Mathematical Programming manuscript No.
(will be inserted by the editor)

Graph Coloring with Decision Diagrams

Willem-Jan van Hoeve

Received: date / Accepted: date

Abstract We introduce an iterative framework for solving graph coloring problems using decision diagrams. The decision diagram compactly represents all possible color classes, some of which may contain edge conflicts. In each iteration, we use a constrained minimum network flow model to compute a lower bound and identify conflicts. Infeasible color classes associated with these conflicts are removed by refining the decision diagram. We prove that in the best case, our approach may use exponentially smaller diagrams than exact diagrams for proving optimality. We also develop a primal heuristic based on the decision diagram to find a feasible solution at each iteration. We provide an experimental evaluation on all 137 DIMACS graph coloring instances. Our procedure can solve 52 instances optimally, of which 44 are solved within 1 minute. We also compare our method to a state-of-the-art graph coloring solver based on branch-and-price, and show that we obtain competitive results. Lastly, we report an improved lower bound for the open instance `C2000.9`.

Keywords Graph coloring · Decision diagrams · Integer programming

1 Introduction

Graph coloring is a fundamental combinatorial optimization problem that asks to color the vertices of a given graph with a minimum number of colors, such that adjacent vertices are colored differently. Graph coloring is a core component of many applications, in particular those related to timetabling or scheduling [35,20,3,22]. The most efficient exact solution techniques include enumerative methods based on the Dsatur vertex ordering [30,10,28,32,14], methods based on integer linear programming formulations [25,26,19], constraint programming [16], and column generation (branch-and-price) [24,23,17,16,27].

Willem-Jan van Hoeve
Carnegie Mellon University, Tepper School of Business
5000 Forbes Avenue, Pittsburgh, PA 15232, USA
E-mail: vanhoeve@andrew.cmu.edu

A major challenge for exact graph coloring methods is finding strong lower bounds to help accelerate the proof of optimality. A natural lower bound is the clique number of a graph—the size of the largest complete subgraph, which requires all its vertices to be colored differently. In this work, we explore an alternative approach that does not directly rely on maximal cliques, but instead makes use of relaxed decision diagrams [2]. Relaxed decision diagrams provide a graphical discrete relaxation of a solution set and can be used to derive optimization bounds [2,9,7] as well as exact solution methods [8,5].

For the graph coloring problem, we let the decision diagram compactly represent the collection of independent sets of the input graph, each of which corresponds to a color class (a subset of vertices with the same color). We obtain a graph coloring lower bound by solving a constrained minimum network flow over the decision diagram that ensures that each vertex only appears in one color class. However, in our relaxed decision diagram, not all color classes may be exact. We therefore identify conflicts (adjacent vertices) in each color class, which are subsequently removed from the diagram by a refinement step. We iteratively apply this process until no more conflicts are found. We show that an integral conflict-free solution to the constrained minimum network flow problem is guaranteed to be optimal. We also develop a primal heuristic that is based on the network flow solution.

Our approach is relatively generic, in that the iterative refinement process based on conflicts is not restricted to graph coloring problems. For example, we can define a very similar procedure for bin packing problems, in which case a subset of items is conflicting if its weight exceeds the capacity of the bin. In fact, it can be viewed as a 'dual' form of column generation: instead of iteratively generating new columns (color classes), our approach iteratively removes infeasible color classes from consideration. In both cases, however, a solution is defined as a subset of color classes.

Contributions. The main contributions of this work include 1) the introduction of a new framework for obtaining graph coloring lower and upper bounds based on relaxed decision diagrams, and proving its correctness, 2) a proof that relaxed decision diagrams can be exponentially smaller than their exact versions for finding optimal solutions, and 3) an experimental evaluation of our framework on the DIMACS graph coloring benchmark, demonstrating that our approach is competitive with a state-of-art exact graph coloring solver based on branch-and-price, and reporting an improved lower bound for instance `C2000.9`.

We note that a preliminary version of this work appeared as an extended abstract in IPCO 2020 [18], which focused primarily on computing lower bounds. The present version introduces a primal heuristic, includes full proofs, provides more details of the implementation, and presents an extensive computational evaluation on all 137 DIMACS benchmark instances.

2 Graph Coloring by Independent Sets

We first present a formal definition of graph coloring [33]. Let $G = (V,E)$ be an undirected simple graph with vertex set V and edge set E. We define $n = |V|$ and $m = |E|$. We denote by N_i the set of neighbors of $i \in V$. For convenience, we label the

vertices V as integers $\{1,\ldots,n\}$. A *vertex coloring* of G is a mapping of each vertex to a color such that adjacent vertices are assigned different colors. We refer to the subset of vertices with the same color as a *color class*. The graph coloring problem is to find a vertex coloring with the minimum number of colors. The minimum number of colors to color G is called the coloring number or the *chromatic number* of G, denoted by $\chi(G)$.

Observe that each color class is defined as a subset of vertices that are pairwise non-adjacent. In other words, a color class corresponds to an *independent set* of G, and conversely each independent set of G can be used as a color class. This allows to formulate the graph coloring problem as follows. Let I be the collection of all independent sets of G. We introduce a binary variable z_i for each independent set $i \in I$, representing whether i is used as a color class in a solution. We let binary parameter a_{ij} represent whether vertex $j \in V$ belongs to independent set $i \in I$ ($a_{ij} = 1$) or not ($a_{ij} = 0$). The graph coloring problem can then be formulated as the following integer program:

$$\begin{aligned} \chi(G) = \min & \sum_{i \in I} z_i \\ \text{s.t.} & \sum_{i \in I} a_{ij} z_i = 1 \ \forall j \in V, \\ & z_i \in \{0,1\} \quad \forall i \in I, \end{aligned} \tag{1}$$

where the equality constraint ensures that each vertex belongs to one color class. This formulation forms the basis of the column generation approaches for graph coloring, as first proposed by Mehrotra and Trick [24]. Instead of enumerating all exponentially many independent sets I, column generation iteratively adds new independent sets (with negative reduced cost) to an initial collection. In our approach, we instead start with compactly representing all subsets of V, and iteratively remove sets that contain adjacent vertices.

Notation: An edge $\{i, j\} \in E$ is an unordered pair of vertices. In what follows, we will denote an edge $\{i, j\}$ by either one of its equivalent ordered definitions (i, j) or (j, i). This notational distinction is convenient because our edge conflict separation procedure requires an ordering of the vertices.

