The Packing Chromatic Number of the Infinite Square Grid is 15*

Bernardo Subercaseaux and Marijn J. H. Heule

Carnegie Mellon University, Pittsburgh, PA 15203, USA
{bsuberca,mheule}@cs.cmu.edu

Abstract. A packing k-coloring is a natural variation on the standard notion of graph k-coloring, where vertices are assigned numbers from $\{1, \ldots, k\}$, and any two vertices assigned a common color $c \in \{1, \ldots, k\}$ need to be at a distance greater than c (as opposed to 1, in standard graph colorings). Despite a sequence of incremental work, determining the packing chromatic number of the infinite square grid has remained an open problem since its introduction in 2002. We culminate the search by proving this number to be 15. We achieve this result by improving the best-known method for this problem by roughly two orders of magnitude. The most important technique to boost performance is a novel, surprisingly effective propositional encoding for packing colorings. Additionally, we developed an alternative symmetry breaking method. Since both new techniques are more complex than existing techniques for this problem, a verified approach is required to trust them. We include both techniques in a proof of unsatisfiability, reducing the trusted core to the correctness of the direct encoding.

Keywords: Packing coloring · SAT · Verification.

1 Introduction

Automated reasoning techniques have been successfully applied to a variety of coloring problems ranging from the classical computer-assisted proof of the *Four Color Theorem* [1], to progress on the *Hadwiger-Nelson problem* [21], or improving the bounds on Ramsey-like numbers [19]. This article contributes a new success story to the area: we show the *packing* chromatic number of the infinite square grid to be 15, thus solving via automated reasoning techniques a combinatorial problem that had remained elusive for over 20 years.

The notion of *packing coloring* was introduced in the seminal work of Goddard et al. [10], and since then more than 70 articles have studied it [3], establishing it as an active area of research. Let us consider the following definition.

Definition 1. *A packing k-coloring of a simple undirected graph $G = (V, E)$ is a function f from V to $\{1, \ldots, k\}$ such that for any two distinct vertices $u, v \in V$, and any color $c \in \{1, \ldots, k\}$, it holds that $f(u) = f(v) = c$ implies $d(u, v) > c$.*

* Both authors are supported by the U.S. National Science Foundation under grant CCF-2015445.

S. Sankaranarayanan and N. Sharygina (Eds.): TACAS 2023, LNCS 13993, pp. 389–406, 2023.
https://doi.org/10.1007/978-3-031-30823-9_20

Note that by changing the last condition to $d(u, v) > 1$ we recover the standard notion of coloring, thus making packing colorings a natural variation of them. Intuitively, in a packing coloring, *larger* colors forbid being reused in a larger region of the graph around them. Indeed, packing colorings were originally presented under the name of *broadcast coloring*, motivated by the problem of assigning broadcast frequencies to radio stations in a non-conflicting way [10], where two radio stations that are assigned the same frequency need to be at distance greater than some function of the power of their broadcast signals. Therefore, a large color represents a powerful broadcast signal at a given frequency, that cannot be reused anywhere else within a large radius around it, to avoid interference. Minimizing the number of colors assigned can thus be interpreted as minimizing the pollution of the radio spectrum. The literature has preferred the name *packing coloring* ever since [3].

Analogously to the case of standard colorings, we can naturally define the notion of *packing chromatic number*, and study its computation.

Definition 2. *Given a graph $G = (V, E)$, define its packing chromatic number $\chi_\rho(G)$ as the minimum value k such that G admits a packing k-coloring.*

Example 1. Consider the infinite graph with vertex set $\mathbb{Z}$ and with edges between consecutive integers, which we denote as $\mathbb{Z}^1$. A packing 3-coloring is illustrated in Figure 1. On the other hand, by examination one can observe that it is impossible to obtain a packing 2-coloring for $\mathbb{Z}^1$.

Fig. 1: Illustration of a packing 3-coloring for $\mathbb{Z}^1$.

While Example 1 shows that $\chi_\rho(\mathbb{Z}^1) = 3$, the question of computing $\chi_\rho(\mathbb{Z}^2)$, where $\mathbb{Z}^2$ is the graph with vertex set $\mathbb{Z} \times \mathbb{Z}$ and edges between orthogonally adjacent points (i.e., points whose ℓ_1 distance equals 1), has been open since the introduction of packing colorings by Goddard et al. [10]. On the other hand, it is known that $\chi_\rho(\mathbb{Z}^3) = \infty$ (again considering edges between points whose ℓ_1 distance equals 1) [9]. The problem of computing $3 \leq \chi_\rho(\mathbb{Z}^2) \leq \infty$ has received significant attention, and it is described as "*the most attractive [of the packing coloring problems over infinite graphs]*" by Brešar et al. [3]. We can now state our main theorem, providing a final answer to this problem.

Theorem 1. $\chi_\rho(\mathbb{Z}^2) = 15$.

An upper bound of 15 had already been proved by Martin et al. [18], who found a packing 15-coloring of a 72×72 grid that can be used for periodically tiling the entirety of $\mathbb{Z}^2$. Therefore, the main contribution of our work consists of proving that 14 colors are not enough for $\mathbb{Z}^2$. Table 1 presents a summary of the historical progress on computing $\chi_\rho(\mathbb{Z}^2)$. It is worth noting that amongst the computer-generated proofs (i.e., all since Soukal and Holub [22] in 2010), ours is the first one to be formally verified, see Section 4.

Table 1: Historical summary of the bounds known for $\chi_\rho(\mathbb{Z}^2)$.

Year	Citation	Approach	Lower bound	Upper bound
2002	Goddard et al. [10]	Manual	9	23
2002	Schwenk [20]	Unkown	9	22
2009	Fiala et al. [8]	Manual + Computer	10	23
2010	Soukal and Holub [22]	Simulated Annealing	10	17
2010	Ekstein et al. [7]	Brute Force Program	12	17
2015	Martin et al. [17]	SAT solver	13	16
2017	Martin et al. [18]	SAT solver	13	15
2022	Subercaseaux and Heule [23]	SAT solver	14	15
2022	**This article**	SAT solver	15	15

For any $k \geq 4$, the problem of determining whether a graph G admits a packing 4-coloring is known to be NP-hard [10], and thus we do not expect a polynomial time algorithm for computing $\chi_\rho(\cdot)$. This naturally motivates the use of satisfiability (SAT) solvers for studying the packing chromatic number of finite subgraphs of $\mathbb{Z}^2$. The rest of this article is thus devoted to proving Theorem 1 by using automated reasoning techniques, in a way that produces a proof that can be checked independently and that has been checked by verified software.

