c} \mapsto \widehat{x}_{v',c} \mid c \in [n]\}$. This encodes that without loss of generality each base vertex has the same colour as the corresponding shadow vertex.
3. Introduce colour usage variables for the base graph using reification constraints
$$K^{3.1}_b \doteq \qquad |V_B| \cdot \widehat{u}^B_b + \sum_{v\in V_B} \overline{\widehat{x}_{v,b}} \geq |V_B| \qquad \text{for each } b \in [n] \tag{11}$$
$$K^{3.2}_b \doteq \qquad \overline{\widehat{u}^B_b} + \sum_{v\in V_B} \widehat{x}_{v,b} \geq 1 \qquad \text{for each } b \in [n] \tag{12}$$

where the variable $\widehat{u}_b^B$ is 1 if and only if colour b is used for a vertex in B.

4. Derive

$$K_b^4 \doteq \qquad \overline{\widehat{x}_{\widehat{w},b}} + \overline{\widehat{u}_b^B} \geq 1 \qquad \text{for all } b \in [n] \tag{13}$$

by reverse unit propagation (RUP) on the constraints $\widehat{A}^5_{v',\widehat{w},b}$ and $K^2_{v,b}$ for $v \in V_B$ together with $K_b^{3.2}$. This constraint enforces that the colour of the top vertex is not used in the base graph.

5. Infer

$$K_b^5 \doteq \qquad \widehat{u}_b + \overline{\widehat{u}_b^B} + \overline{\widehat{x}_{\widehat{w},b}} \geq 2 \qquad \text{for each } b \in [n] \tag{14}$$

using a cutting planes derivation from constraints $\widehat{A}^3_b$, $K^2_{v,b}$, $K_b^{3.2}$, and K_b^4.

6. By adding K_c^5 and $\widehat{A}^1_{\widehat{w}}$ for each $c \in [n]$, derive the constraint

$$K^6 \doteq \qquad \sum_{c\in[n]} \widehat{u}_c \geq \sum_{c\in[n]} \widehat{u}_c^B + 1\,. \tag{15}$$

7. Delete the constraints $K^1_{v,b}$, $K^2_{v,b}$, K^4_b and K^5_b for each $v \in V_B,\ b \in [n]$.

To finish the proof, we add the inequalities $\chi(M) \geq \chi(\widehat{M})$ and $\chi(\widehat{M}) \geq \chi(B) + 1$, which yields $\chi(M) \geq \chi(B) + 1$.

Now consider a level-k Mycielski graph M_k with base graph B. Write $M_0 = B$ and let M_i be the level-i Mycielski graph for B. Then we can prove that $\chi(M_i) \geq \chi(M_{i-1}) + 1$ for $1 \leq i \leq k$, and adding these inequalities yields the required inequality $\chi(M_k) \geq \chi(M_0) + k = \chi(B) + k$. Such a telescoping step can also be performed in the VeriPB proof system by adding the derived inequalities. Finally, if the graph G itself is not a Mycielski graph but just has a Mycielski graph M_k as a subgraph, then we also add $\chi(G) \geq \chi(M_k)$. Showing this inequality in VeriPB is straightforward (and we again refer to Appendix A for details).

Why do we take the detour via the purified Mycielski graph in the proofs above? A crucial part of the proof is arguing that each shadow vertex can take the same colour as its base vertex. However, in case of shared shadows, this would imply that a shadow takes the same colour as two base vertices, which may not be possible. We can see this in Figure 1, where there is no valid 2-colouring of the 6-cycle assigning the same colour to vertices a and f. Hence, there is no way to construct a 2-colouring of the base graph where the vertex g, which is the shared shadow of a and f, has the same colour as both a and f.

In the proof, we avoid generating copy graphs that are pure multi-level Mycielski graphs, which would in principle be possible but could cause serious performance problems. An extreme example would be an n-clique, which can be seen as a non-pure level-n Mycielski graph where the empty graph is the base graph and every shadow function is the identity. Here the number of vertices for the corresponding pure multi-level Mycielski graph grows exponentially in n, so making a pure copy would cause an exponential overhead in logging.

As has already been mentioned, this part of our work is inspired by Yolcu et al. [82], who formalized proofs for the chromatic number of Mycielski graphs for variants of the *PR (propagation redundancy)* proof system [47]. They also use the two-step process of copying the colours from the base to the shadows if possible and then copying the colours of the shadows to the base. However, our approach is more general, as it is not restricted to pure Mycielski graphs with the 2-clique as the base graph, and we also consider graphs that are themselves not Mycielski graphs but have a Mycielski graph as a subgraph (which is a highly non-trivial extra complication). Yet another difference is that our proof uses a factor n fewer

$$\mathsf{is\_colouring}\ k\ \chi\ (V,E) \stackrel{\text{def}}{=} (\forall u.\ u < V \Rightarrow \chi\ u < k) \wedge \forall u\ v.\ u < V \wedge v < V \wedge u \neq v \wedge \mathsf{is\_edge}\ E\ u\ v \Rightarrow \chi\ u \neq \chi\ v$$

$$\mathsf{min\_colour}\ G \stackrel{\text{def}}{=} \mathsf{min}_{\mathsf{set}}\ \{\ k \mid (\exists \chi.\ \mathsf{is\_colouring}\ k\ \chi\ G)\ \}$$

Figure 2 Formalized semantics of graph colouring problems.

steps (where n is the known upper bound on the number of colours) for this copying but factor n larger witness substitutions, which can be more efficient for proof checking.

To argue about the length of the generated proof for a level-k Mycielski graph M_k with base graph B, let us consider each step in the proof separately. The proof length is most affected by introducing the constraints $\widehat{A}^5_{u,v,c}$ as well as $K^1_{v,c}$ and $K^2_{v,c}$. There are $\mathrm{O}(n \cdot |E(M_k)|)$ and $\mathrm{O}(n \cdot |V(M_k)|)$ such steps, respectively. With k levels of recursion, these steps are executed at most k times, yielding total length $\mathrm{O}\big(k \cdot n \cdot (|E(M_k)| + |V(M_k)|)\big)$. For pure Mycielski graphs, the number of edges and vertices is at least halved in each iteration, providing a total proof length of $\mathrm{O}\big(n \cdot 3^k \cdot (|V_B| + |E_B|)\big)$. Restricting the analysis further to the 2-clique as the base graph and k colours, for which Yolcu et al. [82] obtain proof length $\mathrm{O}(k^2 \cdot 3^k)$, the length of our proof is $\mathrm{O}(k \cdot 3^k)$, but in their proofs the clauses and witness substitutions have constant size, while we have linear scaling for these parameters. Therefore, in the end we get the same asymptotic growth $\mathrm{O}(k^2 \cdot 3^k)$ for proof size.