3 Decision Diagrams

Decision diagrams were originally developed to represent switching circuits and, more generally, Boolean functions [21,1,34]. They became particularly popular after the introduction of efficient compilation methods for Reduced Ordered Binary Decision Diagrams [11,12], and have been applied widely to verification and configuration problems. More recently, decision diagrams have been applied to solve optimization problems [5], which is the context we follow in this paper.

3.1 Definitions

For our purposes, a decision diagram will represent the set of solutions to an optimization problem P defined on an ordered set of decision variables $X = \{x_1,\ldots,x_n\}$.

In this paper, we assume that each variable is binary. The feasible set of P is denoted by $\mathrm{Sol}(P)$.

A *decision diagram* for P is a layered directed acyclic graph $D = (N, A)$ with node set N and arc set A. D has $n+1$ layers that represent state-dependent decisions for the variables. The first layer (layer 1) is a single root node r, while the last layer (layer $n+1$) is a single terminal node t. Layer j is a collection of nodes associated with variable $x_j \in X$, for $j = 1, \ldots, n$. Arcs are directed from a node u in layer j to a node v in layer $j+1$, and have an associated label $\ell(u,v)$ which can be either 0 or 1. We refer to the former as 0-arcs and to the latter as 1-arcs. For each node, the outgoing 0-arc and 1-arc must be unique. The layer of node u is denoted by $L(u)$. Each arc, and each node, must belong to a path from r to t. Each arc-specified r-t path $p = (a_1, a_2, \ldots, a_n)$ defines a variable assignment by letting $x_j = \ell(a_j)$ for $j = 1, \ldots, n$. We slightly abuse notation and denote by $\mathrm{Sol}(D)$ the collection of variable assignments defined by all r-t paths in D.

Definition 1 A decision diagram D is *exact* for problem P if $\mathrm{Sol}(D) = \mathrm{Sol}(P)$. A decision diagram D is *relaxed* for problem P if $\mathrm{Sol}(D) \supseteq \mathrm{Sol}(P)$.

The benefit of using decision diagrams for representing solutions is that *equivalent* nodes, i.e., nodes with the same set of completions, can be merged. A decision diagram is called *reduced* if no two nodes in a layer are equivalent. A key property is that for a given fixed variable ordering, there exists a *unique* reduced ordered decision diagram [11]. Nonetheless, even reduced decision diagrams may be exponentially large to represent all solutions for a given problem.

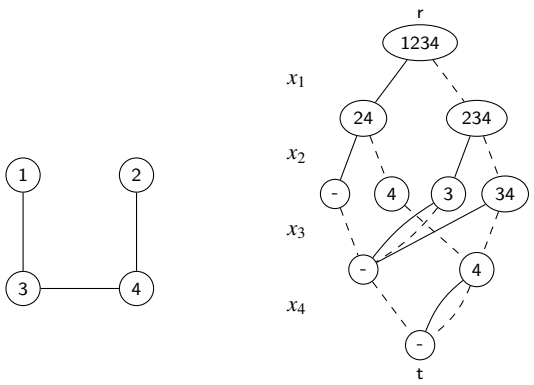
3.2 Exact Compilation

In this work we apply top-down compilation methods that depend on state-dependent information (similar to state variables in dynamic programming models). We limit our exposition to the compilation of decision diagrams for independent set problems, as proposed by Bergman et al. [6,7]. We define a binary variable x_i for each $i \in V$ representing whether i is selected. The state information we maintain is the set of 'eligible vertices', i.e., the set of graph vertices that can be added to the independent sets represented by paths into the node.

Formally, for each node u in the decision diagram we recursively define a set $S(u) \subseteq V$, and we initialize $S(r) = V$. For node u in layer $L(u) = j$ we distinguish two cases. If $j \notin S(u)$, we define a 0-arc (or transition) from u to v, with $S(v) = S(u)$. Otherwise, if $j \in S(u)$, we define both a 1-arc and a 0-arc out of u, with

$$S(v) = \begin{cases} S(u) \setminus (\{j\} \cup N_j) & \text{if } (u,v) \text{ is a 1-arc,} \\ S(u) \setminus \{j\} & \text{if } (u,v) \text{ is a 0-arc.} \end{cases} \tag{2}$$

The top-down compilation procedure starts at the root node, creates all nodes in the next layer (following the 0-arcs and 1-arcs), and merges the nodes that are equivalent. Bergman et al. [6,7] showed that for independent sets, the state information $S(u)$ (the set of eligible vertices) suffices to determine node equivalence, i.e., two nodes u



a. Input graph b. Exact decision diagram

Fig. 1 Input graph for Example 1 (a) and the associated exact decision diagram representing all independent sets (b). The diagram uses the lexicographic ordering of the vertices. Dashed arcs represent 0-arcs, while solid arcs represent 1-arcs. For convenience, the set of eligible vertices (the state information) is given in each node.

and v are equivalent if and only if $S(u) = S(v)$. This top-down compilation procedure therefore yields the unique reduced decision diagram for representing all independent sets (for a given variable ordering), as shown in [6,7].

Example 1 Consider the graph in Figure 1.a. We depict the exact decision diagram representing all independent sets for this graph in Figure 1.b.

3.3 Compilation by Separation

As an alternative to exact compilation, we can apply *constraint separation* to iteratively construct a decision diagram [13,5]. In general, the input to the constraint separation problem is a relaxed decision diagram, and a tuple representing a (partial) variable assignment that violates a constraint. The purpose is to identify and eliminate those paths in the decision diagram that correspond to the given tuple [13]. In our case, we deviate from the literature in that we wish to separate an edge conflict *along a prescribed path* in the decision diagram. This results in a more 'lazy' implementation of constraint separation that will hopefully result in smaller decision diagrams. We do apply a similar technique as [13], however. That is, the path is separated by introducing a new node in each layer of the diagram, following the arc labels prescribed by the path. In the last layer that the path traverses, we forbid the prescribed arc label, thus eliminating the assignment. We will next describe this procedure in more detail, in the context of representing independent sets.