2 Background

We start by recapitulating the components used to obtain a lower bound of 14 in our previous work [23]. Naturally, in order to prove a lower bound for $\mathbb{Z}^2$ one needs to prove a lower bound for a finite subgraph of it. As in earlier work, we consider *disks* (i.e., 2-dimensional balls in the ℓ_1-metric) as the finite subgraphs to study [23]. Concretely, let $D_r(v)$ be the subgraph induced by $\{u \in V(\mathbb{Z}^2) \mid d(u,v) \leq r\}$. To simplify notation, we use D_r as a shorthand for $D_r((0,0))$, and we let $D_{r,k}$ be the instance consisting of deciding whether D_r admits a packing k-coloring. Moreover, let $D_{r,k,c}$ be the instance $D_{r,k}$ but enforcing that the central vertex $(0,0)$ receives color c (Fig. 2).

For example, a simple lemma of Subercaseaux and Heule [23, Proposition 5] proves that the unsatisfiability of $D_{3,6,3}$ is enough to deduce that $\chi_\rho(\mathbb{Z}^2) \geq 7$. We will prove a slight variation of it (Lemma 2) later on in order to prove Theorem 1, but for now let us summarize how they proved that $D_{12,13,12}$ is unsatisfiable.

Encodings. The direct encoding for $D_{r,k,c}$ consists simply of variables $x_{v,t}$ stating that vertex v gets color t, as well as the following clauses:

1. (at-least-one-color clauses, ALOC) $\bigvee_{t=1}^{k} x_{v,t}, \quad \forall v \in V,$
2. (at-most-one-distance clauses, AMOD)

$$\overline{x_{u,t}} \vee \overline{x_{v,t}}, \quad \forall t \in \{1,\ldots,k\}, \forall u,v \in V \text{ s.t. } 0 < d(u,v) \leq t,$$

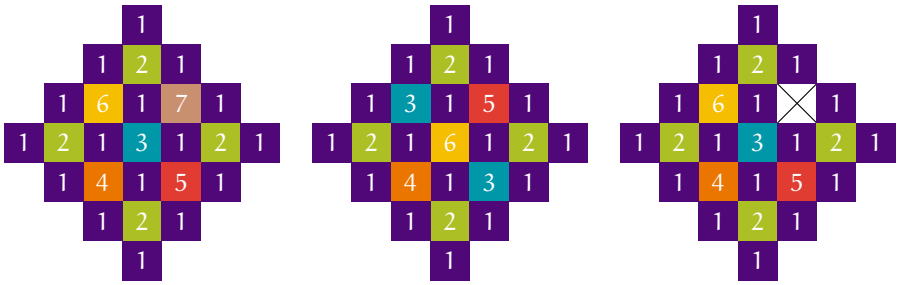


Fig. 2: Illustration of satisfying assignments for $D_{3,7,3}$ and $D_{3,6,6}$. On the other hand, $D_{3,6,3}$ is not satisfiable.

3. (center clause) $x_{(0,0),c}$.

This amounts to $O(r^2k^3)$ clauses [23]. The recursive encoding is significantly more involved, but it leads to only $O(r^2k \log k)$ clauses asymptotically. Unfortunately, the constant involved in the asymptotic expression is large, and this encoding did not give them practical speed-ups [23].

Cube And Conquer. Introduced by Heule et al. [13], the *Cube And Conquer*approach aims to *split* a SAT instance φ into multiple SAT instances $\varphi_1, \ldots, \varphi_m$ in such a way that φ is satisfiable if, and only if, at least one of the instances φ_i is satisfiable; thus allowing to work on the different instances φ_i in parallel. If $\psi = (c_1 \vee c_2 \vee \cdots \vee c_m)$ is a tautological DNF, then we have

$$\text{SAT}(\varphi) \iff \text{SAT}(\varphi \wedge \psi) \iff \text{SAT}\left(\bigvee_{i=1}^{m} (\varphi \wedge c_i)\right) \iff \text{SAT}\left(\bigvee_{i=1}^{m} \varphi_i\right),$$

where the different $\varphi_i \coloneqq (\varphi \wedge c_i)$ are the instances resulting from the split.

Intuitively, each cube c_i represents a *case*, i.e., an assumption about a satisfying assignment to φ, and soundness comes from ψ being a tautology, which means that the split into cases is exhaustive. If the split is well designed, then each φ_i is a particular case that is substantially easier to solve than φ, and thus solving them all in parallel can give significant speed-ups, especially considering the sequential nature of CDCL, at the core of most solvers. Our previous work [23] proposed a concrete algorithm to generate a split, which already results in an almost linear speed-up, meaning that by using 128 cores, the performance gain is roughly a $\times 60$ factor.

Symmetry Breaking. The idea of *symmetry breaking* [6] consists of exploiting the symmetries that are present in SAT instances to speed-up computation. In particular, $D_{r,k,c}$ instances have 3 axes of symmetry (i.e., vertical, horizontal, and diagonal) which allowed for close to an 8-fold improvement in performance for proving $D_{12,13,12}$ to be unsatisfiable. The particular use of symmetry breaking in our previous approach [23] was happening at the *Cube And Conquer* level, where

out of the sub-instances $\varphi_i, \ldots, \varphi_m$ produced by the split, only a $1/8$-fraction of them had to be solved, as the rest were equivalent under isomorphism.

Verification. Arguably the biggest drawback of our previous approach proving a lower bound of 14 is that it lacked the capability of generating a computer-checkable proof. To claim a full solution to the 20-year-old problem of computing $\chi_\rho(\mathbb{Z}^2)$ that is accepted by the mathematics community, we deem paramount a fully verifiable proof that can be scrutinized independently.

The most commonly-used proofs for SAT problems are expressed in the DRAT clausal proof system [11]. A DRAT proof of unsatisfiability is a list of clause addition and clause deletion steps. Formally, a clausal proof is a list of pairs $\langle \mathsf{s}_1, C_1 \rangle, \ldots, \langle \mathsf{s}_m, C_m \rangle$, where for each $i \in 1, \ldots, m$, $s_i \in \{\mathsf{a}, \mathsf{d}\}$ and C_i is a clause. If $s_i = \mathsf{a}$, the pair is called an *addition*, and if $s_i = \mathsf{d}$, it is called a *deletion*. For a given input formula φ_0, a clausal proof gives rise to a set of *accumulated formulas* φ_i ($i \in \{1, \ldots, m\}$) as follows:

$$\varphi_i = \begin{cases} \varphi_{i-1} \cup \{C_i\} & \text{if } \mathsf{s}_i = \mathsf{a} \\ \varphi_{i-1} \setminus \{C_i\} & \text{if } \mathsf{s}_i = \mathsf{d} \end{cases}$$

Each clause addition must preserve satisfiability, which is usually guaranteed by requiring the added clauses to fulfill some efficiently decidable syntactic criterion. The main purpose of deletions is to speed up proof checking by keeping the accumulated formula small. A valid proof of unsatisfiability must end with the addition of the empty clause.