6 End-to-End Formally Verified Proof Checking

CakePB is a verified pseudo-Boolean proof checking backend which offers a range of frontends for certified subgraph solving [38]. VeriPB, CakePB, and certifying solvers operate in tandem in an end-to-end certification workflow as follows:

1. Given an input graph problem, the certifying solver encodes its high-level (non-PB) reasoning as a series of PB proof steps, analogous to the reasoning described in this paper.
2. This proof is checked and *elaborated* by VeriPB to a *kernel* proof format, adding details that make it easy for CakePB to check the proof. The kernel format is designed so that the CakePB proof checker can be readily optimized and verified.
3. For each supported graph problem class, CakePB offers a suite of formally verified *frontends* that parse and encode input graph problems into pseudo-Boolean format. The encoded PB problem is then checked by CakePB against the elaborated proof.

The whole CakePB implementation is formally verified in the HOL4 theorem prover [75] down to its machine-code implementation, giving high assurance that conclusions derived via this workflow are sound with respect to the semantics of the input graph problem.

Our CakePBcolour proof checker for certified graph colouring adds a new verified frontend to CakePB, albeit with key changes to the input interface to improve performance of the certification workflow. Firstly, the complete encoding as described in Section 2.2 is expensive to produce for large graphs. For CakePBcolour, we allow the solver to provide an initial upper bound $n \leq |V|$ on the number of colours. We also generate constraints lazily so that only the subset of constraints used in the proof is produced. This means that the PB proof can only certify that the input graph is *not* k-colourable for some $k \leq n$, so the solver has to produce a colouring that can be checked by the CakePBcolour frontend.

Our formal definition of graph colouring problem semantics is shown in Figure 2. Throughout the formalization $G = (V, E)$ is an undirected input graph on vertices $\{0, 1, \ldots, |V| - 1\}$, and E is its edge set which can be looked up with the is_edge function; colours are represented using the natural numbers $\mathbb{N}$. The predicate is_colouring defines what it means for a colouring

$\vdash$ good_graph $G \Rightarrow$ (check_colouring $k\ \chi\ G \iff$ is_colouring $k\ \chi\ G$)
$\vdash$ good_graph $G \wedge$ lazy_enc $G\ anns =$ Some $(n,obj,fml) \wedge$
$(\forall\, w.$ satisfies $w\ fml \Rightarrow lb \leq$ eval_obj $obj\ w) \wedge lb \leq n \Rightarrow$ is_colouring $k\ \chi\ G \Rightarrow lb \leq k$

Figure 3 Correctness theorems for the colouring checker (top) and the PB encoding (bottom).

function $f\colon \{0,1,\ldots,|V|-1\} \to \mathbb{N}$ to be a proper k-colouring, ignoring self-loops; and min_colour G defines the chromatic number of G as expected.

The high-level correctness theorems for our new frontend are shown in Figure 3, where good_graph is a consistency requirement for the input graph file which is checked by the DIMACS format parser. In the upper-bounding theorem (Figure 3, top), check_colouring is a straightforward pass over the graph applying the colouring f to check that it is indeed a k-colouring. In the lower-bounding theorem (Figure 3, bottom), lazy_enc is our lazy encoding of the input graph G, based on the annotated constraint usage hints ($anns$) from the solver. It produces the initial colour bound n, a PB objective obj, and a set of PB constraints fml. The theorem says that if the PB problem has a provable lower bound $lb \leq n$ on the objective value for all satisfying assignments, then any k-colouring of G must satisfy $lb \leq k$.

The theorems from Figure 3 justify our frontend implementation, which we connect to the verified CakePB backend. The completed checker is then compiled down to machine code using a formally verified compiler [77]. This results in an end-to-end machine-code level correctness theorem for CakePBcolour (not shown here), which states that if the checker prints out `s VERIFIED CHROMATIC NUMBER = k`, then the input graph file indeed parses to a graph G with $\chi(G) = k$, and moreover the colouring provided is a valid k-colouring of G. In practice, CakePBcolour can also certify non-tight lower and upper bounds for $\chi(G)$.

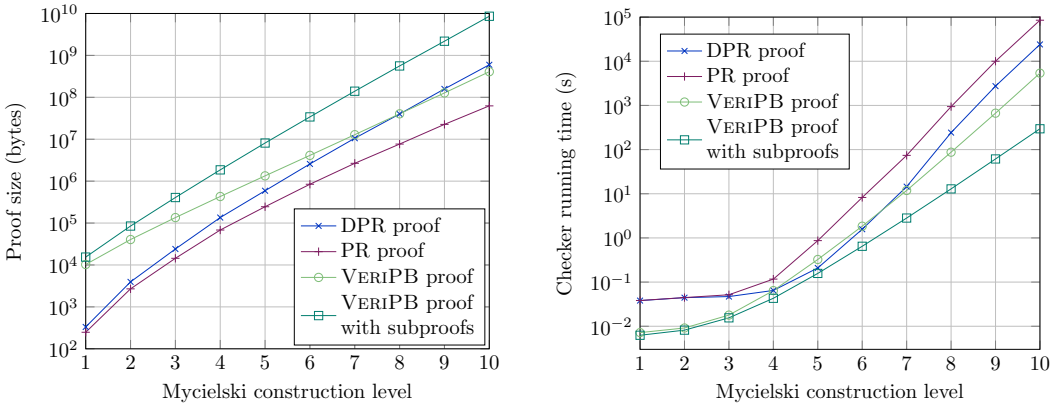
7 Experimental Evaluation

We have implemented VeriPB proof logging in the ZykovColor graph colouring solver [11], covering all techniques used by the default configuration `-zykov-color` except for the fractional bound computed by column generation (since VeriPB cannot reason with rational numbers). Fortunately, this is the feature that has the smallest impact on performance [11]. To evaluate our implementation, we ran experiments on machines with Intel(R) Core(TM) i5-1145G7 CPUs, a 100GB solid-state drive as storage, and 16GB of memory, where we restricted the memory limit to 14GB. Each instance ran exclusively on one machine.

As we started implementing proof logging for ZykovColor, the proof checker detected that in many cases the colouring returned used more colours than the claimed chromatic number. This was caused by a bug in preprocessing, where the solver first created a heuristic colouring and then reduced the graph and computed another heuristic colouring on the reduced version. The correct upper bound is the minimum number of colours in these two colourings, but the solver only stored and reported the latter colouring regardless of how many colours it used. We fixed this bug before conducting our proof logging experiments.

