An Incremental SAT-Based Approach to the Graph Colouring Problem

Gael Glorian[1], Jean-Marie Lagniez[1], Valentin Montmirail[2], and Nicolas Szczepanski[1]

[1] CRIL, Artois University and CNRS, F-62300 Lens, France
{glorian,lagniez,szczepanski}@cril.fr
[2] I3S, Côte d'Azur University and CNRS, Nice, France
valentin.montmirail@univ-cotedazur.fr

Abstract. We propose and evaluate a new *CNF encoding* based on *Zykov's tree* for computing *the chromatic number of a graph*. Zykov algorithms are branch-and-bound procedures, that branch on pairings of vertices that express whether or not two non-adjacent vertices have the same colour. Thus, vertices with the same colour are contracted whereas edges are added between vertices when they have different colours. Such pairings make possible the use of a well-known recurrence relation, that states that the chromatic number of a graph cannot be lower than the the chromatic number of its subgraphs. Our encoding associates with any graph and integer k a CNF formula that is satisfiable if and only if the chromatic number of the graph is at least k. We first show that any colouring satisfying a complete pairing always required a fixed number of colours. Then, we establish a *CNF encoding that counts* the number of colours required by a pairing. However, due to a large number of clauses required to encode transitivity constraints on pairings, a direct encoding does not scale well in practice. To avoid this pitfall, we designed a CEGAR-based (Counter-Example Guided Abstraction Refinement) approach that only encodes a part of the problem and then adds the missing constraints in an *incremental way* until a *valid solution* with k colours is found or the unsatisfiability of the problem is proven, meaning that the chromatic number of the graph is greater than k. We show that our encoding scheme performs in many cases significantly better than the state-of-the-art approaches to colouring.

Keywords: Chromatic number, Zykov, CEGAR, SAT encoding

1 Introduction

Graph colouring is the problem of assigning a minimum number of colours to all vertices of a graph such that no adjacent vertices, *i.e.* vertices that are linked by an edge, receive the same colour. The smallest number of colours needed to colour the graph is called the *chromatic number* of the graph. The problem appears in a variety of areas included (but not limited to) scheduling problem [1], sudokus [2], register allocation used in compiler optimization [3], sports scheduling [4] and exam timetabling [5].

Determining the chromatic number of a graph is an NP-hard task. Several computational approaches for the colouring problem, that prove empirically viable for many instances, have been pointed out [6,7,8,9,10,11,12,13]. They can be divided into two categories: complete methods and incomplete methods. Incomplete methods are usually

based on greedy or meta-heuristic algorithms and are able to deal with graph containing a large number of vertices. Nevertheless, such methods are only able to find bounds that can be far from the optimal solution. Complete approaches are commonly based on the branch-and-bound paradigm and are able to guarantee that the returned solution is optimal. In this work, we propose a complete approach to compute the chromatic number of a graph.

Recently, the authors of [13] proposed a hybrid CP/SAT approach, called `gc-cdcl`, using new lower bound and branching heuristic, and that is so far the most efficient approach to graph colouring. In the CP-based approaches cited before, seeing all the colours as a *domain* for each vertex is common. However, because of the interchangeability of colours, such a representation leads to many symmetries which have to be broken. As done previously [7,13], we propose to take advantage of Zykov's tree to break symmetries. More precisely, our CNF encoding does not represent the allowed colours but choose to encode with propositional variables the fact that two vertices are coloured in the same way. Thus, instead of colouring the graph, our encoding tries to pair vertices between them. We demonstrate that, once all vertices are correctly paired, the number of colours required to colour the graph, while satisfying the pairing, is fixed and can be computed efficiently by a Boolean circuit. Then, we present a CNF encoding of this Boolean circuit together with the set of clauses representing the constraints ensuring the pairing correctness, allows us to represent as a whole the k-Colouring decision problem. We empirically tested Partial `MaxSAT` solvers, as well as linear and binary searches on the number of colours, using this encoding to see the performances against state-of-the-art approaches. Unfortunately, we quickly observed that this encoding cannot scale on large graphs: a cubic number of clauses are needed to ensure the correctness of pairings.

To make our approach scalable in practice, we propose a CEGAR approach using our new CNF encoding. The idea is as follows: instead of designing an equisatisfiable propositional formula, we generate an *under-abstraction* (a formula which is under-constrained, also called *relaxation* in other domains). If this under-abstraction is unsatisfiable, then, by construction the original formula is unsatisfiable; otherwise, the SAT solver outputs a model that can then be checked in polynomial time. It could be the case that the approach is lucky and the model of the under-abstraction is also a model of the original formula, in which case the problem is solved. In general, the under-abstraction is continually refined, *i.e.*, it comes closer to the original formula and, in the worst-case, will eventually become equisatisfiable with the original formula after a finite number of refinements. Notably, CEGAR has been successfully proposed in many problems such as Bounded Model Checking [14], Satisfiability Modulo Theory [15], Planning [16], the Hamiltonian Cycle Problem [17], and more recently RCC8-Reasoning [18] and Minimal S5 Satisfiability Problem [19].

Abstracting decision problems with a CEGAR-under approach is well-known in the SAT community. However, the CP/OR community is more familiar with the Logic-based Benders decomposition (LBBD) [20], which can be viewed as the CEGAR-under approach for optimization. It is used in many domains where one wants to abstract and solve an optimization problem. LBBD approaches are orders of magnitude faster than state-of-the-art MIP for all problems where it has been applied [21,22,23]. One

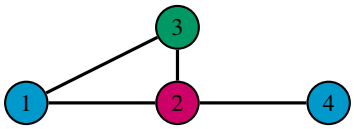

Fig. 1. Illustration of a graph G with 4 vertices such that $\chi(G) = 3$.

could also see the CEGAR-under approach as a Lazy-SMT approach [24,25], where the problem-specific knowledge extracted from the abstraction is used to guide the refinement process, instead of a theory solver.

The paper is organized as follows: after a few preliminaries, we present our new CNF based encoding and demonstrate its soundness and completeness. Then, we demonstrate how it can be adapted to be used in a CEGAR-based approach. Then, we show empirically that first: the direct SAT encoding, either with a linear/binary search on the number of colours or via a `MaxSAT` solver is quite competitive against the state-of-the-art approaches for minimal graph colouring, but, more importantly, that the CEGAR-based approach outperforms all the tested approaches on the benchmarks that have been considered.

2 Preliminaries

2.1 *k*-colouring problem

An undirected graph is a pair $G = (V, E)$, where V it the set of $|V| = n$ vertices (or nodes) and $E \subseteq V \times V$ a set of edges. A sub-graph $G' = (V', E')$ of $G = (V, E)$ is a graph such that $V' \subseteq V$ and $E' \subseteq E$. Let us note $G' = G \setminus V'$ the sub-graph G' obtained by removing from G vertices of V', *i.e.* $G' = (V \setminus V', \{(u, v) \in E \mid \{u, v\} \cap V \neq \emptyset\})$. The contraction G/uv of a graph G is the graph obtained by removing any edge containing the vertices u and v, and by merging the vertices. $G + uv$ is the graph G with the edge (u, v) added.