3 Optimizations

Even with the best choice of parameters for our previous approach, solving the instance $D_{12,13,12}$ takes almost two days of computation with a 128-core machine [23]. In order to prove Theorem 1, we will require to solve an instance roughly 100 times harder, and thus several optimizations will be needed. In fact, we improve on all aspects discussed in Section 2; we present five different forms of optimization that are key to the success of our approach, which we summarize next.

1. We present a new encoding, which we call the plus *encoding* that has conceptual similarities with the recursive encoding of Subercaseaux and Heule [23], while achieving a significant gain in practical efficiency.
2. We present a new split algorithm that works substantially better than the previous split algorithm when coupled with the plus encoding.
3. We improve on symmetry breaking by using multiple layers of symmetry-breaking clauses in a way that exploits the design of the split algorithm to increase performance.
4. We study the choice of color to fix at the center, showing that one can gain significantly in performance by making instance-based choices; for example, $D_{12,13,6}$ can be solved more than three times as fast as $D_{12,13,12}$ (the instance used by Subercaseaux and Heule [23]).

5. We introduce a new and extremely simple kind of clauses called ALOD clauses, which improve performance when added to the other clauses of any encoding we have tested.

The following subsections present each of these components in detail.

3.1 "*Plus*": a New Encoding

Despite the asymptotic improvement of the recursive encoding of Subercaseaux and Heule [23], its contribution is mostly of "theoretical interest" as it does not improve solution times. Nonetheless, that encoding suggests the possibility of finding one that is both more succinct than the direct encoding and that speed-ups computation. Our path towards such an encoding starts with *Bounded Variable Addition (BVA)* [16], a technique to automatically re-encode CNF formulas by adding new variables, with the goal of minimizing their resulting size (measured as the sum of the number of variables and the number of clauses). BVA can significantly reduce the size of $D_{r,k,c}$ instances, even further than the recursive encoding. Moreover, BVA actually speeds-up computation when solving the resulting instances with a CDCL solver, see Table 2. Figure 3 compares the number of AMOD clauses between the direct encoding and the BVA encoding; for example in the direct encoding, for D_{14} color 10 would require roughly 30000 clauses, whereas it requires roughly 3500 in the BVA encoding. It can be observed as well in Figure 3 that the direct encoding grows in a very structured and predictable way, where color c in D_r requires roughly r^2c^2 clauses. On the other hand, arguably because of its locally greedy nature, the results for BVA are far more erratic, and roughly follow a $4r^2 \lg c$ curve.

The encoding resulting from BVA does not perform particularly well when coupled with the split algorithm of Subercaseaux and Heule. Indeed, Table 2 shows that while BVA heavily improves runtime under sequential CDCL, it does not provide a meaningful advantage when using *Cube And Conquer*. Furthermore, encodings resulting from BVA are hardly interpretable, as BVA uses

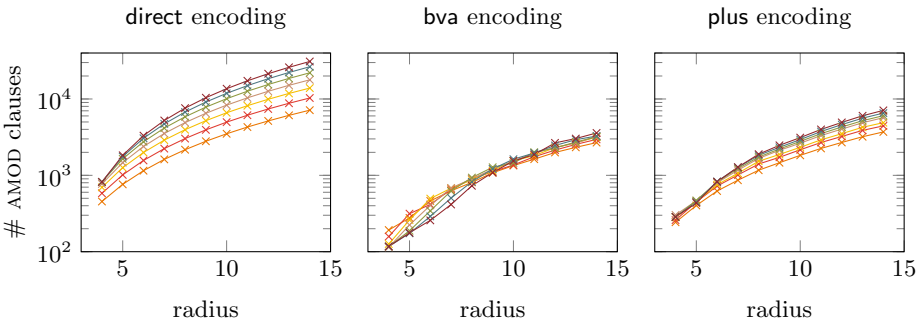


Fig. 3: Comparison of the size of the at-most-one-color clauses between the direct encoding and the BVA-encoding, for D_4 up to D_{14} and colors $\{4, \ldots, 10\}$.

Table 2: Comparison between the different encodings. *Cube And Conquer* experiments were performed with the approach of Subercaseaux and Heule [23] (parameters $F = 5, d = 2$) on a 128-core machine. Hardware details in Section 5.

	direct encoding		bva encoding		plus encoding	
	$D_{5,10,5}$	$D_{6,11,6}$	$D_{5,10,5}$	$D_{6,11,6}$	$D_{5,10,5}$	$D_{6,11,6}$
Number of variables	610	935	973	1559	673	1039
Number of clauses	10688	21086	2313	3928	4063	7548
CDCL runtime (s)	255.12	10774.79	39.88	2539.38	15.90	811.66
Cube-and-conquer wall-clock (s)	0.77	26.20	0.78	17.97	0.50	6.68

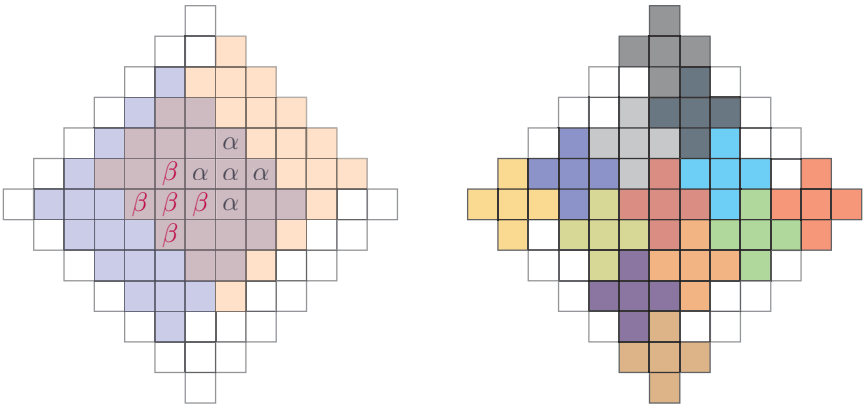
a locally greedy strategy for introducing new variables. As a result, the design of a split algorithm that could work well with BVA is a very complicated task. Therefore, our approach consisted of reverse engineering what BVA was doing over some example instances, and using that insight to design a new encoding that produces instances of size comparable to those generated by BVA while being easily interpretable and thus compatible with natural split algorithms.