7.1 Comparison of Mycielski Graph Proofs

We first compare our proof method for Mycielski graphs to that of Yolcu et al. [82]. While our approach can deal with any base graph for the construction, Yolcu et al. only consider 2-cliques as base graphs. Accordingly, we only consider such base graphs for our experiments, resulting in chromatic number $k+2$ for Mycielski construction level k. The proofs were generated using standalone scripts. The DPR and PR proofs by Yolcu et al. were checked by



(a) Size of the proof for Mycielski graphs.

(b) Time required to check Mycielski proofs.

Figure 4 Comparison of DPR and PR proofs and our VeriPB proofs for Mycielski graphs. The "VeriPB proof with subproofs" plots add detailed subproofs for some VeriPB proof steps.

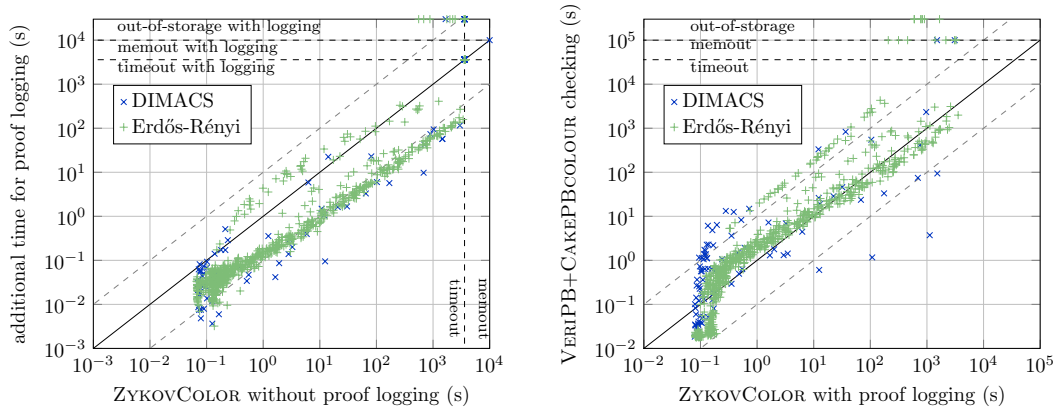
the DPR-TRIM checker, and our proofs were checked using VeriPB. For complex proof steps that VeriPB is in principle able to check automatically without additional information, such as redundance-based strengthening steps, it is also possible to add explicit subproofs to make the checking more efficient. In our experiments, we also evaluated proof logging with such explicit subproofs for some otherwise time-consuming strengthening steps.

Figure 4a compares how the sizes of the four types of Mycielski graph proofs scale. The PR proof is smallest and the VeriPB proof with subproofs is the largest, but it should be noted that the VeriPB format is more verbose than both the PR and DPR formats. Figure 4b shows how long it takes to check each proof for increasing levels of the construction. According to the experimental data, it seems that the startup of DPR-TRIM is slower than VeriPB. Except for levels 5 and 6, where checking the VeriPB proofs are slightly slower than checking the DPR proof, our VeriPB proofs are faster to check and seem to have better asymptotic scaling than both the DPR and PR proofs. Our VeriPB proof with subproofs is checked significantly faster than both the DPR and PR proofs and also seems to scale better than the VeriPB proof without subproofs. Therefore, in comparison to the approach by Yolcu et al., the checking performance for our VeriPB proofs is better and our approach additionally offers more features and flexibility as discussed in Section 5.

We can conclude that proof size does not necessarily correlate with proof checking time, and that performance can be improved if the proof system has flexible support for different levels of detail in subproofs. There is a trade-off here in that spelling out full details in subproofs can slow down the checker significantly, since a third of the checking time for VeriPB proofs with subproofs is spent on parsing. Many SAT proof logging systems have a binary format to allow faster parsing, and this would be desirable also for VeriPB.

7.2 Performance Results for Graph Colouring Solver

To evaluate the ZykovColor proof logging, we used the 137 graph instances in the DIMACS colouring benchmark set [50], to which we added the first 20 Erdős-Rényi random graphs from Brand et al. [10] for each vertex set size and edge probability to get 1000 more instances. The proofs were elaborated with VeriPB and checked by the formally verified proof checker CakePBColour. We used a time limit of 3 600s for solving and 36 000s for checking the proofs. For all averages, we use the shifted geometric mean with an additive shift of 1s.



(a) Overhead for proof logging.

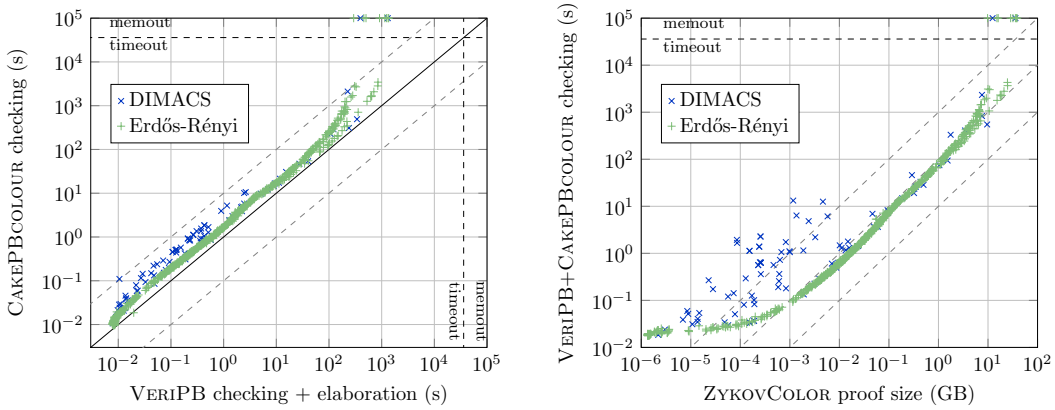
(b) Proof logging vs. checking time.

Figure 5 Proof logging overhead in ZykovColor and comparison of proof logging and checking.

In Figure 5a, we compare the running time of ZykovColor without proof logging to the additional time required to generate the proof (where the lower off-diagonal dashed line is the desired 10% overhead). Out of the 1137 instances, 769 are solved by ZykovColor without proof logging and 759 with proof logging; the remaining 10 instances fail because the proofs exceed the 100GB limit. For the 759 instances solved by ZykovColor with proof logging, the solver with proof logging is on average 13.7% slower than the version without proof logging. Overall, this shows a moderate overhead for generating the proof, but there are about 2% of the instances where proof logging slows down the solver by at least 4×. This is mainly due to large proofs arising from Erdős-Rényi instances with an edge probability of 90%, which have many large cliques. Since our proof for the clique bound has to be repeated for each colour, this introduces an additional linear factor overhead for proof logging compared to the time for detecting the clique in the solver. Similar problems have also been observed in proof logging for constraint programming [62, 63, 64].