A graph colouring problem aims to assign colours to certain elements of a graph subject to certain constraints. Vertex colouring is the most common graph colouring problem and is defined as follows: given an undirected graph $G = (V, E)$ and an integer k (number of colours), find a mapping $c : V \mapsto \{1, 2, \ldots, k\}$ that associates, each vertex $i \in V$ of G, a colour $c(i)$ so that no adjacent vertex $j \in V$ shares the same colour (*i.e.* $\forall (i, j) \in E$ we have $c(i) \neq c(j)$). A mapping c which verifies that $c(i) \neq c(j), \forall (i, j) \in E$ is called a valid colouring. We assume that V contains only integers from 1 to n. Moreover, for obvious reasons, we suppose that $\nexists i \in V$ such $(i, i) \in E$.

The most common type of vertex colouring seeks to minimise the number of colours for a given graph. The smallest number of colours needed for a graph G is called its **chromatic number** and is denoted by $\chi(G)$. An illustration of a graph G such that $\chi(G) = 3$ is given in Fig.1. The problem of finding a minimum colouring for a graph is known to be NP-hard [26]. In fact, graph colouring is even NP-hard to approximate in specific scenarios [27]. Because its NP-hardness, k-colouring problem can be naturally translated into CNF.

2.2 Logical preliminaries and CEGAR framework

Let $\mathcal{L}$ be a standard Boolean logical language built on a finite set $\mathbb{P}$ of Boolean variables and usual connectives (namely, $\wedge$, $\vee$, $\neg$, $\Rightarrow$ and $\Leftrightarrow$ standing for conjunction, disjunction, negation, material implication and equivalence, respectively). Formulas will be noted using lower-case Greek letters such. Regarding the semantics aspect of the propositional logic, an interpretation $\mathcal{I}$ assigns valuation from $\{\mathtt{1}, \mathtt{0}\}$ to every Boolean variable, thus, following usual compositional rules, to all formulas of $\mathcal{L}$. We denote by $\mathcal{I}(l)$ is $\mathtt{1}$ if l is satisfied by $\mathcal{I}$, and $\mathtt{0}$ otherwise. A formula α is *satisfiable* (also called consistent) when there exists at least one interpretation that satisfies α, i.e., that makes α true: such an interpretation is called a model of α and is represented by the set of variables that it satisfies. If a formula is false for any interpretation, this formula is *unsatisfiable*. $\models$ denotes deduction, i.e., $\alpha \models \beta$ denotes that β is a logical consequence of α, namely that β is satisfied in all models of α. Without loss of generality, any formula in $\mathcal{L}$ can be represented (while preserving satisfiability) in conjonctive normal form (CNF) i.e., as a conjunction of clauses [28], where a clause is a finite disjunction of literals and where a literal is a Boolean variable that can be negated.

Example 1 (Basic graph colouring encoding). Let $G = (V, E)$ be a graph, the following CNF formula encodes the problem of deciding if it is possible to colour the graph G with at most k colours (x_{vj} is true when the vertex v takes colour j):

$$\bigwedge_{v \in V} \Big(\bigvee_{i=1}^{k} x_{vi}\Big) \wedge \bigwedge_{1 \leq i < j \leq k} (\neg x_{vi} \vee \neg x_{vj})) \wedge \bigwedge_{(u,v) \in E} \Big(\bigwedge_{i=1}^{k} (\neg x_{ui} \vee \neg x_{vi})\Big)$$

Counter-Example-Guided Abstraction Refinement (CEGAR) is an incremental way to decide the satisfiability of problems. It has been originally designed for model checking [14], *i.e.*, to answer questions such as "Does $\alpha \models \beta$ hold?" or, likewise, "Is $\phi = (\alpha \wedge \neg\beta)$ unsatisfiable?", where α describes a system and β a property. For such highly structured problems, it is often the case that only a small part of the formula is needed to answer the question. The keystone of CEGAR is to replace ϕ by an abstraction ϕ', easier to solve in practice. There are two kinds of abstractions: an over-abstraction (resp. under-abstraction) of ϕ is a formula $\hat{\phi}$ (resp. $\check{\phi}$) such that $\hat{\phi} \models \phi$ (resp. $\phi \models \check{\phi}$) holds. $\hat{\phi}$ has at most as many models as ϕ and $\check{\phi}$ has at least as many models as ϕ.

Roughly, the CEGAR-based methods work on an abstraction of the original model, which is the current problem targeted by the solver. If it is an over-abstraction (resp. under-abstraction) which is proven satisfiable (resp. unsatisfiable), then the initial problem is also proven satisfiable (resp. unsatisfiable). Otherwise, the result returned by the solver can be spurious, and in this case, several situations may arise. If it is possible to check that the outcome is a solution to the initial problem, in the positive case, then the initial problem is also solved. It is also possible to decide the problem with the current result when the abstraction is equisatisfiable to the input problem. In all other situations, the CEGAR method refines the abstraction using information from the outcome in order to carefully select the next abstraction.

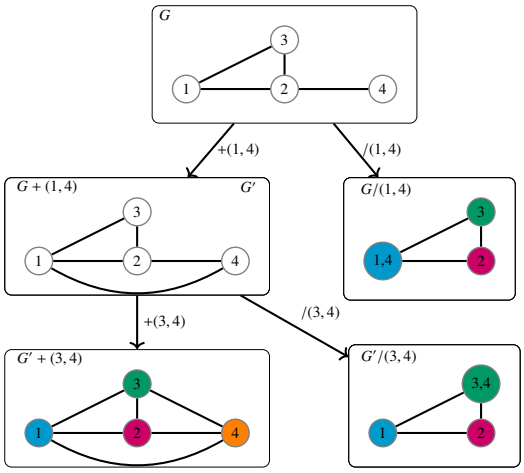

Fig. 2. Zykov's tree for the graph in Fig.1

3 From Colouring to Zykov, and Vice Versa

A straightforward algorithm for deciding whether or not a graph can be coloured with k colours is to search among all mappings from the set of vertices to the set of colours (brute force). This algorithm, despite being correct, is inefficient for all but the smallest input graphs. Its lack of effectiveness can be partially explained by the fact that colours are interchangeable. When a colouring is incorrect, conflicting colourings exist that can be obtained by permutation. This observation can reveal, partially, why CP solvers are not able to decide the k-colouring problem for *big* graphs when the basic encoding (see Section 1) is used. Even if CP solvers are efficient on a wide range of problems, it is also known that they do not perform well on symmetric problems [29]. To break symmetries, [7] propose to take advantage of Zykov's tree in order to add conflict clauses on the fly, in such a way that they cover all permutations of the colours.

We now move on towards the definitions of Zykov trees. Let consider $G = (V, E)$ a graph, and x and y two non-adjacent vertices of G. In any proper colouring of G, either x and y have different colours or they have the same colour. Thus, a well-known result, so-called Zykov's deletion-contraction[1] recurrence is defined as follows:

$$\chi(G) = \texttt{min}\{\, \chi(G/(ij)),\ \chi(G + (ij)))\ \}\ \forall (i, j) \notin E \tag{1}$$

Zykov's tree can be recursively constructed by starting with the single node G, the root of our binary tree, and branching repeatedly using *vertex-contraction* on one side and *edge-deletion* on the other side on vertices that are not yet connected. Each leaf of a Zykov tree for G is a complete graph. Of course, we cannot branch on G if G is complete, and if $G = (V, E)$ is complete, it is easy to show that $\chi(G) = |V|$. By Eq.1, we know that $\chi(G)$ is the minimum value among all leaves of a Zykov tree for G. Let us consider the graph of Fig.1, Fig.2 shows the related Zykov tree.

[1] The historical name is misleading: it either merges vertices or adds non-existing edges.