As we will work with *relaxed* decision diagrams, we need to define an initial diagram D. We start with the simplest possible diagram, having only one node per layer. As before, we can recursively define the diagram. That is, we initialize the root node state as $S(r) = V$. For node u in layer $L(u) = j$, we again define a 0-arc and a 1-arc. This time, however, we merge their endpoints and connect both arcs to the

node in layer $j+1$. Doing so, the state of node u is $S(u) = \{j,\ldots,n\}$, for $j = 1,\ldots,n$, and $S(t) = \varnothing$. Since the resulting diagram represents all 2^n binary n-tuples, it also contains all independent sets for the graph under consideration, and is therefore a valid relaxed decision diagram.

The input to our separation algorithm is:

- a reduced decision diagram D, i.e., no layer contains multiple equivalent nodes,
- an edge conflict $(j,k) \in E$, where $j < k$,
- a path $u_j, u_{j+1}, \ldots, u_{k-1}$ with arc labels $l_j, l_{j+1}, \ldots, l_{k-1}$ associated with the conflict (j,k), and containing no edge conflicts (j',k') such that $j \leq j' < k' < k$.

The goal of the separation algorithm is to resolve the conflict along the path by splitting nodes, and arcs, appropriately. This is described in Algorithm 1.

In the algorithm, we represent D as a two-dimensional vector $\mathsf{D}[\,][\,]$ of nodes, such that D is a vector of 'layers' and $\mathsf{D}[\,]$ is a vector of 'nodes', one for each layer. Both are indexed starting from 1. The size of vector D is fixed to $n+1$, while we dynamically update the size of the layers $\mathsf{D}[\,]$. The root is represented as $\mathsf{D}[1][1]$ and the terminal as $\mathsf{D}[n+1][1]$. Each node $u = \mathsf{D}[\mathsf{j}][\mathsf{i}]$ (u is the i-th node in layer j) has state information $\mathsf{D}[\mathsf{j}][\mathsf{i}].\mathsf{S} = S(u)$, a reference to the node in layer $j+1$ that represents the endpoint of its 1-arc $\mathsf{D}[\mathsf{j}][\mathsf{i}].\mathsf{oneArc}$, and a similar reference for its 0-arc $\mathsf{D}[\mathsf{j}][\mathsf{i}].\mathsf{zeroArc}$. If the 1-arc does not exist, the reference holds value -1.

Algorithm 1 considers each node $\mathsf{D}[i][u_i]$ along the path in sequence (line 3) and splits off the next node in the path. This is done by first creating a temporary node w (lines 4-6). If an equivalent node already exists in layer $i+1$, we direct the path to its index (lines 7-8). Otherwise we complete the definition of w by copying the outgoing arcs of node $\mathsf{D}[i+1][u_{i+1}]$ (lines 10-12), and we add w to layer $i+1$ (lines 13-14). Lastly, we redirect the path from u_i to the new node with index t (lines 16-18).

Example 2 Figure 2 gives an illustration of Algorithm 1. When we apply the algorithm to the decision diagram (a), with the all-ones path and conflicting edge $(1,3)$ as input, we obtain the decision diagram (b).

The following lemmas characterize the outcome of Algorithm 1. The first lemma states that the algorithm soundly removes conflicts from paths in the decision diagram:

Lemma 1 *Let decision diagram D, edge conflict (j,k), and node-label specified path $(u_j, l_j, u_{j+1}, l_{j+1}, \ldots, u_{k-1}, l_{k-1})$ be the input to Algorithm 1. The application of the algorithm results in a decision diagram in which the arc-specified path $(l_j,\ldots,l_{k-1},1)$ starting at u_j no longer exists, but $(l_j,\ldots,l_{k-1},0)$ does.*

Proof We start with the observation that the first label on the path (i.e., l_j) must be 1, since otherwise there cannot be a conflict with node k. Therefore, when we start the loop in line 3 for $i = j$, the creation of new node w (line 4) and the elimination of N_i (line 5) removes $k \in N_i$ from $S(w)$. In later iterations, i.e., for $i > j$, node w inherits its state from its parent node (line 5), and can therefore never include k.

Note that the state update in line 6 eliminates N_i from $S(w)$ when $l_i = 1$, which may potentially eliminate 1-arcs from the given path. However, since no conflicts

Algorithm 1: Separating edge conflict (j,k) in decision diagram D.

1 **input:** reduced decision diagram D (D[i][j] represents the jth node in layer i), path node indices $u_j,\dots,u_{k-1}$, path arc labels $l_j,\dots,l_{k-1}$, conflict (j,k) (it is assumed that the path contains no edge conflicts (j',k') such that $j \le j' < k' < k$), and list of neighbors N_i for each $i \in V$
2 **output:** reduced decision diagram in which the conflict along the path has been eliminated
3 **for** $i = j,\dots,k-1$ **do**
4 create node w // split the path towards node w
5 w.S $\leftarrow$ D[i][u_i].S $\setminus \{i\}$ // copy the parent state and remove i
6 **if** $l_i = 1$ **then** w.S $\leftarrow w$.S $\setminus N_i$ // remove N_i in case of 1-arc
7 $t \leftarrow -1$ // t is index of the new node in layer $i+1$
8 **if** $\exists k$ *such that* D[$i+1$][k].S = w.S **then** $t \leftarrow k$ // check for equivalent node
9 **else**
10 **if** $i+1 \in w$.S **then** w.oneArc $\leftarrow$ D[$i+1$][u_{i+1}].oneArc // copy 1-arc
11 **else** w.oneArc $\leftarrow -1$
12 w.zeroArc $\leftarrow$ D[$i+1$][u_{i+1}].zeroArc // copy 0-arc
13 D[$i+1$].add(w) // append w as new node to layer $i+1$
14 $t \leftarrow$ |D[$i+1$]| // update t to last index of layer $i+1$
15 **if** $l_i = 1$ **then** D[i][u_i].oneArc $\leftarrow t$ // re-direct path in case of 1-arc
16 **else** D[i][u_i].zeroArc $\leftarrow t$ // re-direct path in case of 0-arc
17 $u_{i+1} \leftarrow t$ // update path index