In Figure 5b, the running time of ZykovColor with proof logging is compared to the combined time of VeriPB and CakePBcolour for elaborating and checking the proof and obtaining the formally verified conclusion (where the upper off-diagonal dashed line is the desired 10× limit on overhead). Out of the 759 instances solved by ZykovColor with proof logging, verification successfully terminated for 742 proofs while 10 instances failed due to CakePBcolour exhausting the memory limit and 7 more instances failed because the elaborated proofs exceeded 100GB. For the 742 successfully checked instances, the average combined running time of VeriPB and CakePBcolour is 1.4× the running time of ZykovColor with proof logging, which is relatively low compared to similar proof logging works with formally verified conclusions [38, 48, 53]. In addition to the 2% of the instances with a large logging overhead, proofs for large Mycielski graphs were also slow to check. The reason for this is that proving the Mycielski bound requires formalizing the somewhat subtle argument in Proposition 4 in pseudo-Boolean reasoning, and writing and checking this proof is expensive, while the solver can just use the bound as a known theorem.

To analyse the proof checking performance in more detail, we compare the running time of VeriPB to CakePBcolour in Figure 6a. CakePBcolour is roughly 2× slower than VeriPB, and its performance degrades in comparison to VeriPB when memory usage gets close to the memory limit since it uses garbage collection. There are some DIMACS instances where CakePBcolour is significantly slower. These instances are solved in preprocessing



(a) Elaboration vs. formally verified checking time. **(b)** ZykovColor proof size vs. checking time

Figure 6 Detailed analysis of proof checking performance.

by finding a large clique, which shows that the performance of the pure cutting planes proof checker could be improved in CakePB. In Figure 6b, we compare the checking performance to the size of the proof, where checking generally scales linearly in proof size. However, we again observe that the performance of CakePBcolour degrades towards the memory limit. Similarly, checking the DIMACS instances solved by finding an initial large clique takes significantly longer, which can be attributed to the compactness of the cutting planes proofs.

Overall, our approach performs very well with respect to both proof logging and proof checking compared to prior work on certifying solvers for NP-hard graph problems [37, 38, 39]. This is probably because ZykovColor uses SAT solving as its main procedure, which VeriPB supports efficiently, but is also thanks to several rounds of careful, in-depth optimizations of the proof logging code, such as lazily introducing the equality variables and reusing parts of the proof for the clique bound as described in Section 4.2.

8 Concluding Remarks

In this work, we present an efficient toolchain for end-to-end verification of graph colouring. We implement pseudo-Boolean proof logging in the state-of-the-art solver ZykovColor, covering a wide range of techniques in graph colouring, with the proofs checked by the VeriPB and CakePB tools. Since we use general-purpose pseudo-Boolean proof logging, we are optimistic that this approach should be possible to adapt also to other graph colouring solvers. Our work demonstrates, for the first time, that efficient proof logging with fully formally verified checking is possible for graph problems, even though the underlying proof logging system does not have any dedicated rules for dealing with graphs.

A key technique in ZykovColor is the use of chromatic number lower bounds of (sub)graphs, where we improve on work by Yolcu et al. [82] by providing more general proofs that are also more efficient to check.

An interesting future challenge is whether VeriPB could be made to support (efficient) proof logging for column generation, and in particular for computing fractional chromatic numbers. Another direction for future research is to improve even further the performance of proof logging and checking, especially for graphs containing many large cliques. Our experiments also show that a significant bottleneck consists of writing and parsing proof files, which could be improved by introducing a binary file format for VeriPB.

References

1 Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Metamorphic testing of constraint solvers. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP '18)*, volume 11008 of *Lecture Notes in Computer Science*, pages 727–736. Springer, August 2018.

2 Markus Anders, Bart Bogaerts, Benjamin Bogø, Arthur Gontier, Wietze Koops, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Adrián Rebola-Pardo, and Yong Kiam Tan. Faster certified symmetry breaking using orders with auxiliary variables. In *Proceedings of the 40th AAAI Conference on Artificial Intelligence (AAAI '26)*, pages 14140–14148, January 2026.

3 Paul Beame, Joseph C. Culberson, David G. Mitchell, and Cristopher Moore. The resolution complexity of random graph k-colorability. *Discrete Applied Mathematics*, 153(1-3):25–47, December 2005.

4 Richard Beigel and David Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, February 2005.

5 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, Tobias Paxian, and Dieter Vandesande. Certifying without loss of generality reasoning in solution-improving maximum satisfiability. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:28, September 2024.

6 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.

7 Armin Biere. Tracecheck. `http://fmv.jku.at/tracecheck/`, 2006.

8 Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.

9 Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8-9):606–618, June 2007. Extended version of paper in *SAT '06*.

10 Timo Brand, Daniel Faber, Stephan Held, and Petra Mutzel. A customized SAT-based solver for graph coloring, October 2025. Repository with experimental data. `doi:10.5281/zenodo.17328845`.

11 Timo Brand, Daniel Faber, Stephan Held, and Petra Mutzel. A customized SAT-based solver for graph coloring. In *2026 Proceedings of the SIAM Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 142–155. SIAM, 2026.

12 Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, April 1979. URL: `https://doi.org/10.1145/359094.359101`.

13 Robert Brummayer and Armin Biere. Fuzzing and delta-debugging SMT solvers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories (SMT '09)*, pages 1–5, August 2009.

14 Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, July 2010.

15 Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 233–350. IOS Press, 2nd edition, February 2021.

16 Gregory J. Chaitin. Register allocation & spilling via graph coloring. *SIGPLAN Not.*, 17(6):98–101, June 1982.

17 Kevin K. H. Cheung, Ambros M. Gleixner, and Daniel E. Steffy. Verifying integer programming results. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO '17)*, volume 10328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, June 2017.

18 Jonas Conneryd, Susanna F. de Rezende, Jakob Nordström, Shuo Pang, and Kilian Risse. Graph colouring is hard on average for polynomial calculus and Nullstellensatz. In *Proceedings of the 64th Annual IEEE Symposium on Foundations of Computer Science (FOCS '23)*, pages 1–11, November 2023.

19 William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

20 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, August 2017.

21 Luís Cruz-Filipe, João P. Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 118–135. Springer, April 2017.

22 Emir Demirović, Ciaran McCreesh, Matthew McIlree, Jakob Nordström, Andy Oertel, and Konstantin Sidorov. Pseudo-Boolean reasoning about states and transitions to certify dynamic programming and decision diagram algorithms. In *Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming (CP '24)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:21, September 2024.