By manually inspecting BVA encodings one can deduce that a fundamental part of their structure is what we call *regional variables/clauses*. A regional variable $r_{S,c}$ is associated with a set of vertices S and a color c, meaning that at least one vertex in S receives color c. Let us illustrate their use with an example.

Example 2. Consider the instance $D_{6,11}$, and let us focus on the *at-most-one-distance* (AMOD) clauses for color 4. Figure 4a depicts two regional clauses: one in orange (vertices labeled with α), and one in blue (vertices labeled with β), each consisting of 5 vertices organized in a *plus* (+) shape. We thus introduce variables $r_{\text{orange},4}$ and $r_{\text{blue},4}$, defined by the following clauses:

1. $\overline{r_{\text{orange},4}} \vee \bigvee_{v \text{ has label } \alpha} x_{v,4}$,
2. $\overline{r_{\text{blue},4}} \vee \bigvee_{v \text{ has label } \beta} x_{v,4}$,
3. $r_{\text{orange},4} \vee \overline{x_{v,4}}$, for each v with label α,
4. $r_{\text{blue},4} \vee \overline{x_{v,4}}$, for each v with label β.

The benefit of introducing these two new variables and $2 + (5 \cdot 2) = 12$ additional clauses will be shown now, when using them to forbid conflicts more compactly. Indeed, each vertex labeled with α or β participates in $|D_4| - 1 = 40$ AMOD clauses in the direct encoding, which equals a total of $10 \cdot 40 - \binom{10}{2} = 355$ clauses for all of them (subtracting the clauses counted twice). However, note that all 36 vertices shaded in light orange are at distance at most 4 from all vertices labeled with α, and thus they are in conflict with $r_{\text{orange},4}$. This means that we can encode all conflicts between α-vertices and orange-shaded vertices with 36 clauses. The same can be done for β-vertices and the 36 vertices shaded in light blue. Moreover, all pairs of vertices (x, y) with x being an α-vertex and y being a β-vertex are in conflict, which we can represent simply with the clause $(r_{\text{orange},4} \vee r_{\text{blue},4})$, instead of $5 \cdot 5 = 25$ pairwise clauses. We still need,



(a) Illustration of regions interacting in $P_{6,11,6}$, for color 4.

(b) Illustration of the placement of regions of the 13 regions in $P_{6,11,6}$.

Fig. 4: Illustrations for $P_{6,11,6}$.

however, to forbid that more than one α-vertex receives color 4, and the same for β-vertices, which can be done by simply adding all $2 \cdot \binom{5}{2} = 20$ AMOD clauses between all pairs. In total, the total number of clauses involving α or β vertices has gone down to $12+2\cdot 36+20+1 = 105$ clauses, from the original 355 clauses, by merely adding two new variables.

As shown in Example 2, the use of regional clauses can make encodings more compact, and this same idea scales even better for larger instances when the regions are larger. A key challenge for designing a *regional encoding* in this manner is that it requires a choice of regions (which can even be different for every color). After trying several different strategies for defining regions, we found one that works particularly well in practice (despite not yielding an optimal number for the metric #variables + #clauses), which we denote the plus *encoding*. The plus encoding is based on simply using "+" shaped regions (i.e., D_1) for all colors greater than 3, and to not introduce any changes for colors 1, 2 and 3 as they only amount to a very small fraction of the total size of the instances we consider. We denote with $P_{d,k,c}$ the plus encoding of the diamond of size d with k colors, and the centered being colored with c. Figure 4b illustrates $P_{6,11,6}$. Interestingly, the BVA encoding opted for larger regions for the larger colors, using for example D_2's or D_3's as regions for color 14. We have experimentally found this to be very ineffective when coupled with our split algorithms. In terms of the locations of the "+" shaped regions, we have placed them manually through an interactive program, arriving to the conclusion that the best choice of locations consists of packing as many regions as possible and as densely around the center as possible. A more formal presentation of all the clauses involved in the plus encoding is presented in the extended *arXiv* version [24] of this paper, but all its components have been illustrated in Example 2.

The exact number of clauses resulting from the plus encoding is hard to analyze precisely, but it is clear that asymptotically it only improves from the direct encoding by a constant multiplicative factor. Figure 3 and Table 2 illustrate the compactness of the plus encoding over particular instances, and its increase in efficiency both for CDCL solving as well as with the *Cube And Conquer* approach of Subercaseaux and Heule [23].

3.2 Symmetry Breaking

Another improvement of our approach is a static symmetry-breaking technique, while Subercaseaux and Heule [23] achieved symmetry breaking by discarding all but 1/8 of the cubes. We cannot do this easily since the plus encoding does not have an 8-fold symmetry. Instead it has a 4-fold symmetry (see Figure 4b). We add symmetry breaking clauses directly on top of the direct encoding (i.e., instead of using it after a *Cube And Conquer* split), as $D_{r,k,c}$ has indeed an 8-fold symmetry (see Figure 5b). Concretely, if we consider a color t, it can only appear once in the $D_{\lfloor t/2 \rfloor}$, as if it appeared more than once said appearances would be at distance $\leq t$. Given this, we can assume without loss of generality that if there is one appearance of t in $D_{\lfloor t/2 \rfloor}$, then it appears with coordinates (a, b) such that $a \geq 0 \wedge b \geq a$. We enforce this by adding negative units of the form $\overline{x_{(i,j),t}}$ for every pair $(i, j) \in D_{\lfloor t/2 \rfloor}$ such that $i < 0 \vee j < i$. This is illustrated in Figure 5b for $D_{5,10}$. Note however that this can only be applied to a single color t, as when a vertex in the *north-north-east* octant gets assigned color t, the 8-fold symmetry is broken. However, if the symmetry breaking clauses have been added for color t, and yet t does not appear in $D_{\lfloor t/2 \rfloor}$, then there is still an 8-fold symmetry in the encoding we can exploit by breaking symmetry on some other color t'. This way, our encoding uses $L = 5$ *layers* of symmetry breaking, for colors $k, k-1, \ldots, k-L+1$. At each layer i, where symmetry breaking is done over color $k - i$, except for the first (i.e., $i > 0$), we need to concatenate a clause

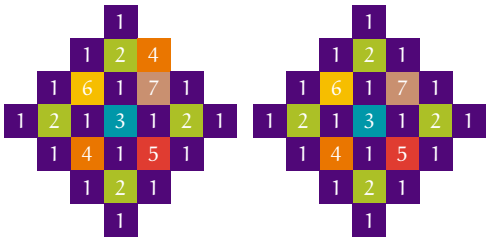
$$\text{SymmetryBroken}_i \coloneqq \bigvee_{t=k-i}^{k} \bigvee_{\substack{(a,b) \in D_{\lfloor t/2 \rfloor} \\ 0 \leq a \leq b}} x_{(a,b),t}$$

to each symmetry breaking clause, so that symmetry breaking is applied only when symmetry has not been broken already. Table 3 (page 14) illustrates the impact of this symmetry breaking approach, yielding close to a $\times 40$ speed-up for $D_{6,11,6}$.