The search space of the coloured graphs visited when using Zykov's tree is more succinct than the one visited by an approach that branches on colours. Indeed, it is enough to observe that methods that branch on colours implicitly construct a Zykov tree. Except for the first vertex, each time a new vertex is coloured, it can be contracted with all the vertices previously coloured in the same way; and an edge can be added between the newly coloured one and the vertices we already coloured differently. When the colouring c gives colour at each vertex, each pair of vertices is either contracted or an edge is added between them. The resulting graph is then complete, and it is present in the Zykov's tree as leaf. Because the tree is constructed in a deterministic manner, it is easy to show that with each colouring c it is possible to associate only one leaf of the Zykov tree. Note that the opposite does not hold: each permutation c' of a colouring c leads to the same Zykov's tree leaf. Even if Zykov's trees contain symmetric nodes, there are fewer symmetries in them than in the search space explored by methods that colour vertices.

As explained in [7,13], it is possible to explore the search space represented by Zykov's tree using a CNF encoding. For a graph $G = (V, E)$, this encoding considers a set S of Boolean variables s_{ij} for all $i, j \in V$ that are used to pair the vertices together. Because edges are not oriented, we only consider s_{ij} s.t. $i < j$. In the case when $i > j$, s_{ij} is a renaming for s_{ji}. A variable s_{ij} set to true means that i and j are coloured in the same way (contraction). When set to false, it means there exists an edge between i and j (deletion). The set of clauses $\texttt{tr}(G)$ consists in unit literals $\neg s_{ij}$ for all $(i, j) \in E$, and a cubic number of clauses to ensure path consistency between Boolean variables. For every triplet i, j and k, we have to encode $s_{i,j} \wedge s_{j,k} \Rightarrow s_{i,k}$ (transitivity) and $s_{i,j} \wedge s_{i,k} \Rightarrow s_{j,k}$ (Euclideanity). In the following, we call *pairing* an assignment that gives a value for each s_{ij}. A *valid pairing* is a pairing that satisfies both path consistency and unit clauses. Let $G = (V, E)$ be a graph, the clauses $\texttt{tr}(G)$ that encode valid pairings are given by:

$$\texttt{tr}(G) := \neg s_{ij}\ \forall (i, j) \in E \wedge \texttt{transitivity}() \wedge \texttt{euclideanity}()$$
$$\texttt{transitivity}(i, j, k) := (\neg s_{ij} \vee \neg s_{jk} \vee s_{ik})$$
$$\texttt{euclideanity}(i, j, k) := (\neg s_{ij} \vee \neg s_{ik} \vee s_{jk})$$
$$\texttt{transitivity}() := \texttt{transitivity}(i, j, k)\ \forall i, j, k \in V s.t\ (i < j)\ and\ (j < k)$$
$$\texttt{euclideanity}() := \texttt{euclideanity}(i, j, k)\ \forall i, j, k \in V s.t\ (i < j)\ and\ (j < k)$$

Example 2. The following CNF formula encodes the Zykov search space induced by the graph given in Fig.1:

$$\begin{aligned}\texttt{tr}(G) = {} & \neg s_{12} \wedge \neg s_{13} \wedge \neg s_{23} \wedge \neg s_{24} \\ & \wedge (\neg s_{12} \vee \neg s_{23} \vee s_{13}) \wedge (\neg s_{13} \vee \neg s_{23} \vee s_{12}) \wedge (\neg s_{13} \vee \neg s_{12} \vee s_{23}) \\ & \wedge (\neg s_{12} \vee \neg s_{24} \vee s_{14}) \wedge (\neg s_{12} \vee \neg s_{14} \vee s_{24}) \wedge (\neg s_{14} \vee \neg s_{24} \vee s_{12}) \\ & \wedge (\neg s_{23} \vee \neg s_{34} \vee s_{24}) \wedge (\neg s_{23} \vee \neg s_{24} \vee s_{34}) \wedge (\neg s_{24} \vee \neg s_{34} \vee s_{23}) \\ & \wedge (\neg s_{13} \vee \neg s_{34} \vee s_{14}) \wedge (\neg s_{13} \vee \neg s_{14} \vee s_{34}) \wedge (\neg s_{14} \vee \neg s_{34} \vee s_{13})\end{aligned}$$

which is, after unit propagation: $\texttt{tr}(G) = (s_{14} \vee s_{34})$. As saw earlier on Fig.2, either we assign s_{14} to true, and s_{34} to false, which gives us the graph $G/(1, 4)$, or we assign s_{14} to false, and s_{34} to true, and thus we obtain the graph $G + (1, 4)/(3, 4)$, or finally, we assign both variables to true, which gives us the final leaf, the graph $G + (1, 4) + (3, 4)$.

Thus, searching among the Zykov's tree leaves, ammounts to searching among the set of valid pairings. Unfortunately, the previous encoding does not give the number of colours associated with a valid pairing. If we look back at the Zykov's tree, every leaf is a complete graph; their chromatic number is their number of vertices. In our case, the graph is not explicitly constructed and the information is missing. However, Property 1 shows that it is enough to know which vertices are paired together to compute the number of colours needed while respecting a specific pairing. The general idea is that if we try to colour the graph vertex by vertex, following the information contained in the pairing, then an additional colour is required for the vertex j exactly when all the already coloured vertices i are such that the s_{ij} are false. Thus, it is enough to consider vertices in a given order to compute the number of required colours.

Property 1. Let us consider $G = (V, E)$ a graph s.t. $|V| > 0$ and S the set of pairing variables associated to G. If $I_S \in 2^S$ is a valid pairing, then the number of colours needed to colour G w.r.t. I_S is given by the following formula:

$$\Psi(I_S) = 1 + \sum_{j=2}^{n} min(I_S(s_{ij}) \text{ s.t. } s_{ij} \in S \text{ and } i < j)$$

Proof. Let us demonstrate this result by structural induction on the number of nodes.
Base case: Show that the statement is true for the sub-graph $G \setminus \{v_2, v_3, \ldots, v_n\}$. It is clear that the number of colours needed to colour a graph with only one node is $\Psi(I_S) = 1$.
Inductive step: Show that if this property holds for $G \setminus \{v_n\}$, then it also holds for G. Let us consider $S' = \{s_{ij} \in S \text{ s.t. } i < j \leq n-1\}$. Using the induction hypothesis, we have $\Psi(I_{S'}) = 1 + \sum_{j=2}^{n-1} min(I_{S'}(s_{ij}) \text{ s.t. } s_{ij} \in S' \text{ and } i < j)$. By construction of S', we also have $\Psi(I_{S'}) = 1 + \sum_{j=2}^{n-1} min(I_S(s_{ij}) \text{ s.t. } s_{ij} \in S \text{ and } i < j)$. Now, let us consider G with the associated pairing I_S. It is easy to show that if there exists $s_{in} \in S$ that is true under I_S and s.t. $i < n$, then no additional colour is required (actually, the vertex v_n can be coloured as the vertices v_i). Otherwise, if $\forall s_{in} \in S$ s.t. $i < n$ the value of $I_S(s_{in})$ is false, then it is impossible to colour the last node with an already used colour. Consequently, the number of additional colours needed when considering the last vertex is $min(I_S(s_{in}) \text{ s.t. } s_{in} \in S \text{ and } i < j)$, and then we have:

$$\begin{aligned}\Psi(I_S) &= \Psi(I_{S'}) + min(I_S(s_{in}) \text{ s.t. } s_{in} \in S \text{ and } i < j) \\ &= 1 + \sum_{j=2}^{n-1} min(I_S(s_{ij}) \text{ s.t. } s_{ij} \in S \text{ and } i < j) + min(I_S(s_{in}) \text{ s.t. } s_{in} \in S \text{ and } i < j) \\ &= 1 + \sum_{j=2}^{n} min(I_S(s_{ij}) \text{ s.t. } s_{ij} \in S \text{ and } i < j)\end{aligned}$$