23 Simon Dold, Malte Helmert, Jakob Nordström, Gabriele Röger, and Tanja Schindler. Pseudo-Boolean proof logging for optimal classical planning. In *Proceedings of the 35th International Conference on Automated Planning and Scheduling (ICAPS '25)*, pages 54–63, November 2025.

24 Nicholas Downing, Thibaut Feydy, and Peter J. Stuckey. Explaining alldifferent. In *Proceedings of the 35th Australasian Computer Science Conference (ACSC '12)*, pages 115–124, January 2012.

25 Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, 197(2):793–812, February 2023.

26 Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 2020.

27 Salomé Eriksson, Gabriele Röger, and Malte Helmert. Unsolvability certificates for classical planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS '17)*, pages 88–97, June 2017.

28 Salomé Eriksson, Gabriele Röger, and Malte Helmert. A proof system for unsolvable planning tasks. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS '18)*, pages 65–73, June 2018.

29 Daniel Faber, Adalat Jabrayilov, and Petra Mutzel. SAT Encoding of Partial Ordering Models for Graph Coloring Problems. In *Proceedings of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT '24)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:20, August 2024.

30 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. IPASIR-UP: User Propagators for CDCL. In *Proceedings of the 26th International Conference on Theory and Applications of Satisfiability Testing (SAT '23)*, volume 271 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:13, July 2023.

31 Mathias Fleury. *Formalization of Logical Calculi in Isabelle/HOL*. PhD thesis, Universität des Saarlandes, 2020. Available at `https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/28722`.

32 Fabio Furini, Virginie Gabrel, and Ian-Christopher Ternier. An improved DSATUR-based branch-and-bound algorithm for the vertex coloring problem. *Networks*, 69(1):124–141, 2017.

33 Graeme Gange, Geoffrey Chu, and Peter J. Stuckey. Certifying optimality in constraint programming. Manuscript. Available at `https://people.eng.unimelb.edu.au/pstuckey/papers/certified-cp.pdf`, 2023.

34 Assefaw Hadish Gebremedhin, Fredrik Manne, and Alex Pothen. What color is your Jacobian? graph coloring for computing derivatives. *SIAM Review*, 47(4):629–705, 2005. `doi:10.1137/S0036144504444711`.

35 Xavier Gillard, Pierre Schaus, and Yves Deville. SolverCheck: Declarative testing of constraints. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 565–582. Springer, October 2019.

36 Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:25, August 2022.

37 Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

38 Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. End-to-end verification for subgraph solving. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI '24)*, pages 8038–8047, February 2024.

39 Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1134–1140, July 2020.

40 Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP '22)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, August 2022.

41 Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

42 Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, pages 886–891, March 2003.

43 Justin S. Gray, Tristan A. Hearn, and Bret A. Naylor. Using graph coloring to compute total derivatives more efficiently in OpenMDAO. In *AIAA Aviation 2019 Forum*, pages 3108:1–3108:15, June 2019.

44 Stefano Gualandi and Federico Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.

45 Emmanuel Hebrard and George Katsirelos. Constraint and satisfiability reasoning for graph coloring. *Journal of Artificial Intelligence Research*, 69:33–65, 2020. URL: `https://doi.org/10.1613/jair.1.11313`, `doi:10.1613/JAIR.1.11313`.

46 Stephan Held, William Cook, and Edward C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012.

47 Marijn J. H. Heule, Benjamin Kiesl, Martina Seidl, and Armin Biere. PRuning through satisfaction. In *13th International Haifa Verification Conference (HVC '17)*, volume 10629 of *Lecture Notes in Computer Science*, pages 179–194. Springer, November 2017.

48 Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified MaxSAT preprocessing. In *Proceedings of the 12th International Joint Conference on Automated Reasoning (IJCAR '24)*, volume 14739 of *Lecture Notes in Computer Science*, pages 396–418. Springer, July 2024.

49 Adalat Jabrayilov and Petra Mutzel. New integer linear programming models for the vertex coloring problem. In Michael A. Bender, Martín Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics*, pages 640–652. Springer International Publishing, 2018.

50 David S. Johnson and Michael A. Trick. Introduction to the second DIMACS challenge: Cliques, coloring, and satisfiability. In *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–10. American Mathematical Society, 1996.

51 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer, 1972.

52 Daniela Kaufmann and Armin Biere. Fuzzing and delta debugging and-inverter graph verification tools. In *Proceedings of the 16th International Conference on Tests and Proofs (TAP '22)*, volume 13361 of *Lecture Notes in Computer Science*, pages 69–88. Springer, July 2022.

53 Wietze Koops, Daniel Le Berre, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Yong Kiam Tan, and Marc Vinyals. Practically feasible proof logging for pseudo-Boolean optimization. In *Proceedings of the 31st International Conference on Principles and Practice of Constraint Programming (CP '25)*, volume 340 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:27, August 2025.

54 Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 1396–1402, August 2021.

55 Peter Lammich. Efficient verified (UN)SAT certificate checking. *Journal of Automated Reasoning*, 64(3):513–532, March 2020. Extended version of paper in *CADE* 2017.

56 Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A framework for certified Boolean branch-and-bound optimization. *Journal of Automated Reasoning*, 46(1):81–102, January 2011.

57 Vahid Lotfi and Sanjiv Sarin. A graph coloring algorithm for large scale scheduling problems. *Computers & Operations Research*, 13(1):27–32, 1986.

58 Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

59 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, May 2011.

60 Ciaran McCreesh, William Pettersson, and Patrick Prosser. Understanding the empirical hardness of random optimisation problems. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 333–349. Springer, September 2019.

61 Colin McDiarmid. Colouring random graphs. *Annals of Operations Research*, 1(3):183–200, October 1984.

62 Matthew McIlree and Ciaran McCreesh. Proof logging for smart extensional constraints. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:17, August 2023.

63 Matthew McIlree and Ciaran McCreesh. Certifying bounds propagation for integer multiplication constraints. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI '25)*, pages 11309–11317, February-March 2025.

64 Matthew McIlree, Ciaran McCreesh, and Jakob Nordström. Proof logging for the circuit constraint. In *Proceedings of the 21st International Conference on the Integration of Constraint*

Programming, Artificial Intelligence, and Operations Research (CPAIOR '24), volume 14743 of *Lecture Notes in Computer Science*, pages 38–55. Springer, May 2024.

65 Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996. doi:10.1287/ijoc.8.4.344.

66 António Morgado and João P. Marques-Silva. On validating Boolean optimizers. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI '12)*, pages 924–926, November 2011.

67 Esther Mugdan, Remo Christen, and Salomé Eriksson. Optimality certificates for classical planning. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS '23)*, pages 286–294, July 2023.

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

69 Aina Niemetz, Mathias Preiner, and Clark W. Barrett. Murxla: A modular and highly extensible API fuzzer for SMT solvers. In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV '22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 92–106. Springer, August 2022.