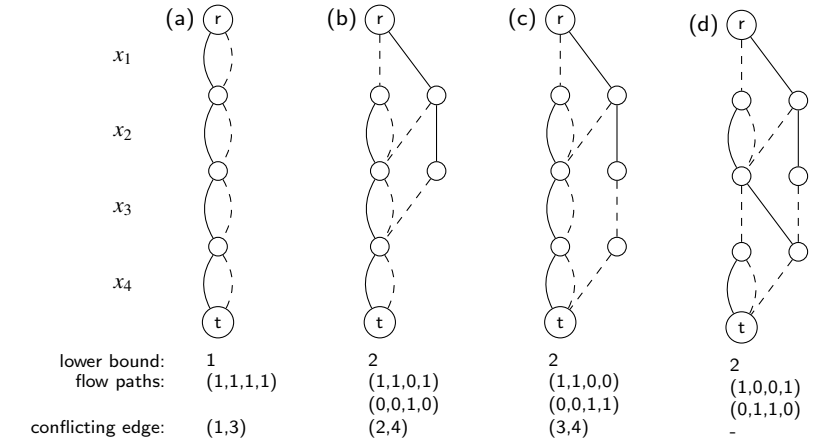


Fig. 2 Applying constraint separation to the input graph of Example 1. The initial relaxed diagram (a) is iteratively refined until the optimal solution (the flow paths) no longer contains infeasible color classes.

(j',k') with $j \le j' < k' < k$ appear on the path, no 1-arc other than the last one will be removed from the path. As a consequence, at each iteration i of the algorithm, there exists an arc a along the arc-specified path with arc label $\ell(a) = l_i$. Merging w into an equivalent node (line 8) does not influence the existence of the arc label l_i at iteration i.

When $i = k-1$, since k does not appear in $S(w)$, the 1-arc out of w is eliminated. As a result, only the 0-arc is connected to the child node of u_{k-1}. Therefore, upon termination there is no more path starting at u_k with arc labels $(l_1,\dots,l_{k-1},1)$, but there exists a path with labels $(l_1,\dots,l_{k-1},0)$. □

The following lemma shows that the set of solutions (paths) represented by the diagram becomes strictly smaller after applying the algorithm:

Lemma 2 *Algorithm 1 does not introduce new arc-specified paths to the input decision diagram.*

Proof New paths can only be introduced by re-directing arcs or by merging non-equivalent nodes. The node splitting process redirects the path from u_i to the new node w but maintains the original paths since the outgoing arc(s) are copied (lines 10-12). Furthermore, the algorithm only merges equivalent nodes (line 8). □

Lemma 3 *Given a reduced decision diagram as input, Algorithm 1 produces a reduced decision diagram.*

Proof The state representation $S(u)$ for node u suffices to determine state equivalence in each layer, and equivalent nodes are merged (line 8). Therefore, if the input diagram is reduced, so is the resulting diagram. □

The next lemma shows that the algorithm can only increase the size of the diagram linearly. Recall that the n vertices of the input graph are labeled $\{1,\dots,n\}$ and correspond to the layer indices of the decision diagram.

Lemma 4 *Given edge conflict (j,k) with $j < k$ as input, Algorithm 1 increases the size of the relaxed decision diagram by at most $k-j$ nodes.*

Proof In each iteration only one node is created (lines 3-4), and there are $k-j$ iterations. □

We note that a similar result was first presented by Cire and Hooker [13, Theorem 2]: separating a given (partial) variable assignment from a decision diagram can increase each layer by at most one node. As an extension, Perez and Régin [29] describe an algorithm that separates multiple variable assignments (of a specific form) at once.

Lemma 4 is an appealing property, because separating k conflicts will only increase the size of the diagram by at most kn nodes. However, iterative application of the algorithm may require separating the same edge multiple times, over distinct paths, which may ultimately lead to an exponential number of conflicts to be separated.

Theorem 1 *Given a reduced decision diagram D as input, and an oracle that provides us with edge conflicts and associated paths in D, repeated application of Algorithm 1 results in the unique exact decision diagram.*

Proof This follows from Lemma 3 and repeated application of Algorithm 1. □

In Section 5 we will present an iterative refinement procedure that repeatedly applies Algorithm 1, as in Theorem 1. The characteristics of the separation algorithm described above ensure that our approach produces the smallest decision diagram for separating the conflict at each iteration (Lemma 3) and ultimately terminates with the optimal solution (Theorem 1).

4 Network Flow-Based Formulation

We next reformulate the graph coloring problem based on independent sets, as a constrained network flow problem on a ~~given~~ decision diagram $D = (N, A)$. We let $\delta^+(u)$ and $\delta^-(u)$ denote the set of arcs leaving, respectively entering node $u \in N$. For each arc $a \in A$ we introduce a variable y_a that represents the 'flow' through a. We then define:

$$(F) = \min \sum_{a \in \delta^+(r)} y_a \tag{3}$$

$$\text{s.t.} \sum_{a=(u,v) | L(u)=j, \ell(a)=1} y_a = 1 \qquad \forall j \in V \tag{4}$$

$$\sum_{a \in \delta^-(u)} y_a - \sum_{a \in \delta^+(u)} y_a = 0 \qquad \forall u \in N \setminus \{r,t\} \tag{5}$$

$$y_a \in \{0, 1, \ldots, n\} \qquad \forall a \in A \tag{6}$$

The objective function (3) minimizes the total amount of flow. Constraints (4) define that in each layer exactly one 1-arc is selected. Constraints (5) ensure flow conservation. Constraints (6) ensure that the flow is integer.

Lemma 5 *A solution to model (F) corresponds to a (not necessarily unique) partition of vertex set V.*

Proof A solution to model (F) is an *r-t* flow, and can therefore be decomposed into a collection of paths and cycles. Since the decision diagram is a directed acyclic graph, no cycles exist. A partition can be found by applying the following algorithm that decomposes the flow into paths:

- Initialize the sets $V' = V$ and $P = \emptyset$ (P will represent a collection of subsets of V). Define a 'residual' flow vector y' that is initialized as $y'_a = y_a$ for all $a \in A$.
- While $\overline{V}$ is not empty: Select any $i \in V'$. By constraints (4), there is exactly one arc $a = (u, v)$ for which $y'_a = 1$. By constraints (5) there exists an *r-u* path and a *v-t* path for which $y'_a \geq 1$ for each arc a along the path, which together with arc (u, v) forms an *r-t* path p. Define the set $S = \{j \mid a = (u, v) \in p, L(u) = j, \ell(a) = 1\}$. Update $V' = V' \setminus S$, $P = P + S$, and $y'(a) := y'(a) - 1$ for all $a \in p$.[1]

By constraints (4) each vertex $i \in V$ must be part of exactly one path, and therefore P is a partition of V. The cardinality of P is equal to the number of paths in the decomposition, which equals the value of the objective function (3). □

Theorem 2 *If the decision diagram is exact, model (F) finds an optimal solution to the graph coloring problem.*

Proof By the proof of Lemma 5, we can decompose the solution to (F) into paths corresponding to a partition of V. Because each path in the exact decision diagram represents an independent set, the partition represents a coloring of the input graph. Since (F) minimizes the number of paths, the optimal objective function value is the chromatic number. □

[1] The operator $+$ indicates the addition of a set to a set (instead of the union).