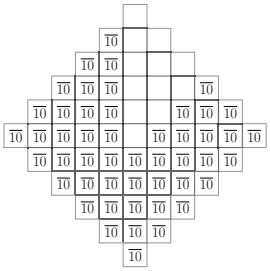
3.3 *At-Least-One-Distance* clauses

Yet another addition to our encoding is what we call *At-Least-One-Distance* (ALOD) clauses, which consist on stating that, for every vertex v, if we consider $D_1(v)$, then at least one vertex in $D_1(v)$ must get color 1. Concretely, the *At-Least-One-Distance* clause corresponding to a vertex $v = (i, j)$ is

$$C_v = x_{(i,j),1} \vee x_{(i+1,j),1} \vee x_{(i-1,j),1} \vee x_{(i,j+1),1} \vee x_{(i,j-1),1}.$$



(a) Illustration of the effect of adding ALOD clauses. The right figure, with ALOD clauses, presents a *chessboard pattern*.



(b) Some symmetry-breaking unit clauses added to $D_{5,10}$.

Fig. 5: The effect of adding ALOD clauses (left) and symmetry-breaking (right).

Note that adding these clauses preserves satisfiability since they are *blocked clauses* [15]; this can be seen as follows. If no vertex in $D_1(v)$ gets assigned color 1, then we can simply assign $x_{v,1}$, thus satisfying the new clause C_v.

The purpose of ALOD clauses can be described as *incentives* towards assigning color 1 in a *chessboard pattern* (see Figure 5a), which seems to simplify the rest of the computation. Empirically, their addition improves runtimes; see Table 3.

3.4 Cube And Conquer Using Auxiliary Variables

The split of Subercaseaux and Heule [23] is based on cases about the $x_{v,c}$ variables of the direct encoding, and specifically using vertices v that are close to the center and colors c that are in the top-t colors for some parameter t.

Our algorithm is instead based on cases only around the new regional variables $r_{S,c}$, which appears to be key for exploiting their use in the encoding.

More concretely, our algorithm, which we call PTR, is roughly based on splitting the instance into cases according to which out of the R regions that are closest to the center get which of the T highest colors (noting that a region can get multiple colors). A third parameter P indicates the maximum number of positive literals in any cube of the split. More precisely, there are cubes with i positive literals for $i \in \{0, 1, \ldots, P-1, P\}$, and the set of cubes with i positive literals is constructed by PTR as follows:

1. Let $\mathcal{R}$ be the set of R regions that are the closest to the center, and $\mathcal{T}$ the set consisting of the T highest colors (i.e., $\{k, k-1, \ldots, k-T+1\}$).
2. For each of the R^i tuples $\vec{S} \in \mathcal{R}^i$, we create $\binom{T}{i}$ cubes as described in the next step.
3. For each subset $Q \subseteq \mathcal{T}$ with size $|Q| = i$, let $q_1, \ldots, q_i$ be its elements in increasing order, and then create a cube with positive literals $r_{\vec{S}_j, q_j}$ for $j \in \{1, \ldots, i\}$. Then, if $i < P$, add to the cube negative literals $\overline{r_{\vec{S}_j, q_\ell}}$ for $j \in \{1, \ldots, i\}$ and every $q_\ell \notin Q$.

Lemma 1. *The cubes generated by the* PTR *algorithm form a tautology.*

The proof of Lemma 1 is quite simple, and we refer the reader to the proof of Lemma 7 in Subercaseaux and Heule [23] for a very similar one. Moreover, because our goal is to have a verifiable proof, instead of relying on Lemma 1, we test explicitly that the cubes generated by our algorithm form a tautology in all the instances mentioned in this paper. Pseudo-code for PTR is presented in the extended *arXiv* version of this paper [24].

3.5 Optimizing the Center Color

Our previous work [23] argued that for an instance $D_{r,k}$, one should fix the color of the central vertex to $\min(r, k)$. However, our experiments suggest otherwise. As the proof of Lemma 2 (in extended *arXiv* version [24]) implies, we are allowed to fix any color in the center, and as long as the resulting instance is unsatisfiable, that will allow us to establish the same lower bound. It turns out that the choice of the center color can dramatically affect performance, as shown for instance $D_{12,13}$ (the one used to prove $\chi_\rho(\mathbb{Z}^2) \geq 14$) in Figure 6. Interestingly, performance does not change monotonically with the value fixed in the center. Intuitively, it appears that fixing smaller colors in the center is ineffective as they impose restrictions on a small region around the center, while fixing very large colors in the center does not constrain the center much; for example, on the one hand, fixing a 1 or 2 in the center does not seem to impose any serious constraints on solutions. On the other hand, when a 12 is fixed in the center (as in our previous work [23]), color 6 can be used 5 times in D_6, whereas if color 6 is fixed in the center, it can only be used once in D_6. The apparent advantage of fixing 12 in the center (that it cannot occur anywhere else in $D_{12,13}$), is outweighed by the extra constraints around the center that fixing color 6 imposes; Subercaseaux and Heule already observed that most conflicts between colors occur around the center [23]), thus explaining why it makes sense to optimize in that area.

The main result of Subercaseaux and Heule [23] is the unsatisfiability of $D_{12,13,12}$, which required 45 CPU hours using the same SAT solver and similar hardware. Let $P^\star_{d,k,c}$denote $P_{d,k,c}$ with ALOD clauses and symmetry-breaking

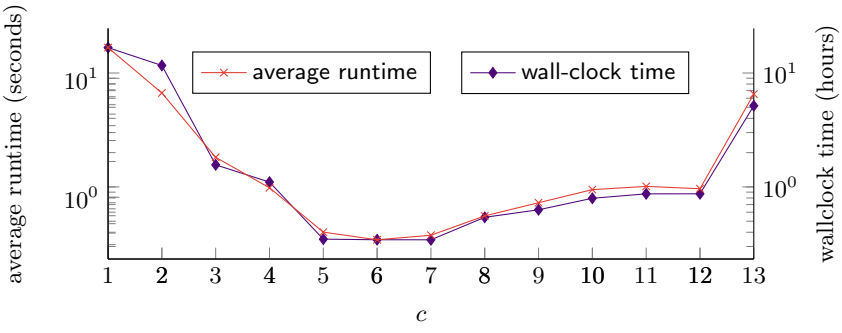


Fig. 6: The impact of the color in the center (c) on the performance for $P^\star_{12,13,c}$.