□

It is well known that computing the minimum value of a Boolean vector can be encoded as an AND gate. We can rewrite the previous sum as one on the set of Boolean variables $C = \{c_2, c_3, \ldots, c_n\}$ s.t. $\Psi(I_S) = 1 + \sum_{j=2}^{n} c_j$ where c_j is defined $\forall 1 < j \leq n$ as follows:

$$c_j \Leftrightarrow \bigwedge_{s_{ij} \in S \text{ and } i<j} \neg s_{ij} \tag{2}$$

By considering $\texttt{tr}(G)$ and the constraint generated by Equation.2, it is possible to compute the chromatic number of a given graph G by considering the minimisation problem that consists in satisfying a maximum number of c_i to false. This problem can be encoded as a partial `MaxSAT` problem where the hard clauses Σ are given by $(\texttt{tr}(G) \wedge$

Equation 2) and the soft clauses Δ are given by the units literals $c_i \in C$. Thus, the minimum number of colours required is $1 + \texttt{MaxSAT}(\Sigma, \Delta)$.

To deal with the decision problem that consists in deciding if the chromatic number of a given graph is at least k, it is enough to consider the CNF formula composed with the clauses of ($\texttt{tr}(G) \wedge$ Equation 2) and the set of clauses that encodes ($\sum_{j=2}^{n} c_j \leq k-1$) which can be represented using classical encoding, such as a Cardinality Network Encoding [30]. In the following, we consider different Partial `MaxSAT` solvers to determine wherther the approach that consists in minimising the number of c_j assigned to true is a competitive approach to the *minimum graph colouring problem*. However, by looking at the CNF encoding, we can observe that `transitivity`() and `euclideanity`() add a cubic number of ternary clauses which slow down the whole approach (see section 6). One way to circumvent them is to use a CEGAR version of the encoding and find a way to refine it by adding as few clauses as possible while minimising the number of SAT calls.

4 A CEGAR Version of the Encoding

The main concern with our encoding is the number of clauses needed to guarantee pairing validity. To overcome this difficulty, we propose to relax transitivity and Euclideanity constraints and incrementally execute a SAT solver on an under-approximation $\check{\phi}$ of the problem. If the solution violates some transitivity or Euclideanity constraints, we prevent them in the new abstraction by adding clauses. To compute the violated constraints, we should consider each triple (likely a large number) and check its consistency with the result. Such an approach is clearly impractical when the number of vertices grows.

To avoid this pitfall, we propose to colour the graph using the information contained in the returned pairing λ. Indeed, we assign each vertex u with the set $c[u]$ of the k possible colours. Then, we consider each vertex u incrementally in the natural order and, when it is possible ($c[u] \neq \emptyset$), select an available colour i for it in $c[u]$. Afterward, the pairings are used in order to colour as u every vertex v such that $s_{uv} \in \lambda$. If v cannot be coloured as u because $i \notin c[v]$, then $c[v]$ becomes empty and we consider a vertex w that removes i in $c[v]$. In such a case, transitivity and Euclideanity constraints on (u, v, w) are added in $\check{\phi}$. We next consider every vertex vs.t.$\neg s_{uv} \in \lambda$ and remove from $c[v]$ the colour i. If $c[v]$ becomes empty, we search for a vertex w that forces v to be coloured with the colour i. In the case when such w exists, the transitivity or/and Euclideanity constraints on the triple (u, v, w) are missing and must be added. In both cases, the solver is run once more on the updated under-approximation and the all process is repeated until spurious triples can be identified. Algorithm 1 gives the pseudo-code of our checking method. The following property shows that if check(λ, G, k) returns $\emptyset$ then it is possible, following λ, to colour G with k colours.

Property 2. Let $G = (V, E)$ a graph, k an integer and λ an interpretation that satisfies an under-abstraction of $tr(G)$ that contains at least: the unit clauses $\neg s_{uv}$ for all $(u, v) \in E$, the clauses encoding Equation.2 and the clauses that encodes $\sum_{c_i}^{n} c_i \leq k-1$. If $T =$ check(λ, G, k) returns $\emptyset$ then it is possible, following λ, to colour G with k colours. Otherwise, $\exists (i, j, k) \in T$ then $\lambda \not\models \texttt{transitivity}(i, j, k) \wedge \texttt{euclideanity}(i, j, k)$.

Algorithm 1: check(λ s.t $\lambda \models \check{\phi}, G = (V, E)$ a graph, k an integer) : T a set

```
1  T ← ∅; c a map;
2  for u ← 1 to |V| do c[u] ← {1, 2, ..., k};
3  for u ← 1 to |V| do
4      if c[u] ≠ ∅ then
5          c[u] = {i} s.t. i ∈ c[u];
6          for v ← u + 1 to |V| do
7              if ¬s_uv ∈ λ then
8                  c[v] = c[v] \ c[u];
9                  if c[v] = ∅ and ∃w s.t. w < u, s_wv ∈ λ and c[w] = c[u] then
10                     T ← T ∪ {(u, v, w)}
11             else
12                 c[v] = c[v] ∩ c[u];
13                 if c[v] = ∅ then
14                     let w s.t. w < u, s_vw ∈ λ and c[w] ≠ c[u] or ¬s_wv ∈ λ and c[w] = c[u];
15                     T ← T ∪ {(u, v, w)};
16 return T;
```

Proof. First, let us demonstrate that if $\exists(i, j, k) \in T$ then $\lambda \not\models \texttt{transitivity}(i, j, k) \wedge \texttt{euclideanity}(i, j, k)$. Let us consider the two cases where a triple t can be added:

- t is added line 10, that means we have two vertices u and v s.t. $u < v$, $c[u] = \{i\}$, $\neg s_{uv} \in \lambda$ and $c[v] \setminus \{i\} = \emptyset$. By the if condition (line 9), we also have $\exists w$ s.t. $w < u$ $s_{vw} \in \lambda$ and $c[w] = c[u]$. Since $c[u] = \{i\}$ and $w < u$ then $s_{wu} \in \lambda$ (otherwise i would have been removed from $c[u]$ by w). Thus, we have $\{s_{wu}, s_{wv}, \neg s_{uv}\} \subseteq \lambda$ which implies that $\lambda \not\models \textit{euclideanity}(w, u, v)$.
- t is added line 15, that means we have two vertices u and v s.t. $u < v$, $c[u] = \{i\}$, $s_{uv} \in \lambda$ and $i \notin c[v]$. $i \notin c[v]$ means that i has been previously removed (line 8 or line 12) when some vertex $w < u$ has been considered. Two cases have to be considered. Either $c[w] = \{i\}$, in this case $\neg s_{wv} \in \lambda$ and we necessary have $\{\neg s_{wv}, s_{wu}, s_{uv}\} \subseteq \lambda$ ($s_{wu} \in \lambda$ because $i \in c[u]$ and $w < u$). Or $c[w] \neq \{i\}$, which implies that $s_{wv} \in \lambda$ and then we have $\{s_{wv}, \neg s_{wu}, s_{uv}\} \subseteq \lambda$ (*similarly*, $\neg s_{wu} \in \lambda$ because $i \in c[u]$ and $w < u$). In both cases, we have $\lambda \not\models \texttt{transitivity}(u, v, w) \wedge \texttt{euclideanity}(u, v, w)$.