By the definition of relaxed decision diagrams, we have the following corollary:

Corollary 1 *If the decision diagram is relaxed, the objective value of model (F) is a lower bound on the chromatic number.*

One may wonder whether model (F) can be solved in polynomial time, because of the structure imposed by the additional constraints (4). Such result would not imply a polynomial solution method for graph coloring, because the decision diagram may have exponential size. The answer, however, is negative:

Theorem 3 *Solving model (F) for an arbitrary decision diagram is NP-hard.*

Proof By reduction from minimum set partitioning. We are given a collection S of sets based on a universe of elements E, and need to find a subset of S of minimum cardinality such that each element in E belongs to exactly one subset. We define a decision diagram with $|E|+1$ node layers, such that layer i represents the i-th element from E following an arbitrary but fixed ordering of E. We then define an r-t path for each set $s \in S$ by introducing nodes and arcs between each layer i and $i+1$, with arc label 1 if the i-th element of E is in s and 0 otherwise. To do so, we apply the following algorithm, for each set $s \in S$:

- Initialize $u = r$ (start at the root node).
- For $i = 1, \ldots, |E|$: If there does not exist an arc $a = (u, v)$ with label $\ell(a) = \mathbb{I}[i \in s]$, add a new node v to layer $i+1$ and define arc a. Update $u = v$.

The resulting diagram has at most $|S|$ nodes in any layer, where each node has at most two outgoing arcs, and is therefore of size $O(|E||S|)$. An optimal solution to model (F) directly corresponds to solving the minimum set partitioning problem. □

As stated in Lemma 5, a solution to model (F) may give rise to multiple partitions of vertex set V. Indeed, paths in the solution are not explicitly given, but instead follow from the arc flow values. Moreover, paths can share nodes and 0-arcs. This gives rise to the following result:

Theorem 4 *If the decision diagram is relaxed, deciding whether a solution to model (F) corresponds to a feasible graph coloring solution is NP-complete, even if the objective value of (F) is the chromatic number of the input graph.*

Proof Let $G = (V, E)$ be the input graph, and consider the following transformation. Let (V', E') be a copy of (V, E), and define $G' = (V \cup V', E \cup E')$. By design, $\chi(G) = \chi(G')$. We define a relaxed decision diagram for G' where the first $|V|$ layers are associated with V, and the remaining layers are associated with V'. Up to, and including, layer $|V|$, we let the decision diagram be exact. All nodes in layer $|V|+1$ are merged into a single node u, with state $S(u) = V'$. Layers $|V|+1$ to $2|V|$ are each defined to contain a single node, and connect to the next layer with both the 1-arc and the 0-arc. We solve model (F) with this relaxed decision diagram as input. Because the layers associated with V are exact, by Theorem 2 the optimal objective value of (F) will be at least $\chi(G)$. Because the layers associated with V' do not restrict the solution further, we know that the optimal objective value of (F) is equal to $\chi(G)$ and thus to $\chi(G')$. Furthermore, the optimal solution corresponds to a collection of

Algorithm 2: Edge conflict detection via flow decomposition.

input: decision diagram $D = (N, A)$ ($\mathsf{D}[i][j]$ represents the jth node in layer i), solution to model (F) (represented as $\mathsf{D}[i][j].\mathsf{oneArcFlow}$ and $\mathsf{D}[i][j].\mathsf{zeroArcFlow}$) with optimal objective value B, input graph $G = (V, E)$ and list of neighbors N_i for each $i \in V$
output: certificate of edge conflict with associated node and arc-label specified paths (an empty conflict represent feasibility)
define: node index vector P, arc label vector L, vector of selected vertices S
while $B \geq 1$ **do**
    empty P, L, S
    P.add(1) // start at root node
    for i = *1, . . . , n* **do**
        $u \leftarrow P$.last
        if $\mathsf{D}[i][u].\mathsf{oneArcFlow} = 1$ **then** // select 1-arc if it carries flow
            for $j \in S$ **do**
                if $(j, i) \in E$ **then**
                    return conflict (j, i) with node path $P_j, P_{j+1}, \ldots, P_{i-1}$ and arc-label path $L_j, L_{j+1}, \ldots, L_{i-1}$
            P.add($\mathsf{D}[i][u].\mathsf{oneArc}$)
            L.add(1)
            S.add(i)
        else
            P.add($\mathsf{D}[i][u].\mathsf{zeroArc}$)
            L.add(0)
    for $i = 1, \ldots, n$ **do**
        if $L_i = 0$ **then** $\mathsf{D}[i][P_i].\mathsf{zeroArcFlow} \leftarrow \mathsf{D}[i][P_i].\mathsf{zeroArcFlow} - 1$
        else $\mathsf{D}[i][P_i].\mathsf{oneArcFlow} \leftarrow \mathsf{D}[i][P_i].\mathsf{oneArcFlow} - 1$
    $B \leftarrow B - 1$
return 0

conflict-free paths for the first $|V|$ layers, all ending in u. For layers $|V|+1$ to $2|V|$, the solution will assign a value of 1 to each 1-arc, and a value of $\chi(G') - 1$ to each 0-arc (by flow conservation starting at node u).