$$\overbrace{D_{15,14,6} \equiv \underbrace{D^{\star}_{15,14,6}}_{\text{re-encoding proof}}}^{\text{symmetry proof}} \equiv \overbrace{P^{\star}_{15,14,6} \vDash \underbrace{N_{15,14,6} \vDash \bot}_{\text{tautology proof}}}^{\text{implication proof}}$$

Fig. 7: Illustration of the verification pipeline.

predicates. We show unsatisfiability of $P^{\star}_{12,13,12}$ in 1.18 CPU hours and of $P^{\star}_{12,13,6}$ in 0.34 CPU hours. So the combination of the plus encoding and the improved center reduces the computational costs by two orders of magnitude.

4 Verification

Our pipeline proves that, in order to trust $\chi_\rho(\mathbb{Z}^2) = 15$ as a result, the only component that requires unverified trust is the direct encoding of $D_{15,14,6}$. Indeed, let $P^{\star}_{15,14,6}$ be the instance $P_{15,14,6}$ with ALOD-clauses and 5 layers of symmetry breaking clauses, and let $\psi = \{c_1, \ldots, c_m\}$ be the set of cubes generated by the PTR algorithm with parameters $P = 6, T = 7, R = 9$. We then prove:

1. that $D_{15,14,6}$ is satisfiability equivalent to $P^{\star}_{15,14,6}$.
2. the DNF $\psi = c_1 \vee c_2 \vee \cdots \vee c_m$ is a tautology.
3. each instance $(P^{\star}_{15,14,6} \wedge c_i)$, for $c_i \in \psi$ is unsatisfiable.
4. hence the negation of each cube is implied by $P^{\star}_{15,14,6}$.
5. since ψ is a tautology, its negation $N_{15,14,6}$ is unsatisfiable.

As a result, Theorem 1 relies only on our implementation of $D_{15,14,6}$. Fortunately, this is quite simple, and the whole implementation is presented in the extended *arXiv* version of this paper [24]. Figure 7 illustrates the verification pipeline, and the following paragraphs detail its different components.

Symmetry Proof. The first part of the proof consists in the addition of symmetry-breaking predicates to the formula. This part needs to go before the re-encoding proof, because the plus encoding does not have the 8-fold symmetry of the direct encoding. Each of the clauses in the symmetry-breaking predicates have the substitution redundancy (SR) property [5]. This is a very strong redundancy property and checking whether a clause C has SR w.r.t. a formula φ is NP-complete. However, since we know the symmetry, it is easy to compute a SR certificate. There exists no SR proof checker. Instead, we implemented a prototype tool to convert SR proofs into DRAT for which formally verified checkers exists. Our conversion is similar to the approach to converted propagation redundancy into DRAT [12]. The conversion can significantly increase the size of the proof, but the other proof parts are typically larger for harder formulas, thus the size is acceptable.

Re-encoding Proof. After symmetry breaking, the formula encoding is optimized by transforming the direct encoding into the plus encoding and adding the ALOD clauses. This part of the proof is easy. All clauses in the plus encoding and all ALOD clauses have the RAT redundancy property w.r.t. the direct encoding. This means that we can add all these clauses with a single addition step per clause. Afterward, the clauses that occur in the direct encoding but not in the plus encoding are removed using deletion steps.

Implication Proof. The third part of the proof expresses that the formula cannot be satisfied with any of the cubes from the split. For easy problems, one can avoid splitting and just use the empty cube as tautological DNF. For harder problems, splitting is crucial. We solve $D_{15,14,6}$ using a split with just over 5 million cubes. Using a SAT solver to show that the formula with a cube is unsatisfiable shows that the negative of the cube is implied by the formula. We can derive all these implied clauses in parallel. The proofs of unsatisfiability can be merged into a single implication proof.

Tautology Proof. The final proof part needs to show that the negation of the clauses derived in the prior steps form a tautology. In most cases, including ours, the cubes are constructed using a tree-based method. This makes the tautology check easy as there exists a resolution proof from the derived clauses to the empty clause using $m-1$ resolution steps with m denoting the number of cubes. This part can be generated using a simple SAT call.

The final proof merges all the proof parts. In case the proof parts are all in the DRAT format, such as our proof parts, then they can simply be merged by concatenating the proofs using the order presented above.

5 Experiments

Experimental Setup. In terms of hardware, all our experiments were run in the Bridges2 [4] supercomputer. Each node has the following specifications: Two AMD EPYC 7742 CPUs, each with 64 cores, 256MB of L3 cache, and 512GB total RAM memory. Our code and various formulas are publicly available at the repository `https://github.com/bsubercaseaux/PackingChromaticTacas`. In terms of software, all sequential experiments were run on state-of-the-art solver `CaDiCaL` [2], while parallel experiments with *Cube And Conquer* were run using a new implementation of parallel `iCaDiCaL` because it supports incremental solving [13] while being significantly faster than `iLingeling`.

Effectiveness of the Optimizations. We evaluated the optimizations to the direct encoding as proposed in Section 3: the plus encoding, the addition of the ALOD clauses, and the new symmetry breaking. The results are shown in Table 3. We picked $D_{6,11,6}$ for this evaluation since it is the largest diamond that can still be solved within a couple of hours on a single core.

The main conclusion is that the optimizations significantly improve the runtime. A comparison between the direct encoding without symmetry breaking and

the plus encoding with symmetry breaking and the ALOD clauses shows that the latter can be solved roughly 200x faster. Table 3 shows all 8 possible configurations. Turning on any of the optimizations always improves performance. The effectiveness of the plus encoding and ALOD clauses is somewhat surprising: the speed-up factor obtained by re-encoding typically does not exceed the factor by which the formula size is reduced. In this case, the reduction factor in formula size is less than 3, while the speed-up is larger than 13 (see the difference between the first and second row of Table 3). Moreover, we are not aware of the effectiveness of adding blocked clauses. Typically SAT solvers remove them.

We also constructed DRAT proofs of the optimizations (shown as derivation in the table) and the solver runtime. We merged them into a single DRAT proof by concatenating the files. The proofs were first checked with the `drat-trim` tool, which produced LRAT proofs. These LRAT files were validated using the formally-verified `cake-lpr` checker. The size of the DRAT proofs and the checking time are shown in the table. Note that the checking time for the proofs with symmetry breaking is always larger than the solving times. This is caused by expressing the symmetry breaking in DRAT resulting in a 436 Mb proof part.