Therefore, if $\exists(i, j, k) \in T$ then $\lambda \not\models \texttt{transitivity}(i, j, k) \wedge \texttt{euclideanity}(i, j, k)$. Now, let us demonstrate that if check(λ, G) returns $\emptyset$ then λ makes possible the construction of a k-colouring for $G = (V, E)$. First, we show that if T is empty then all the vertices v are coloured in $c[v]$, *i.e.* $\nexists v \in V$ s.t. $c[v] = \emptyset$. Towards a contradiction, suppose that after the for-loop (line 16) $\exists v \in V$ s.t. $c[v] = \emptyset$ and $T = \emptyset$. By considering the first emptied vertex v, it is easy to show that the only situation, where $c[v]$ can be emptied whereas no triple are added in T, is line 10 where no vertex w can be found when u is considered. Indeed, if $c[v]$ becomes empty at line 12, then a triple is added immediately (lines 13 – 15). Otherwise, if we can find $w < u$ s.t. $s_{wv} \in \lambda$ and $c[w] = c[u]$ then a triple is necessary added. Therefore, $\forall w < v$ we have $\neg s_{wv} \in \lambda$ and then by Equation.2 we also have $c_u \in \lambda$. Thus, since $\sum_{i=2}^{n} c_i <= k - 1$ must be satisfied by λ then $\sum_{i=2}^{v-1} c_i <= k - 2$

should be satisfied by λ as well. Since $c[v]$ is empty and $\neg s_{wv} \in \lambda$ for all $w < v$, then there exist k vertices w_j s.t. $c[w_j] = \{j\}$ for each $j \in \{1, 2, \ldots, k\}$. Thus, for each colour j it is possible to determine the vertex w'_j that has been assigned first to the colour j.

Let us show that $\forall w'_j, \neg s_{ww'_j} \in \lambda, \forall w < w'_j$. Towards a contradiction, let us suppose that $\exists w$ s.t. $s_{ww'_j} \in \lambda$. Since the vertices are considered in the natural order, w is coloured before w'_j. But since w'_j is the first vertex assigned to j, we have $c[w] \neq \{j\}$. Thus, since $w < v$, we have $c[w] \neq \emptyset$, and thus the following instructions are executed. Because $s_{ww'_j} \in \lambda$ the else part of the if/else instruction (lines 11–15) should have deleted the colour j from $c[w'_j]$, making impossible to colour w'_j in j. Consequently, $\forall w'_j$ we have $\neg s_{ww'_j} \in \lambda, \forall w < w'_j$.

Therefore, there exist k vertices w'_j s.t. $\neg s_{ww'_j} \in \lambda$, $\forall w < w'_j$. Since vertex 1 is necessary coloured first, it is easy to show that $w'_1 = 1$. By construction, $\check{\phi}$ encodes Equation.2, and then we have $\lambda \models \bigwedge_{j=2}^{n}(c_j \Leftrightarrow \bigwedge_{s_{ij} \in S \text{ and } i<j} \neg s_{ij})$. Consequently, λ satisfies c_v at least $k-1$ literals c_i s.t. $i < v$ and then $\lambda \not\models \sum_{i=2}^{v-1} c_i <= k-2$, proving the claim.

We conclude by proving that c is a k-colouring for G. Since $1 \leq i \leq k$, stating that c is a valid colouring necessary implies it is a k-colouring. Thus, it is enough to show that c is a valid colouring. Towards a contradiction, suppose c is not valid. Then, there exist two vertices u and v s.t. $c[u] = c[v] = \{i\}$ and $(u, v) \in E$. Without loss of generality, suppose that $u < v$. Because $(u, v) \in E$, $\neg s_{uv}$ is a unit clause of $\check{\phi}$ and therefore $\neg s_{uv} \in \lambda$. Thus, when u is coloured (line 4), the set $c[v]$ is updated w.r.t. λ. Since $\neg s_{uv} \in \lambda$, i should be removed from $c[v]$ (line 8) and $c[v]$ is necessary different from $\{i\}$. To conclude, if $T = \emptyset$ then c is a valid colouring of G and a k-colouring of G. □

5 Related Work

Brelaz' `Dsatur` (*degree of saturation*) [11] greedy algorithm, is one of the oldest but still successful technique for graph colouring. It works as follows: for each vertex v, we compute the *degree of saturation* of v and we use this value and the degree of each vertex to determine an order to colour the vertices. This heuristic is used within a branch-and-bound algorithm with one variable per vertex whose domain is the set of possible colours.

Another way to compute the chromatic number of a graph is to take advantage of its NP-hardness and use the constraint programming paradigm. Since it is trivial to encode colouring problems into propositional logic, several SAT-based approaches have been proposed. To be efficient in practice, such approaches also add constraints to break symmetries or to represent explicitly the information between non-adjacent vertices [12,7]. Let us cite `color6` that is one of the most efficient solvers [12].

Schaafsma *et al.* [7] CP approach is very clever but unfortunately, we cannot compare our approach to it. As explained in [13]: *"We could not compare our method to the method of Schaafsma et al. directly. [...] Firstly, the algorithm is restricted to instances with at most 32 colours. Secondly it solves the satisfiability problem $\chi(G) \leq K$ and uses a file converter. Finally, the changes made to Minisat's code do not seem to be robust."*. However, their approach deserves some explanations. The authors also exploit Zykov's contraction. They introduce additional variables e_{ij} in the encoding if the vertices i and

j should be merged using Zykov's contraction. However, a fundamental difference with our approach is that they encode the colours as a CP domain. For each colour c and each vertex v, a Boolean variable x_{vc} stating that the vertex v has the colour c is used. Even though they propose symmetry breaking [31] to speed-up their approach, they, unfortunately, suffer lack of efficiency when the number of colours is large.

The second work, closely related to our own one, is the most recent (and most efficient) approach to the minimum graph colouring problem, namely `gc-cdcl` [13]. Unlike Schaafsma *et al.* [7], they do not need to encode colours as a CP domain since they have a variable for every non-edge in the input graph. Again, as in [7] and our approach, they rely on Zykov's recurrence.

Hebrard and Katsirelos [13] made the same analysis as us about the performance of Schaafsma *et al.*. They propose a CP hybridisation introducing peculiar propagators to enforce constraints on the bounds. Contrastingly, we have developed a complete SAT-based approach when the constraints are relaxed by performing a CEGAR-based search on the spurious examples that the SAT solver may find.

6 Experimental Evaluation

To assess our approach, we created a tool called Picasso. Picasso is an open-source solver (written in C++)[2]. In the following, we compare different versions of our method:

- **Full *Decision***, full encoding that decides the bound by using sum constraints and oracle calls (ascending (**1toN**), descending (**Nto1**) and binary search (**Dicho**);
- **Full *MaxSAT solver***, full encoding in combination with a MaxSAT solver;
- **CEGAR *Decision***, the relaxed counterpart of the full encoding that decides the bound by using sum constraints and oracle calls (ascending (**1toN**), descending (**Nto1**) and binary search (**Dicho**).