To decide whether the solution of (F) corresponds a feasible coloring to G' (i.e., on (V, E) and (V', E') simultaneously), we need to decompose it into $\chi(G')$ conflict-free paths. Starting from the root, the first $|V|$ arcs along these paths are given by the solution to (F), since that part of the diagram is exact. Each of these paths can be arbitrarily extended to contain a subset of 1-arcs from layers $|V|+1$ to $2|V|$. Completing a feasible coloring to G' therefore amounts to decomposing the solution into $\chi(G')$ conflict-free paths in layers $|V|+1$ up to $2|V|$. This is equivalent to determining whether the original input graph is $\chi(G)$-colorable, which is NP-complete [15]. □

5 Iterative Refinement Procedure

In this section we describe the iterative refinement procedure that employs Algorithm 1 to separate edge conflicts from paths in the relaxed decision diagram. While the worst-case complexity results in the previous section are negative (Theorems 3

Algorithm 3: Iterative refinement by conflict detection and separation.

```
1  input: input graph G = (V,E), and list of neighbors N_i for each i ∈ V
2  output: chromatic number of G
3  foundSol ← false
4  initialize width-1 decision diagram D
5  while foundSol = false do
6      solve model (F) with decision diagram D
7      lowerBound ← obj(F)
8      apply Algorithm 2 to determine conflict (j,k) with node/label path vectors P,L
9      if no conflict is detected then foundSol ← true
10     else separate conflict (j,k) along path P,L in D using Algorithm 1
11 return lowerBound
```

and 4), we can, however, efficiently identify a conflict in a given solution, as presented in Algorithm 2.

Theorem 5 *For a given solution to (F) and decision diagram D, Algorithm 2 either determines feasibility or identifies an edge conflict and associated path, in $O(n^3)$ time).*

Proof Algorithm 2 implements a path decomposition of the flow. By the specification of model (F), each path in the decomposition carries a flow of value 1. While the remaining flow is at least 1 (line 4), the algorithm identifies a path from r to t such that the flow along each arc is at least 1. The algorithm maintains a vector P of node indices and vector L of arc labels representing the path. It also maintains a vector S of vertices i for which the associated label in the path is 1. Before vertex i is added to S, we inspect whether there exists an edge $(j,i) \in E$ with $j \in S$. If so, we terminate and return the conflict (j,i) as well as the node and label specified paths P and L (line 12). Otherwise, we update P and L based on whether a 1-arc (lines 15-16) or 0-arc (lines 20-21) is selected. If no conflict has been identified, the flow along the path is decreased by 1 (lines 24-27), as is the remaining flow value (line 28). If during the process no conflict is detected, the path decomposition represents a feasible coloring, and the algorithm returns 0 (line 30).

Finding one r-t path takes n steps (line 7). The edge inspection (lines 10-11) takes $O(n)$ time per event, which makes the total time for identifying a single path $O(n^2)$. Since there are at most n paths, i.e., $B \leq n$, the total time is $O(n^3)$. □

We can now describe our overall iterative refinement procedure, presented as Algorithm 3. The algorithm repeatedly solves the flow model (line 6), the objective of which provides a lower bound (line 7). If the solution contains a conflict (line 8), the decision diagram is refined accordingly (line 10), and we repeat the process. Otherwise, we return the optimal solution (line 9). The process is illustrated in the following example.

Example 3 Figure 2 depicts the iterative refinement procedure when applied to the input graph given in Figure 1(a). We start with the trivial relaxation in Fig. 2(a), which encodes all possible subsets of V, and find the all-ones solution as a single path. Thus, the lower bound is 1. The first conflict we identify is edge (1,3) which

is subsequently separated. An optimal flow for the diagram in Fig. 2(b) contains two paths which yields lower bound 2, and we can identify conflict (2,4) to be separated. This continues until we find the diagram in Fig. 2(d) and identify flow paths without conflicts. Observe that the diagram in Fig. 2(d) is not exact, yet it yields an optimal solution. Furthermore, observe that the flow paths are not unique. If we had identified the flow paths (1,0,0,1) and (0,1,1,0) as optimal solution for the diagram in Fig. 2(b), the algorithm would have terminated after one iteration.

Theorem 6 *Given graph G, Algorithm 3 computes the chromatic number of G.*

Proof The validity of the lower bound is provided by Corollary 1. Feasibility of the solution is determined by Algorithm 2 and guaranteed by Theorem 5. In case of a conflict, Algorithm 2 returns the first edge (j,i) it encounters along the path P, L (line 10-11). Therefore, no edge conflicts (j',i') with $j \leq j' < i' < i$ exist along the path, and (j,i), P, and L satisfy the conditions for Algorithm 1. Since at each iteration of Algorithm 3 the decision diagram is refined in case of a conflict, termination is guaranteed by Theorem 1. □

Example 3 shows that iterative refinement may yield a relaxed decision diagram that is smaller than the exact decision diagram, to prove optimality. This is formalized in the following key result:

Theorem 7 *For a given graph G, the iterative refinement procedure can find the chromatic number of G with a relaxed decision diagram that is exponentially smaller (in the size of G) than the exact diagram that is defined on the same variable ordering.*

Proof We prove that there exists a graph coloring instance class, and associated vertex ordering, such that the exact decision diagram is of exponential size while a polynomial-size relaxed decision diagram exists that proves optimality. Consider a graph $G = (V,E)$ with vertex labels $V = \{1,\ldots,n\}$ and edge set $\{(i,i+1) \mid i \in \{1,\ldots,n-1\}\}$, i.e., G is a path from vertex 1 to n. We define the following fixed variable ordering to compile the decision diagrams:

- For layers $i = 1,\ldots,\lfloor n/3 \rfloor$, we associate the variable corresponding to vertex $1 + 3(i-1)$.
- For layers $i = \lfloor n/3 \rfloor + 1,\ldots,n$, we associate the variable corresponding to an arbitrary vertex that has not been assigned to layers $1,\ldots,i-1$.

We first the consider the exact decision diagram that is compiled according to this ordering.

- Observation 1: Up to layer $\lfloor n/3 \rfloor$, none of the associated vertices is adjacent, and appears in each of the states. Therefore, each state up to layer $\lfloor n/3 \rfloor$ has two outgoing arcs, the 0-arc and the 1-arc.
- Observation 2: Consider the transition of a node u with state $S(u)$ into state $S(u')$, following the transition rules (2). Up to layer $\lfloor n/3 \rfloor$, the transition along each 1-arc eliminates the associated vertex as well as its neighbors, while each 0-arc eliminates the associated vertex. Furthermore, up to layer $\lfloor n/3 \rfloor$, the associated vertices have no common neighbors. Therefore, when expanding layer i into layer $i+1$, each of the resulting states is distinct, for $i < \lfloor n/3 \rfloor$.

These two observations imply that the exact decision diagram requires at least $O(2^{\lfloor n/3 \rfloor})$ states (the size of layer $\lfloor n/3 \rfloor$).