70 Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, January 2009.

71 Tobias Paxian and Armin Biere. MaxSAT fuzzing and delta debugging. *Journal of Artificial Intelligence Research*, 85, February 2026.

72 Aaron Potechin and Jeff Xu. Sum-of-squares lower bounds for coloring random graphs. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25)*, pages 84–95, June 2025.

73 Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Towards bridging the gap between SAT and Max-SAT refutations. In *Proceedings of the 32nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI '20)*, pages 137–144, November 2020.

74 Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Proofs and certificates for Max-SAT. *Journal of Artificial Intelligence Research*, 75:1373–1400, December 2022.

75 Konrad Slind and Michael Norrish. A brief overview of HOL4. In *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs '08)*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer, August 2008.

76 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation redundancy checking in CakeML. In *Proceedings of the 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '21)*, volume 12652 of *Lecture Notes in Computer Science*, pages 223–241. Springer, March-April 2021.

77 Yong Kiam Tan, Magnus O. Myreen, Ramana Kumar, Anthony C. J. Fox, Scott Owens, and Michael Norrish. The verified CakeML compiler backend. *Journal of Functional Programming*, 29:e2:1–e2:57, February 2019.

78 Cesare Tinelli. Scalable proof production and checking in SMT. In *Proceedings of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT '24)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:2, August 2024.

79 Raoul Van Doren, Timos Antonopoulos, and Ruzica Piskac. Efficient and verifiable proof logging for MaxSAT solving. In *Proceedings of the 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE '25)*, pages 1325–1337, November 2025.

80 Michael Veksler and Ofer Strichman. A proof-producing CSP solver. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI '10)*, pages 204–209, July 2010.

81 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, July 2014.

82 Emre Yolcu, Xinyu Wu, and Marijn J. H. Heule. Mycielski graphs and PR proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 201–217. Springer, 2020.

83 Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, pages 880–885, March 2003.

84 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, August 2007. Preliminary version in *STOC '06*.

85 Alexander Aleksandrovich Zykov. On some properties of linear complexes. *Matematicheskii Sbornik*, 66(2):163–188, 1949.

A Detailed Description of Proof Logging for Mycielski Graphs

In this appendix, we describe how lower bounds for the chromatic number of Mycielski graphs can be formalized as pseudo-Boolean proofs that can be verified by the VeriPB proof checker.

A.1 Taking Subgraphs Can Only Decrease Chromatic Number

We start with a simple fact that we use when claiming that the chromatic number of a Mycielski graph is a lower bound for the full graph in which we have detected the Mycielski graph, namely that going to a subgraph can never increase the chromatic number.

▶ **Fact 5** (Chromatic number of subgraph). *Let H be a subgraph of G. Then $\chi(G) \geq \chi(H)$.*

We formalize this claim using a VeriPB proof in three steps. In order to express $\chi(H)$, we first introduce colour usage variables u_b^H for H, so that $\sum_{b\in[n]} u_b^H$ counts the number of colours used for H. Then we show that each colour used in the subgraph H is also used in the supergraph G. In more detail, the formal derivation goes as follows.

1. Introduce colour usage variables for H using reification, i.e.,

$$J_b^{1.1} \doteq \quad |V(H)|u_b^H + \sum_{v\in V(H)} \overline{x_{v,b}} \geq |V(H)| \qquad \text{for each } b \in [n], \tag{16}$$

$$J_b^{1.2} \doteq \quad \overline{u_b^H} + \sum_{v\in V(H)} x_{v,b} \geq 1 \qquad \text{for each } b \in [n]. \tag{17}$$

2. Relate the colour usage in G to H with

$$J_b^2 \doteq \quad \overline{u_b^H} + u_b \geq 1 \qquad \text{for each } b \in [n]\,, \tag{18}$$

   which is inferred by adding $J_b^{1.2}$ to A_c^3, weakening $x_{v,b}$ for $v \notin V(H)$ and saturating.
3. Derive

$$J^3 \doteq \quad \sum_{b\in[n]} u_b \geq \sum_{b\in[n]} u_b^H. \tag{19}$$

   by adding constraints J_b^2 for all $b \in [n]$.

A.2 Purifying Mycielski Graphs Can Only Decrease Chromatic Number

Next, we provide more details on how we show using the VeriPB proof system that the chromatic number of a Mycielski graph M is at least the chromatic number of the purified Mycielski graph $\widehat{M}$.

▶ **Proposition 3.** *For a Mycielski graph M and a pure Mycielski graph $\widehat{M}$ with the same base graph B, it holds that $\chi(M) \geq \chi(\widehat{M})$.*

To prove this proposition, let us write $B = (V_B, E_B)$ to denote the base graph and $\widehat{M} = \left(V_{\widehat{M}}, E_{\widehat{M}}\right)$ for the purified Mycelski graph with

$$V_{\widehat{M}} = V_B \cup \{v' \mid v \in V_B\} \cup \{\widehat{w}\} \tag{20a}$$

and

$$E_{\widehat{M}} = E_B \cup \{\{u, v'\} \mid u, v \in V_B, \{u, v\} \in E_B\} \cup \{\{v', \widehat{w}\} \mid v \in V_B\}\,, \tag{20b}$$

and let s and w denote the shadow function and top vertex of M. For any colouring f of M, we can define a colouring $\widehat{f}$ of $\widehat{M}$ by letting $\widehat{f}(v) = f(v)$ for $v \in V_B$, $\widehat{f}(v') = f(s(v))$ for $v \in V_B$, and $\widehat{f}(\widehat{w}) = f(w)$. By the construction of $\widehat{M}$, we see that this is a proper colouring.

To formalize this in the VeriPB proof system, consider the witness substitution

$$\omega_{V_{\widehat{M}} \mapsto V_M} = \{\widehat{x}_{\widehat{w},c} \mapsto x_{w,c} \mid c \in [n]\} \cup \{\widehat{x}_{v,c} \mapsto x_{v,c} \mid v \in V_B, c \in [n]\} \\ \cup \{\widehat{x}_{v',c} \mapsto x_{s(v),c} \mid v \in V_B, c \in [n]\} \cup \{\widehat{u}_c \mapsto u_c \mid c \in [n]\}\,. \tag{21}$$

This substitution provides the same mapping as we used before to define a colouring $\widehat{f}$ of $\widehat{M}$ based on a colouring f of M. By the construction of $\widehat{M}$, this substitution maps every constraint used to define the chromatic number of $\widehat{M}$ to a constraint used to define the chromatic number of M, which has already been derived by assumption. Hence, we could derive all required constraints using redundance-based strengthening with this substitution.