We used, as SAT solver, glucose (4.0) [32,33], in incremental mode (with its caching activated). We also tried several Partial `MaxSAT` solvers, such as: maxHS-b [34], mscg2015b [35], RC2-B [36] and MSUnCore [37]. We selected MaxSAT solvers which have shown good performances in the 2018 MaxSAT competition [38]. We considered the state-of-the-art approaches for graph colouring according to [13]: `gc-cdcl` [13], `color6` [12] and `DSatur` [11]. We used instances from a colouring webpage[3], the "Graph coloring" and the "Quasi-random coloring" problems. This leads to a list of 159 instances. The experiments ran on a 4 cores Xeon at 3.3 GHz with CentOS 7.0. The memory limit was set to 32GB and the runtime limit to 900 seconds per solver per benchmark.

6.1 Overall evaluation of the different methods

We start with an overall evaluation of the effectiveness of each approach. The results are presented in Fig.3 under the form of cactus plots. It makes explicit the number of instances solved in a given amount of time per instance. As expected, the methods using

[2] The source are accessible at: `https://github.com/Mystelven/picasso`
[3] `https://mat.tepper.cmu.edu/COLOR03/`

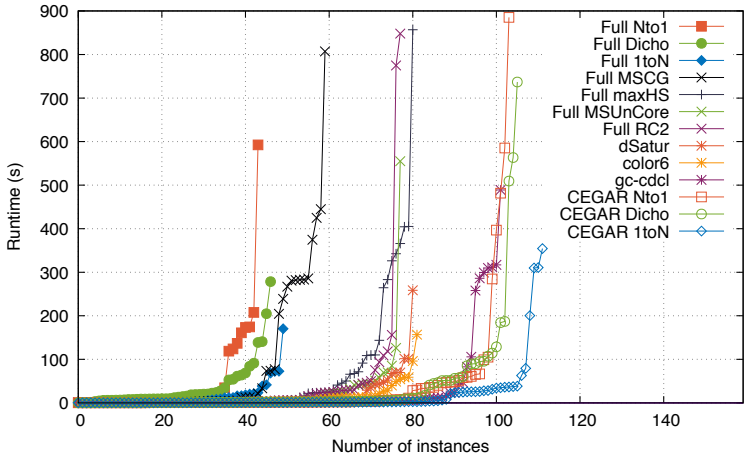

Fig. 3. Cactus-plot of the runtime.

our full encoding are not very effective. Even if the methods using MaxSAT solvers are more efficient than the one that computes the bound, our best version is no more effective than the weaker state-of-the-art approach. Actually, having a closer look at their behaviour, the lack of efficiency does not come from SAT solver, but is due to the encoding itself. 97.3% of the translations have been solved. Hence, there is definitely a bottleneck here due to the translation.

One can observe in Figure.3, that CEGAR approaches outperform state-of-the-art approaches. They manage to solve more instances than `gc-cdcl`, which solves 102 instances out of 159, which was definitely the best overall approach, as explained in [13]. It is important to note that the results do not depend on the way the optimisation problem is solved: the three types of search perform better than `gc-cdcl`. We can also note that, either in Full or in CEGAR mode, it seems that the 1toN approaches perform better than their Nto1 and Dicho counterparts. this can be explained because the chromatic number is generally far from the number of vertices. Thus, it is better to start from 1 than to start from the number of vertices. Table 1 reports the results regarding the number of problems solved depending on the family of the instance under consideration. As we can see, CEGAR approaches work well on all the families except for the `other` and `random` categories, where `color6` outperforms it. On the random benchmarks, the six unsolved instances are due to the SAT solving phase. These problems seem to have a *random nature* for which CDCL SAT solvers are ill-suited, whereas `color6` seems to deal with them extremely well. In the `other` category, it is more sparse, we do not lose on one big category but few instances here and there, except somewhat for the `school` one, which represents Class Scheduling Problems. All the state-of-the-art approaches solve these instances except us. In our case, it seems that verifying solutions with the checker is time-consuming. It returns only a few triples each time, therefore lacking time to solve the instance.

We also report a comparison between `gc-cdcl` and CEGAR 1toN (now denoted Picasso) in Fig.4. Each dot corresponds to a colouring instance. The x-axis of the figure represents the computation time needed to compute the chromatic number when

	Full			MaxSAT				dSatur	color6	gc-cdcl	CEGAR		
	Nto1	Dicho	1toN	MSCG	msuncore	maxHS	RC2				Nto1	Dicho	1toN
DSJ (15)	3	3	3	4	4	**6**	4	**6**	2	4	3	3	4
fpsol2 (3)	0	0	0	0	0	0	0	**3**	2	**3**	**3**	**3**	**3**
inithx (3)	0	0	0	0	0	0	0	**3**	1	**3**	**3**	**3**	**3**
le450 (12)	0	0	0	0	0	0	0	4	4	8	**10**	**10**	**10**
mulsol (5)	4	4	4	4	**5**	**5**	**5**	**5**	4	**5**	**5**	**5**	**5**
book (5)	**5**	**5**	**5**	**5**	**5**	**5**	**5**	2	3	**5**	**5**	**5**	**5**
miles (5)	2	2	2	1	1	1	1	**5**	4	**5**	4	4	**5**
queen (13)	5	8	8	5	**13**	**13**	**13**	7	11	10	**13**	**13**	**13**
myciel (5)	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
mugg (4)	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
insertion (25)	8	8	8	10	12	12	12	5	7	15	15	15	**19**
wap (8)	0	0	0	0	0	0	0	0	0	1	**3**	**3**	**3**
qg (4)	0	0	0	0	0	0	0	1	0	**3**	**3**	**3**	**3**
random (18)	2	2	5	12	12	13	14	14	**16**	6	10	10	10
other (45)	6	6	6	10	17	17	15	18	**25**	**25**	18	20	20
Total (159)	44	47	50	60	78	81	78	81	82	102	104	106	**112**

Table 1. Number of instances of each sub-family solved by the approaches in consideration, in **bold** the best approach for each sub-family.

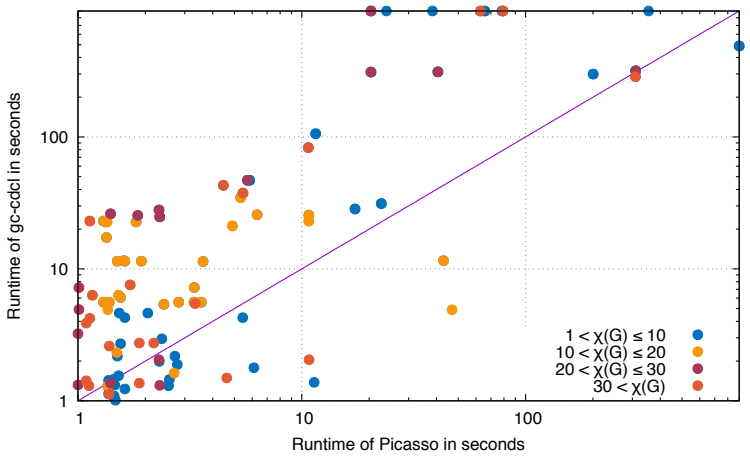

Fig. 4. Picasso vs. `gc-cdcl`.