Next, consider the relaxed decision diagram that is constructed by applying the iterative refinement procedure, starting with a width-1 diagram. The initial solution is the path consisting of all 1-arcs. After refining the first conflict the lower bound becomes 2, which is the chromatic number of G. For this diagram, there exists an optimal solution to (F) and associated path decomposition consisting of two conflict-free paths (one with 1-arcs associated with odd vertices, and the other with 1-arcs associated with even vertices). The iterative refinement procedure can therefore terminate with a decision diagram of $O(n)$ size. □

6 Primal Heuristic

In this section we describe how the relaxed decision diagram, together with the optimal flow solution, can be used to develop a primal heuristic. The heuristic is presented in Algorithm 4. It follows a similar structure as the flow path decomposition of Algorithm 2. Namely, until all vertices have been assigned to a color class (line 6), the algorithm identifies a path in the decision diagram. The heuristic greedily follows the 1-arcs that have at least as much flow value as the alternative 0-arc (line 11). Otherwise, the 0-arc is selected by default (line 20). If the 1-arc is selected, the corresponding vertex is added to the path's color class (line 16), and the vertex and its neighbors are added to the set *NS* of neighbors. Once the path is completed, the color class is added if it exists (line 25) and the flow values are updated according the minimum flow value 'val' along the path (lines 27-29). If the color class is not used, and not all vertices have been assigned to a color class (line 30), the remaining vertices are added to the first non-conflicting color class (lines 32-35) or otherwise to a newly created color class (line 37).

The main differences with Algorithm 2 are that arcs can be selected in the path even if their flow value is zero, and that 1-arcs in the path need not necessarily be added to the color class.

Theorem 8 *Given input graph G, decision diagram D, and a solution to model (F), Algorithm 4 finds a coloring of G in $O(n^3)$ time.*

Proof In each iteration we either find a path with at least one unselected vertex, or determine that no such path exists. In the first case, we decrease the number of available vertices by at least one. In the second case, we terminate the path decomposition and continue adding the vertices to available color classes. Since each color class only contains non-adjacent vertices, the algorithm finds a feasible coloring of G, in at most n iterations.

Computing one path takes n steps (line 9), in which the neighbor membership test (line 15) takes $O(n)$ time. Overall, this bounds the complexity to $O(n^2)$ per path. The second part of the algorithm (adding the unassigned vertices) can be amortized to take $O(n)$ time for each unassigned vertex $i \notin S$. Namely, all classes $c \in C$ are disjoint and therefore the maximum number of edge checks (line 33) is bounded by n. The overall complexity of the algorithm is therefore $O(n^3) + O(n^2) = O(n^3)$. □

Algorithm 4: Flow-based primal heuristic.

```
1  input: decision diagram D = (N,A) (D[i][j] represents the jth node in layer i), solution to model
     (F) (represented as D[i][j].oneArcFlow and D[i][j].zeroArcFlow), and input graph G = (V,E)
2  output: coloring of G
3  define: node index vector P, arc label vector L, vector of selected vertices S, set of neighbors NS,
     collection of color classes C
4  begin
5      colorUsed ← true
6      while (|S| < n) ∧ ( colorUsed=true) do
7          class ← ∅; colorUsed ← false; empty P,L,NS ; val ← n
8          P.add(1)
9          for i = 1, ..., n do
10             u ← P.last
11             if D[i][u].oneArcFlow ≥ D[i][u].zeroArcFlow then          // prefer selecting 1-arc
12                 val ← min(val, D[i][u].oneArcFlow)
13                 P.add(D[i][u].oneArc)
14                 L.add(1)
15                 if (i ∉ NS) then
16                     class.add(i)
17                     S ← S ∪ {i}
18                     NS ← NS ∪ {i} ∪ {j | (i, j) ∈ E}
19                     colorUsed ← true
20             else
21                 val ← min(val, D[i][u].zeroArcFlow)
22                 P.add(D[i][u].zeroArc)
23                 L.add(0)
24         if colorUsed then
25             C.add(class)
26             if val > 0 then
27                 for i = 1,...,n do
28                     if L_i = 0 then D[i][P_i].zeroArcFlow ← D[i][P_i].zeroArcFlow−val
29                     else D[i][P_i].oneArcFlow ← D[i][P_i].oneArcFlow−val
30     if |S| < n then
31         for i ∉ S do
32             for c ∈ C do
33                 if ∄j ∈ c | (i, j) ∈ E then
34                     c.add(i)                                  // add i to existing color class
35                     S ← S ∪ {i}
36                     break
37             if i ∉ S then
38                 C.add({i})                                    // add i as new color class
39                 S ← S ∪ {i}
40     return C
```

Algorithm 5: Iterative refinement with lower and upper bounds.

```
1  input: input graph G = (V, E)
2  output: chromatic number of G
3  begin
4      initialize width-1 decision diagram D
5      lowerBound ← 0
6      upperBound ← n
7      while lowerBound < upperBound do
8          solve model (F) with decision diagram D
9          lowerBound ← obj(F)
10         apply Algorithm 4 to determine coloring C
11         upperBound ← min(upperBound, |C|)
12         apply Algorithm 2 to determine conflict (j,k) with node/label path vectors P,L
13         if no conflict is detected then upperBound ← obj(F)
14         else separate conflict (j,k) along path P,L in D using Algorithm 1
15     return lowerBound
```

The addition of the primal heuristic can help speed up the iterative refinement procedure. In addition to terminating the refinement when no more edge conflict is detected (Algorithm 3), we can prove optimality when the lower bound equals the best found solution. This is described in Algorithm 5, which extends Algorithm 3 with an upper bound based on the primal heuristic, and the associated stopping criterion.

7 Implementation Details

We next describe the main features of the implementation that influence its computational efficiency.

7.1 Variable Ordering

The single most important parameter that influences the performance of the algorithm is the variable ordering of the decision diagram. This is a well-known feature of decision diagrams in general, and has also been studied for independent set problems [6, 7]. The orderings developed in those works are applied to a top-down compilation (layer by layer) of the diagram, and can therefore dynamically incorporate the state information at each layer to select the next variable. In our case, we instead adopt an iterative refinement strategy, which prevents the application of dynamic ordering heuristics. Instead, our variable ordering is computed in a pre-processing phase and applied in a fixed manner. We consider the following variable ordering heuristics:

- Lexicographic: Order the variables by the specification of the input graph, i.e., $\{1,2,\ldots,n\}$.
- Dsatur: Order the variables using the Dsatur heuristic.
- Max-Connected-Degree: Among all unselected vertices, select the one that is connected to the most vertices that have been selected so far. In case of ties, select a vertex with the highest degree.