However, deriving all constraints in this way is too expensive, since it requires writing the long witness substitution in (21) to file for every derived constraint. We therefore optimize the proof in two ways. The first observation is that as long as a literal only occurs with one polarity, i.e., only as either an unnegated or a negated variable, we can always without loss of generality set that literal to true. Hence, for all edge constraints it suffices to map one of the variables $\widehat{x}_{u,c}$ in that constraint to false. Intuitively, this says that we can make the edge constraints hold by simply not assigning any colours at all. Similarly, since literals appear negated in the at-most-one-colour constraints (when these are written as a greater-than-or-equal constraints), these also can be derived using a substitution setting all variables occuring in the derived constraint to false.

For the colour constraints, we would need to use the long substitution $\omega_{V_{\widehat{M}} \mapsto V_M}$ in (21) for each constraint. We instead infer all these constraints as one-way reified constraints with a common activation variable a that trivializes the constraints when set to false. In this way, all constraints can be inferred by strengthening with a witness mapping a to false. Finally, infer that a can be fixed to true without loss of generality using the substitution $\omega_{V_{\widehat{M}} \mapsto V_M}$, and then remove the activation variables from the constraints by adding $a \geq 1$. In this way, the use of an activation variable means that we only need to employ the cumbersome witness substitution $\omega_{V_{\widehat{M}} \mapsto V_M}$ a single time instead of for every vertex.

In more detail, we derive the required constraints in the following way:

1. Introduce the edge constraints

$$\widehat{A}^5_{u,v,c} \doteq \qquad \overline{\widehat{x}_{u,c}} + \overline{\widehat{x}_{v,c}} \geq 1 \qquad \text{for each } \{u,v\} \in E_{\widehat{M}},\ c \in [n] \tag{22}$$

   using strengthening with witness substitution $\{\widehat{x}_{u,c} \mapsto 0\}$. These constraints are analogues of $A^5_{u,v,c}$.
2. Infer the at-most-one-colour constraints

$$\widehat{A}^2_v \doteq \qquad \sum_{c \in [n]} \widehat{x}_{v,c} \leq 1 \qquad \text{for each } v \in V_{\widehat{M}} \tag{23}$$

   using strengthening with substitution $\{\widehat{x}_{v,c} \mapsto 0 \mid c \in [n]\}$. These constraints are analogues of A^2_v.
3. Derive the colour constraints reified with activation variable a as

$$W^3_v \doteq \qquad \overline{a} + \sum_{c \in [n]} \widehat{x}_{v,c} \geq 1 \qquad \text{for each } v \in V_{\widehat{M}} \tag{24}$$

   with substitution $\{a \mapsto 0\}$.

4. Define colour usage variables $\widehat{u}_c$ for $\widehat{M}$ using one-way reifications

$$\widehat{A}^3_c \doteq \qquad |V_{\widehat{M}}|\widehat{u}_c + \sum_{v \in V_{\widehat{M}}} \overline{\widehat{x}_{v,c}} \geq |V_{\widehat{M}}| \qquad \text{for each } c \in [n] \tag{25}$$

in analogy with constraints A^3_c.

5. Introduce the conditional relation of the colour usage in M to $\widehat{M}$ with

$$W^5 \doteq \qquad n \cdot \overline{a} + \sum_{c \in [n]} u_c \geq \sum_{c \in [n]} \widehat{u}_c \tag{26}$$

using strengthening with substitution $\{a \mapsto 0\}$.

6. Use strengthening to obtain the constraint

$$W^6 \doteq \qquad a \geq 1 \tag{27}$$

with witness substitution $\{a \mapsto 1\} \cup \omega_{V_{\widehat{M}} \mapsto V_M}$.

7. Derive

$$\widehat{A}^1_v \doteq \qquad \sum_{c \in [n]} \widehat{x}_{v,c} \geq 1 \qquad \text{for each } v \in V_{\widehat{M}} \tag{28}$$

by adding W^6 to W^3_v. These constraints are similar to A^1_v.

8. Derive

$$W^8 \doteq \qquad \sum_{c \in [n]} u_c \geq \sum_{c \in [n]} \widehat{u}_c \tag{29}$$

by adding W^6 to W^5.

9. Delete the constraints W^5 and W^6 as well as W^3_v for all $v \in V_{\widehat{M}}$.

A.3 Chromatic Number of Pure Mycielski Graphs

Finally, let us give more details on how the proof of the inequality $\chi(\widehat{M}) \geq \chi(B) + 1$ in Proposition 4 can be written down as a formal VeriPB derivation. For this particular derivation, the VeriPB proof checker can automatically infer (or "autoprove") all constraints for which the redundance-based strengthening rule (1) requires subproofs, referred to as the *proof goals* for a concrete application of the rule. However, spelling out explicitly what such subproofs for proof goals look like can make the checker run much faster.

Let us review the VeriPB derivation steps:

1. Introduce constraints

$$K^1_{v,b} \doteq \qquad \overline{\widehat{x}_{v,b}} + \widehat{x}_{v',b} + \widehat{x}_{\widehat{w},b} \geq 1 \qquad \text{for each } v \in V_B,\ b \in [n]\,, \tag{30}$$

using strengthening with substitution $\{\widehat{x}_{v',c} \mapsto \widehat{x}_{v,c} \mid c \in [n]\}$.

The proof goals required by the strengthening rule (1) are as follows:

- For each colour $c \in [n]$, the constraint $\widehat{A}^5_{v',\widehat{w},c} \doteq \overline{\widehat{x}_{v',c}} + \overline{\widehat{x}_{\widehat{w},c}} \geq 1$ is mapped to $\overline{\widehat{x}_{v,c}} + \overline{\widehat{x}_{\widehat{w},c}} \geq 1$. For $c = b$, this is implied by the negation of the introduced constraint since $\neg K^1_{v,b}$ implies $\widehat{x}_{\widehat{w},b} = 0$. For $c \neq b$, this follows by RUP as the negated proofgoal propagates $\widehat{x}_{v,c}$ and $\widehat{x}_{\widehat{w},c}$, after which $\widehat{A}^2_v$ propagates $\overline{\widehat{x}_{v,b}}$ so that $\neg K^1_{v,b}$ is falsified.

- For each colour $c \in [n]$ constraint $\widehat{A}^3_c$ is mapped to

$$|V_{\widehat{M}}|\widehat{u}_c + \overline{\widehat{x}_{v,c}} + \sum_{u \in V_{\widehat{M}} \setminus \{v'\}} \overline{\widehat{x}_{u,c}} \geq |V_{\widehat{M}}|\,. \tag{31}$$

  This constraint follows by RUP as the negation of the proofgoal propagates $\overline{\widehat{u}_c}$, then $\widehat{A}^3_c$ propagates $\overline{\widehat{x}_{u,c}}$ for all $u \in V_{\widehat{M}}$, as a result of which the negated proofgoal is falsified.
- All other constraints are either untouched by the witness substitution (and so constitute their own proof) or else are mapped to trivial constraints or to constraints already derived.