Picasso is considered, while the y-axis depicts the time needed to compute the chromatic number when `gc-cdcl` is considered. As we can see, there is a clear trend above the diagonal, especially on the instances solved in less than 10 seconds by Picasso which require almost a hundred seconds for `gc-cdcl`. Moreover, it seems that it does not depend on the chromatic number that needs to be found. Indeed, one could think that the higher the chromatic number is, more CEGAR loops in Picasso and therefore the worst the overall performances. However, as depicted in Fig.4 the chromatic-number of the instance does not have much influence. Actually, the models returned by glucose help to refine quickly the bound and therefore provide a tremendous speed-up in comparison to a simple increasing loop. This is a result already known for people working with

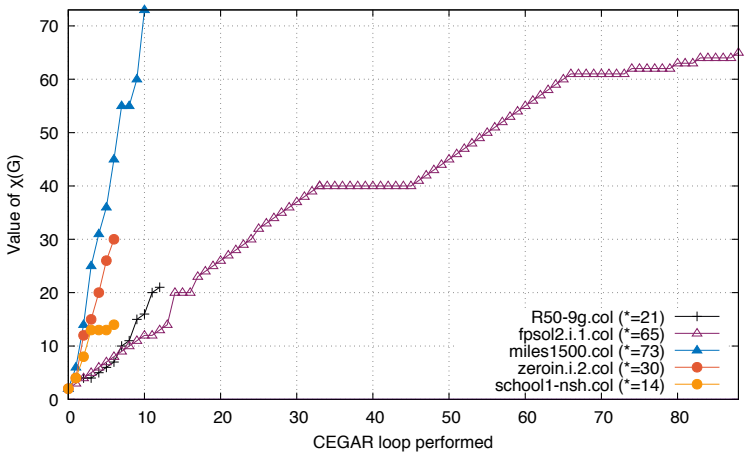

Fig. 5. Progression of Picasso towards the chromatic number of instances.

NP-oracles: an oracle able to output a model can provide to provide a large speed-up compared to one answering only yes or no [39]. Indeed, there are problem solvable with a polynomial number of calls to an oracle that can be solved with a logarithmic number of calls to an oracle outputing a model.

6.2 Analysis on the CEGAR behaviour

Now that we have analysed how good Picasso in CEGAR mode is, let us see how it is behaving exactly, *i.e.* let us determine where is the bottleneck in our approach. First, let us take a look at the number of CEGAR loops that Picasso is performing. We got that:

$$min = 2;\ 1^{st}qu. = 3;\ median = 6;\ mean = 10.3;\ 3^{rd}qu. = 11;\ max = 88$$

This shows that Picasso does not loop so many times. Fig.5 reports for some categories of benchmarks the number of CEGAR loops (x-axis) with respect to the chromatic number of the instance (y-axis). One can observe on this picture that the instances have generally a chromatic number higher than the number of CEGAR loops. The sole exception is `fpsol2.i.1.col` which has a chromatic number of 65, where we identified that, at each step, the model does not provide any information on the chromatic number. The models returned by the SAT solver help to quickly improve the bound. This implies that in many cases, `checker` provides a quite precise refinement and not just one spurious triple at a time.

Finally, we report in Table 2 the cumulative time spent by the different phases of CEGAR (encoding, checking and solving) with respect to the chromatic number of the instance considered. One can observe that, on the instances that we managed to solve, the SAT encoding is not really time-consuming, neither is the SAT solving phase. Except for a few cases, we can observe that the checking phase definitely is the bottleneck of the approach. This is not really surprising. Indeed, Glucose [32] is a very efficient SAT solver, and we used the Cardinality Network Encoding from `open-wbo` [40] which

Time (s)	Encoding			Checking			Solving		
	min	med	max	min	med	max	min	med	max
$1 < \chi(G) \leq 10$	0.00	3.99	32.34	0.01	15.60	152.45	0.00	3.20	72.35
$10 < \chi(G) \leq 20$	0.01	4.11	45.94	0.01	13.67	161.97	0.00	3.11	83.32
$20 < \chi(G) \leq 30$	0.02	4.24	79.30	0.01	12.90	163.41	0.00	1.27	105.23
$30 < \chi(G)$	0.01	5.60	71.66	0.01	17.92	222.54	0.00	2.45	95.66

Table 2. Time distribution details for the three phases in Picasso

allows us to refine the bound by adding as few clauses as possible. Therefore, the only time-consuming task is for each satisfiable answer from Glucose to check the model returned and determine whether clauses must be added. The median times may look relatively low, however, the reader must keeping in mind Fig.3. Indeed, most of the instances solved by Picasso are solved in less than 100 seconds. If we have a look at the instances for which a time-out was reached, it turns out that we spent in median 89.3% of the time to check models. To be convinced that the checker is indeed a bottleneck, we tried to implement a naive one, which consists in testing whether for all i,j and k we have that the constraints $\texttt{transitivity}(i, j, k) \wedge \texttt{euclideanity}(i, j, k)$ are respected. With such a checker, *CEGAR* 1*toN* solves 100 instances, *CEGAR Dicho* solves 96 and *CEGAR Nto*1 solves 95 instances.

7 Conclusion

In this paper, we proposed a new approach for solving the minimal graph colouring problem using an under-abstraction refinement approach within the CEGAR framework. We showed that our encoding is sound and complete and we implemented our approach in the solver Picasso. We compared our solver with the state-of-the-art solvers for the graph colouring problem, on a wide range of benchmarks of different size and difficulty. We conclude that a basic direct-encoding approach is not competitive; many of the available benchmarks are large and require a lot of clauses to be encoded. However, by considering clauses carefully, our CEGAR approach outperforms the other solvers on most of the benchmarks. As future we plan to avoid checking unmodified sub-graphs twice by flagging some nodes, *i.e.*, checking only the part which was modified due to the previous assignment. Moreover, extending the CEGAR approach into a RECAR (Recursively Explore and Check Abstraction Refinement) one [41] could be interesting. The unit propagation of the embedded SAT solver would be stronger, and it could provide us quickly a good upper-bound. Such double-abstraction functions could make the binary search much faster and improve the overall performance. This is an exciting perspective for future work.

Acknowledgements

Part of this work was supported by the French Ministry for Higher Education and Research, the Haut-de-France Regional Council through the "Contrat de Plan État Région (CPER) DATA" and by the IDEX UCA[JEDI].