The Implication Proof. The largest part of the computation consist of showing that $P^{\star}_{15,4,6}$ is unsatisfiable under each of the $5,217,031$ cubes produced by the cube generator. The results of the experiments are shown in Figure 8 (left). The left plot shows that roughly half of the cubes can be solved in a second or less. The average runtime of cubes was 3.35 seconds, while the hardest cube required 1584.61 seconds. The total runtime was 4851.38 CPU hours.

For each cube, we produced a compressed DRAT proof (the default output of `CaDiCaL`). Due to the lack of hints in DRAT proofs, they are somewhat complex to validate using a formally-verified checker. Instead, we use the tool `drat-trim` to trim the proofs and add hints. The result are uncompressed LRAT files, which we validate using the formally-verified checker `cake_lpr`. The verification time was 4336.93 CPU hours, so slightly less than the total runtime.

The sizes of each of the implication proofs show a similar distribution, as depicted in Figure 8 (right). Most proofs are less than 10 MB in size. The

Table 3: Evaluating the effectiveness of the optimizations on $D_{6,11,6}$.

sym	ALOD	plus	#var	#cls	runtime	derivation	proof	check
			935	21086	10741.69	0 b	11.99 Gb	31731.20
		x	1039	7548	809.65	149 Kb	1.29 Gb	1720.82
	x		935	21171	8422.38	1.6 Kb	8.11 Gb	21732.74
	x	x	1039	7633	389.71	151 Kb	1.29 Gb	1708.21
x			935	21286	273.19	436 Mb	0.63 Gb	1390.04
x		x	1039	7748	66.74	436 Mb	0.14 Gb	1022.42
x	x		935	21371	252.71	436 Mb	0.68 Gb	1359.05
x	x	x	1039	7833	55.56	436 Mb	0.10 Gb	997.90

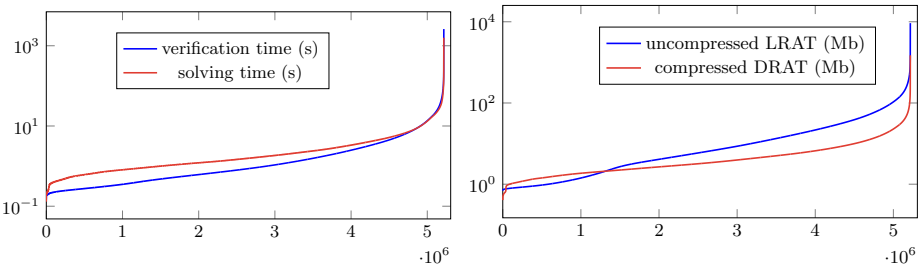


Fig. 8: Cactus plot of solving and verification times in seconds (left) and cactus plot of the size of the compressed DRAT proof and uncompressed LRAT proof in Mb (right).

compressed DRAT proofs are generally smaller compared to the LRAT proofs, but that is mostly due to compression, which reduces the size by around 70%.

The Chessboard Conjecture and its Counterexample. Given that color 1 can be used to fill in $1/2$ of $\mathbb{Z}^2$ in a packing coloring, and the packing colorings found in the past, with $15, 16$ or 17 colors used color 1 with density $1/2$ in a *chessboard pattern* [18], it is tempting to assume that this must always be the case. This way, we conjectured that any instance $D_{r,k,c}$ is satisfiable if and only if it is with the chessboard pattern. The consequence of the conjecture is significant, as if it were true we could fix half of the vertices to color 1, thus massively reducing the size of the instance and its runtime. Unfortunately, this conjecture happens to be false, with the smallest counterexample being $D_{14,14,6}$ as illustrated in Figure 9, which deviates from the chessboard pattern in only 2 vertices. We have proved as well that no solution for $D_{14,14,6}$ deviating in only 1 vertex from the chessboard pattern exists.

Proving the Lower Bound. In order to prove Theorem 1, we require the following 3 lemmas, from where the conclusion easily follows.

Lemma 2. *If $D_{15,14,6}$ is unsatisfiable, then $\chi_\rho(\mathbb{Z}^2) \geq 15$.*

Lemma 3. *If $D_{15,14,6}$ is satisfiable, then $P^\star_{15,14,6}$ is also satisfiable.*

Lemma 4. *$P^\star_{15,14,6}$ is unsatisfiable.*

We have obtained computational proofs of Lemma 3 and Lemma 4 as described above, and thus it only remains to prove Lemma 2, which we include in the appendix. We can thus proceed to our main proof.

Proof (of Theorem 1). Since Martin et al. proved that $\chi_\rho(\mathbb{Z}^2) \leq 15$ [18], it remains to show $\chi_\rho(\mathbb{Z}^2) \geq 15$, which by Lemma 2 reduces to proving Lemma 3 and Lemma 4. We have proved these lemmas computationally, obtaining a single DRAT proof as described in Section 4. The total solving time was 4851.31 CPU hours, while the total checking time of the proofs was 4336.93 CPU hours. The total size of the compressed DRAT proof is 34 terabytes, while the uncompressed LRAT proof weighs 122 terabytes.