2. Introduce constraints

$$K^2_{v,b} \doteq \qquad \overline{\widehat{x}_{v,b}} + \widehat{x}_{v',b} \geq 1 \qquad \text{for each } v \in V_B,\ b \in [n]\,, \tag{32}$$

using strengthening with substitution $\{\widehat{x}_{v,c} \mapsto \widehat{x}_{v',c} \mid c \in [n]\}$.
The proof goals are as follows:

- For each $u \in V_B$ such that $\{u, v\} \in E_B$ and each $c \in [n]$, the constraint $\widehat{A}^5_{v,u',c}$ is mapped to $\overline{\widehat{x}_{v',c}} + \overline{\widehat{x}_{u',c}} \geq 1$. For $c = b$, this is implied by the negation of the introduced constraint since $\neg K^2_{v,b}$ implies $x_{v',b} = 0$. For $c \neq b$, this follows by RUP:
  - The negation of the proofgoal propagates $\widehat{x}_{u',c}$ and $\widehat{x}_{v',c}$.
  - The edge constraint $\widehat{A}^5_{u,v',c}$ propagates $\overline{\widehat{x}_{u,c}}$.
  - $\neg K^2_{v,b}$ propagates $\widehat{x}_{v,b}$ and $\overline{\widehat{x}_{v',b}}$.
  - $K^1_{v,b}$ propagates $\widehat{x}_{\widehat{w},b}$.
  - The at-most-one-colour constraint $\widehat{A}^2_{\widehat{w}}$ propagates $\overline{\widehat{x}_{\widehat{w},d}}$ for each $d \in [n] \setminus \{b\}$.
  - The edge constraint $\widehat{A}^5_{u,v,b}$ propagates $\overline{\widehat{x}_{u,b}}$.
  - The at-most-one-colour constraint $\widehat{A}^2_{u'}$ propagates $\overline{\widehat{x}_{u',d}}$ for each $d \in [n] \setminus \{c\}$.
  - $K^1_{u,d}$ propagates $\overline{\widehat{x}_{u,d}}$ for each $d \in [n] \setminus \{c, b\}$.
  - Now the at-least-one-colour constraint $\widehat{A}^1_u$ is falsified.
- For each $c \in [n]$ the constraint $\widehat{A}^3_c$ is mapped to

$$|V_{\widehat{M}}|\widehat{u}_c + \overline{\widehat{x}_{v',c}} + \sum_{u \in V_{\widehat{M}} \setminus \{v\}} \overline{\widehat{x}_{u,c}} \geq |V_{\widehat{M}}|\,. \tag{33}$$

  This constraint follows by RUP as the negation of the proofgoal propagates $\overline{\widehat{u}_c}$, then $\widehat{A}^3_c$ propagates $\overline{\widehat{x}_{u,c}}$ for all $u \in V_{\widehat{M}}$, then the negated proofgoal is falsified.
- All other constraints are either untouched or else mapped to trivial constraints or to constraints already derived.

3. Introduce colour usage variables for the base graph using reification, i.e.,

$$K^{3.1}_b \doteq \qquad |V_B|\widehat{u}^B_b + \sum_{v \in V_B} \overline{\widehat{x}_{v,b}} \geq |V_B| \qquad \text{for each } b \in [n], \tag{34}$$

$$K^{3.2}_b \doteq \qquad \overline{\widehat{u}^B_b} + \sum_{v \in V_B} \widehat{x}_{v,b} \geq 1 \qquad \text{for each } b \in [n]. \tag{35}$$

4. Derive

$$K^4_b \doteq \qquad \overline{\widehat{x}_{\widehat{w},b}} + \overline{\widehat{u}^B_b} \geq 1 \qquad \text{for all } b \in [n] \tag{36}$$

by RUP, which works since first $\neg K^4_b$ propagates $\widehat{x}_{\widehat{w},b}$ and $\widehat{u}^B_b$, then for each $v \in V_B$ the constraint $\widehat{A}^5_{v',\widehat{w},b}$ propagates $\overline{\widehat{x}_{v',b}}$, following which for each $v \in V_B$ the constraint $K^2_{v,b}$ propagates $\overline{\widehat{x}_{v,b}}$, which leads to the constraint $K^{3.2}_b$ being falsified.

5. Infer

$$K_b^5 \doteq \qquad \widehat{u}_b + \overline{\widehat{u}_b^B} + \overline{\widehat{x}_{\widehat{w},b}} \geq 2 \qquad \text{for each } b \in [n] \tag{37}$$

by the following sequence of derivation steps:
   - Add $K_{v,b}^2$ to $\widehat{A}_b^3$ for each $v \in V_B$ to get $|V_{\widehat{M}}|\widehat{u}_b + \overline{\widehat{x}_{\widehat{w},b}} + \sum_{u \in V_B} 2\overline{\widehat{x}_{u,b}} \geq |V_{\widehat{M}}|$.
   - Next, dividing by 2 yields $(|V_B|+1)\widehat{u}_b + \overline{\widehat{x}_{\widehat{w},b}} + \sum_{u \in V_B} \overline{\widehat{x}_{u,b}} \geq |V_B| + 1$, since $|V_{\widehat{M}}| = 2 \cdot |V_B| + 1$.
   - Add to this $K_b^{3.2}$ to obtain $(|V_B|+1)\widehat{u}_b + \overline{\widehat{x}_{\widehat{w},b}} + \overline{\widehat{u}_b^B} \geq 2$, after which saturation yields $2\widehat{u}_b + \overline{\widehat{x}_{\widehat{w},b}} + \overline{\widehat{u}_b^B} \geq 2$.
   - Add to this K_b^4 to get $2\widehat{u}_b + 2\overline{\widehat{x}_{\widehat{w},b}} + 2\overline{\widehat{u}_b^B} \geq 3$.
   - Finally, dividing by 2 yields the required constraint.

6. Infer

$$K^6 \doteq \qquad \sum_{c \in [n]} \widehat{u}_c \geq \sum_{c \in [n]} \widehat{u}_c^B + 1 \tag{38}$$

by adding up $\widehat{A}_{\widehat{w}}^1$ with K_c^5 for each $c \in [n]$.

7. Delete the constraints $K_{v,b}^1$, $K_{v,b}^2$, K_b^4, and K_b^5 for each $v \in V_B,\ b \in [n]$.