References

1. Marx, D.: Graph Colouring Problems and Their Applications in Scheduling. Periodica Polytechnica Electrical Engineering (Archives) **48**(1-2) 11–16
2. Lewis, R.M.R.: A Guide to Graph Colouring - Algorithms and Applications. Springer (2016)
3. Chaitin, G.J.: Register allocation and spilling via graph coloring (with retrospective). In McKinley, K.S., ed.: 20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation 1979-1999, A Selection, ACM (1982) 66–74
4. Lewis, R., Thompson, J.M.: On the Application of Graph Colouring Techniques in Round-Robin Sports Scheduling. Computers & OR **38**(1) (2011) 190–204
5. Hussin, B., Basari, A.S.H., Shibghatullah, A.S., Asmai, S.A., Othman, N.S.: Exam Timetabling Using Graph Colouring Approach. In: 2011 IEEE Conference on Open Systems. (2011) 133–138
6. Gelder, A.V.: Another Look at Graph Coloring via Propositional Satisfiability. Discrete Applied Mathematics **156**(2) (2008) 230–243
7. Schaafsma, B., Heule, M., van Maaren, H.: Dynamic Symmetry Breaking by Simulating Zykov Contraction. In Kullmann, O., ed.: Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Volume 5584 of Lecture Notes in Computer Science., Springer (2009) 223–236
8. Caramia, M., Dell'Olmo, P.: Coloring Graphs by Iterated Local Search Traversing Feasible and Infeasible Solutions. Discrete Applied Mathematics **156**(2) (2008) 201–217
9. Dowsland, K.A., Thompson, J.M.: An Improved Ant Colony Optimisation Heuristic for Graph Colouring. Discrete Applied Mathematics **156**(3) (2008) 313–324
10. Galinier, P., Hertz, A., Zufferey, N.: An Adaptive Memory Algorithm For The k-Coloring Problem. Discrete Applied Mathematics **156**(2) (2008) 267–279
11. Brélaz, D.: New Methods to Color the Vertices of a Graph. Commun. ACM **22**(4) (1979) 251–256
12. Zhou, Z., Li, C.M., Huang, C., Xu, R.: An Exact Algorithm with Learning for the Graph Coloring Problem. Computers & OR **51** (2014) 282–301
13. Hebrard, E., Katsirelos, G.: Clause Learning and New Bounds for Graph Coloring. In Hooker, J.N., ed.: Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings. Volume 11008 of Lecture Notes in Computer Science., Springer (2018) 179–194
14. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: CounterExample-Guided Abstraction Refinement For Symbolic Model Checking. Journal of ACM **50**(5) (2003)
15. Brummayer, R., Biere, A.: Effective Bit-Width and Under-Approximation. In Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A., eds.: Computer Aided Systems Theory - EUROCAST 2009, 12th International Conference, Las Palmas de Gran Canaria, Spain, February 15-20, 2009, Revised Selected Papers. Volume 5717 of Lecture Notes in Computer Science., Springer (2009) 304–311
16. Seipp, J., Helmert, M.: Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. J. Artif. Intell. Res. **62** (2018) 535–577
17. Soh, T., Le Berre, D., Roussel, S., Banbara, M., Tamura, N.: Incremental SAT-Based Method with Native Boolean Cardinality Handling for the Hamiltonian Cycle Problem. In Fermé, E., Leite, J., eds.: Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings. Volume 8761 of Lecture Notes in Computer Science., Springer (2014) 684–693
18. Glorian, G., Lagniez, J., Montmirail, V., Sioutis, M.: An Incremental SAT-Based Approach to Reason Efficiently on Qualitative Constraint Networks. In Hooker, J.N., ed.: Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille,

France, August 27-31, 2018, Proceedings. Volume 11008 of Lecture Notes in Computer Science., Springer (2018) 160–178
19. Lagniez, J., Le Berre, D., de Lima, T., Montmirail, V.: An Assumption-Based Approach for Solving the Minimal S5-Satisfiability Problem. In Galmiche, D., Schulz, S., Sebastiani, R., eds.: Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings. Volume 10900 of Lecture Notes in Computer Science., Springer (2018) 1–18
20. Hooker, J.N.: Logic-Based Methods for Optimization. In Borning, A., ed.: Principles and Practice of Constraint Programming, Second International Workshop, PPCP'94, Rosario, Orcas Island, Washington, USA, May 2-4, 1994, Proceedings. Volume 874 of Lecture Notes in Computer Science., Springer (1994) 336–349
21. Chu, Y., Xia, Q.: A Hybrid Algorithm for a Class of Resource Constrained Scheduling Problems. In Barták, R., Milano, M., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings. Volume 3524 of Lecture Notes in Computer Science., Springer (2005) 110–124
22. Hooker, J.N.: A Hybrid Method for the Planning and Scheduling. Constraints **10**(4) (2005)
23. Tran, T.T., Beck, J.C.: Logic-based Benders Decomposition for Alternative Resource Scheduling with Sequence Dependent Setups. In Raedt, L.D., Bessière, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., Lucas, P.J.F., eds.: ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012. Volume 242 of Frontiers in Artificial Intelligence and Applications., IOS Press (2012) 774–779
24. de Moura, L.M., Rueß, H., Sorea, M.: Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. In Voronkov, A., ed.: Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings. Volume 2392 of Lecture Notes in Computer Science., Springer (2002) 438–455
25. Ji, X., Ma, F.: An Efficient Lazy SMT Solver for Nonlinear Numerical Constraints. In Reddy, S., Drira, K., eds.: 21st IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2012, Toulouse, France, June 25-27, 2012, IEEE Computer Society (2012) 324–329
26. Gebremedhin, A.H., Manne, F., Pothen, A.: What Color Is Your Jacobian? Graph Coloring for Computing Derivatives. SIAM Review **47**(4) (2005) 629–705
27. Zuckerman, D.: Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. Theory of Computing **3**(1) (2007) 103–128
28. Tseitin, G.S. In: On the Complexity of Derivation in Propositional Calculus. Springer (1983)
29. Gent, I.P., Petrie, K.E., Puget, J.: Symmetry in Constraint Programming. In: Handbook of Constraint Programming. (2006) 329–376
30. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality Networks: a Theoretical And Empirical Study. Constraints **16**(2) (2011) 195–221
31. Roney-Dougal, C.M., Gent, I.P., Kelsey, T., Linton, S.: Tractable symmetry breaking using restricted search trees. In de Mántaras, R.L., Saitta, L., eds.: Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004, IOS Press (2004) 211–215
32. Audemard, G., Lagniez, J., Simon, L.: Improving Glucose for Incremental SAT Solving with Assumptions: Application to MUS Extraction. In Järvisalo, M., Gelder, A.V., eds.: Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings. Volume 7962 of Lecture Notes in Computer Science., Springer (2013) 309–317

33. Audemard, G., Simon, L.: Predicting Learnt Clauses Quality in Modern SAT Solvers. In Boutilier, C., ed.: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. (2009) 399–404
34. Davies, J., Bacchus, F.: Exploiting the Power of MIP Solvers in MaxSAT. In Järvisalo, M., Gelder, A.V., eds.: Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings. Volume 7962 of Lecture Notes in Computer Science., Springer (2013) 166–181
35. Morgado, A., Ignatiev, A., Marques-Silva, J.: MSCG: Robust Core-Guided MaxSAT Solving. JSAT **9** (2014) 129–134
36. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: A Python Toolkit for Prototyping with SAT Oracles. In Beyersdorff, O., Wintersteiger, C.M., eds.: Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings. Volume 10929 of Lecture Notes in Computer Science., Springer (2018) 428–437
37. Heras, F., Morgado, A., Marques-Silva, J.: Core-Guided Binary Search Algorithms for Maximum Satisfiability. In Burgard, W., Roth, D., eds.: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011, AAAI Press (2011)
38. Fahiem Bacchus and Matti Järvisalo and Ruben Martins: Max-SAT 2018: Thirteen Max-SAT Evaluation. https://maxsat-evaluations.github.io/2018/ (2018)
39. Janota, M., Marques-Silva, J.: On the Query Complexity of Selecting Minimal Sets for Monotone Predicates. Artif. Intell. **233** (2016) 73–83
40. Martins, R., Manquinho, V.M., Lynce, I.: Open-WBO: A Modular MaxSAT Solver. In Sinz, C., Egly, U., eds.: Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. Volume 8561 of Lecture Notes in Computer Science., Springer (2014) 438–445
41. Lagniez, J., Le Berre, D., de Lima, T., Montmirail, V.: A Recursive Shortcut for CEGAR: Application To The Modal Logic K Satisfiability Problem. In Sierra, C., ed.: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org (2017) 674–680