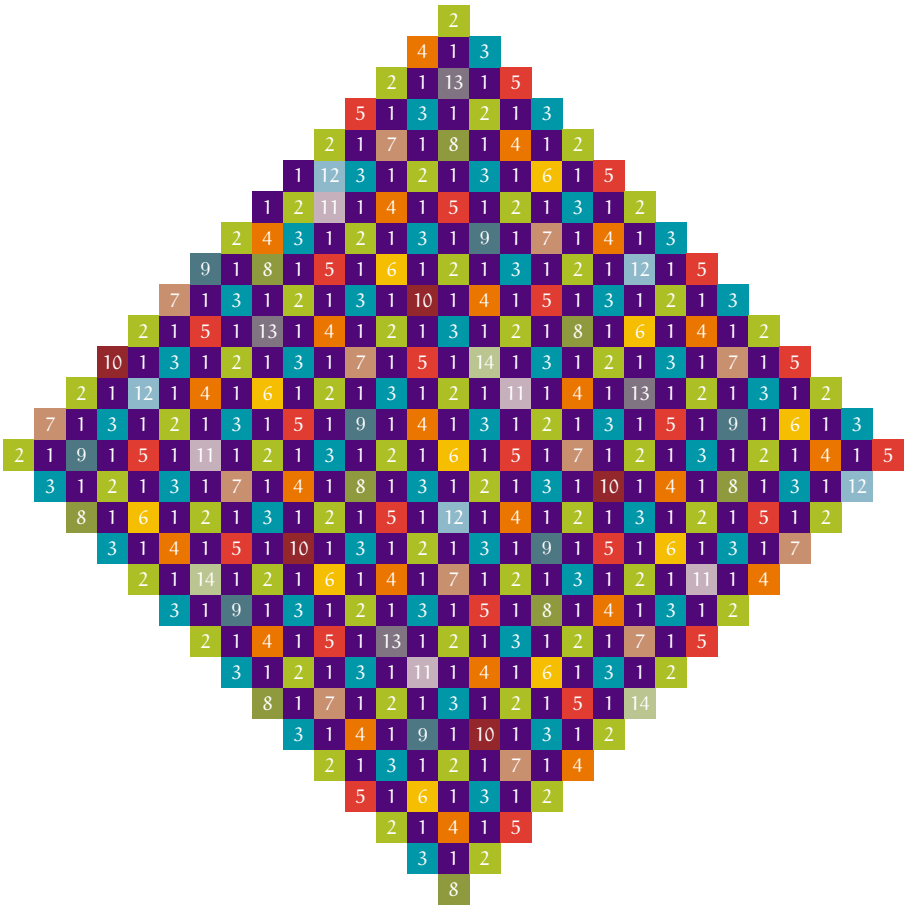


Fig. 9: A valid coloring of $D_{14,14,6}$. No valid coloring exists for this grid with a full chessboard pattern of 1's.

6 Concluding Remarks and Future Work

We have proved $\chi_\rho(\mathbb{Z}^2) = 15$ by using several SAT-solving techniques, in what constitutes a new success story for automated reasoning tools applied to combinatorial problems. Moreover, we believe that several of our contributions in this work might be applicable to other settings and problems. Indeed, we have obtained a better encoding by reverse engineering BVA, and designed a split algorithm that works well coupled with the new encoding; this experience suggests the *split-encoding compatibility* as a new key variable to pay attention to when solving combinatorial problems under the *Cube And Conquer* paradigm. As for future work, it is natural to study whether our techniques can be used to improve other known bounds in the packing-coloring area (see e.g., [3]), as well as to other families of coloring problems, such as *distance colorings* [14].

Acknowledgements We thank the Pittsburgh Supercomputing Center for allowing us to use Bridges2 [4] in our experiments. We thank as well the anonymous reviewers for their comments and suggestions. We also thank Donald Knuth for his thorough comments and suggestions. The first author thanks the Facebook group *"actually good math problems"*, from where he first learned about this problem, and in particular to Dylan Pizzo for his post about this problem.

References

1. Appel, K., Haken, W.: Every planar map is four colorable. Part I: Discharging. Illinois Journal of Mathematics **21**(3), 429 – 490 (1977)
2. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
3. Brešar, B., Ferme, J., Klavžar, S., Rall, D.F.: A survey on packing colorings. Discussiones Mathematicae Graph Theory **40**(4), 923 (2020)
4. Brown, S.T., Buitrago, P., Hanna, E., Sanielevici, S., Scibek, R., Nystrom, N.A.: Bridges-2: A Platform for Rapidly-Evolving and Data Intensive Research, pp. 1–4. Association for Computing Machinery, New York, NY, USA (2021)
5. Buss, S., Thapen, N.: DRAT proofs, propagation redundancy, and extended resolution. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing – SAT 2019. pp. 71–89. Springer International Publishing, Cham (2019)
6. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. In: Proc. KR'96, 5th Int. Conf. on Knowledge Representation and Reasoning, pp. 148–159. Morgan Kaufmann (1996)
7. Ekstein, J., Fiala, J., Holub, P., Lidický, B.: The packing chromatic number of the square lattice is at least 12. CoRR **abs/1003.2291** (2010), http://arxiv.org/abs/1003.2291
8. Fiala, J., Klavžar, S., Lidický, B.: The packing chromatic number of infinite product graphs. Eur. J. Comb. **30**(5), 1101–1113 (jul 2009)
9. Finbow, A.S., Rall, D.F.: On the packing chromatic number of some lattices. Discrete Applied Mathematics **158**(12), 1224–1228 (2010), traces from LAGOS'07 IV Latin American Algorithms, Graphs, and Optimization Symposium Puerto Varas - 2007
10. Goddard, W., Hedetniemi, S., Hedetniemi, S., Harris, J., Rall, D.: Broadcast chromatic numbers of graphs. Ars Comb. **86** (01 2008)
11. Heule, M.J.H.: The DRAT format and drat-trim checker. CoRR **abs/1610.06229** (2016), http://arxiv.org/abs/1610.06229
12. Heule, M.J.H., Biere, A.: What a difference a variable makes. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 75–92. Springer International Publishing, Cham (2018)
13. Heule, M.J.H., Kullmann, O., Wieringa, S., Biere, A.: Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In: Eder, K., Lourenço, J., Shehory, O. (eds.) Hardware and Software: Verification and Testing. pp. 50–65. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
14. Kramer, F., Kramer, H.: A survey on the distance-colouring of graphs. Discrete Mathematics **308**(2), 422–426 (2008)

15. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics **96-97**, 149–176 (1999)
16. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of boolean formulas. In: Proceedings of Haifa Verification Conference 2012 (2012)
17. Martin, B., Raimondi, F., Chen, T., Martin, J.: The packing chromatic number of the infinite square lattice is less than or equal to 16 (2015), http://arxiv.org/abs/1510.02374v1
18. Martin, B., Raimondi, F., Chen, T., Martin, J.: The packing chromatic number of the infinite square lattice is between 13 and 15. Discrete Applied Mathematics **225**, 136–142 (2017)
19. Neiman, D., Mackey, J., Heule, M.J.H.: Tighter bounds on directed Ramsey number R(7). Graphs and Combinatorics **38**(5), 156 (2022)
20. Schwenk, A.: private communication with Wayne Goddard. (2002)
21. Soifer, A.: The Hadwiger–Nelson Problem, pp. 439–457. Springer International Publishing, Cham (2016)
22. Soukal, R., Holub, P.: A note on packing chromatic number of the square lattice. The Electronic Journal of Combinatorics **17**(1), #N17 (Mar 2010)
23. Subercaseaux, B., Heule, M.J.H.: The Packing Chromatic Number of the Infinite Square Grid Is at Least 14. In: Meel, K.S., Strichman, O. (eds.) 25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 236, pp. 21:1–21:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022)
24. Subercaseaux, B., Heule, M.J.H.: The packing chromatic number of the infinite square grid is 15 (2023), https://arxiv.org/abs/2301.09757