As we will see in Section 8, none of these heuristics strictly dominates the others. Although the lexicographic ordering is the weakest, it was for example effective on the highly structured `queen` instances from the DIMACS benchmark set. The Dsatur and Max-Connected-Degree heuristics have similar relative performance, with the Max-Connected-Degree heuristics performing better on average.

7.2 Network Flow Model

It is common in column generation to replace the equality constraints of the set partitioning model (1) with inequality constraints, yielding a set covering formulation. Such reformulation often has computational benefits. We can do the same for our network flow model, i.e., the equality constraints (4) can be replaced by the following inequalities:

$$\sum_{a=(u,v)|L(u)=j,\ell(a)=1} y_a \geq 1 \quad \forall j \in V. \tag{7}$$

This is an equivalent formulation, since all solutions to model (F) with constraint (4) are also solutions with respect to constraints (7). Furthermore, although a vertex may now be associated with multiple paths, we can arbitrarily select one of these paths to assign the vertex to its color class. All results presented in this paper can be easily adapted to handle the inequality constraints. In a preliminary experiment, we compared the two formulations, and found that the inequality constraints indeed perform better on average. Therefore, the results in Section 8 all utilize the inequality constraints (7).

The constrained network flow problem (F) is NP-hard to solve in general (see Lemma 3). In practice, however, modern mixed-integer programming solvers can optimally solve instances with tens or hundreds of thousands variables in reasonable time. Nonetheless, this is still the computational bottleneck for each iteration of our iterative refinement algorithm. To help speed up this process, we therefore explored the use of its linear programming (LP) relaxation, which is obtained by replacing the integrality constraints (6) by $0 \leq y_a \leq n$ for all $a \in A$. Naturally, the LP relaxation of (F) provides a lower bound on the chromatic number. In addition, as the continuous solution is a flow, we can also apply the path decomposition approach to identify conflicts. In our implementation, we apply a path decomposition algorithm for continuous flows that is similar to the one for the primal heuristic (Algorithm 4): the path selects a 1-arc if it has at least as much flow as the 0-arc, and otherwise selects the 0-arc by default. While the LP bound may be weaker than the integer programming (IP) bound, it is faster to compute and may therefore speed up the overall process.

We implemented the LP relaxation as a separate phase of Algorithm 5. That is, in line 8, we start by solving the LP relaxation during each iteration. This is continued until a conflict-free path decomposition is found, yielding the LP optimum. After that, we continue by applying the IP model, as before. We evaluated the potential benefit of LP-based iterative refinement on all 137 DIMACS instances. We ran Algorithm 5 with and without the LP relaxation, each with maximum of 300s per instance. This yielded the following results:

- For 2 instances, IP alone found a better lower bound than LP followed by IP;
- For 31 instances, the LP followed by IP yielded a better lower bound than IP alone;
- For 104 instances, the lower bounds from both methods were equal. For these instances, however, adding the LP relaxation reduced the time to find the best bound by about 50%.

Based on these results, we conclude that the LP relaxation is advantageous, and it is applied to obtain the results in Section 8. In the implementation, the lower bound is calculated as $\lceil z^* - 10^{-5} \rceil$, where z^* is the optimal objective value reported by the solver.

7.3 Separate Multiple Conflicts Per Iterations

Instead of separating a single conflict in each iteration, it is possible to identify and separate multiple conflicts; one for each path in the decomposition (if the path contains a conflict). This requires a slight adaptation of Algorithm 2: instead of terminating once a conflict is found, the algorithm now records the conflict, completes the path to the terminal, and continues the path decomposition. When the remaining flow is zero, and the path decomposition is complete, the algorithm terminates and returns the collection of conflicts. This also has implications for the conflict separation procedure (Algorithm 1), which is now applied in sequence for each conflict.

We point out that separating each conflict will change the structure of the decision diagram, as new nodes are introduced and arcs are redirected. We must therefore ensure that the paths, in particular the node indices, that represent these conflicts still exist when multiple conflicts are separated. Recall that the vertex partition defined by the path decomposition is disjoint. Furthermore, the separation of conflicts only adds nodes at the end of each layer; no nodes are removed from the layer or inserted in between other nodes. This guarantees that the node indices of each path remain accurate.

We evaluated the impact of separating multiple cuts per iteration, on all 137 DIMACS instances. We ran Algorithm 5 with single and multiple conflict separation, in each case for a maximum of 300s per instance. The results are as follows:

- For 3 instances, single conflict separation found a better lower bound than multiple conflict separation;
- For 43 instances, multiple conflict separation found a better lower bound than single conflict separation;
- For 91 instances, the lower bounds from both methods were equal. For these instances, however, multiple conflict separation was about 2.8 times faster than single conflict separation to find the best bound.

Based on these results, we conclude that separating multiple conflicts per iteration is quite effective, and it is applied to obtain the results in Section 8.

7.4 Improved Relaxation by Redirecting Arcs

In previous work, it has been observed that relaxed decision diagrams can be strengthened by redirecting arcs appropriately [4,31]. We apply a similar approach here, tailored to the state representation for independent set problems. Recall that each state in the diagram is a subset of eligible vertices. The transition rules (2) prescribe how for a node u its state $S(u)$ is updated according to a 1-arc or 0-arc transition, resulting in a new state $S(u')$. For exact decision diagrams, the transition is required to end in state u' on the next layer. For relaxed decision diagrams, u' may not exist, in which case the transition is directed to another state. To ensure a proper relaxation, we must select a state (in fact, any state) v in the next layer, such that $S(u') \subseteq S(v)$. A natural heuristic choice is to select the 'most similar' node, i.e., a node v for which $S(u') \cap S(v)$ is maximized.

Consider now the conflict separation algorithm (Algorithm 1). As we follow the arcs along the path, we split off new nodes and copy their associated 1-arcs and 0-arcs by directing them to the original nodes along the given path (lines 10, 12). However, we could alternatively redirect the arcs to other nodes in the layer, which may result in a better relaxation. For this purpose, we implemented the 'most similar' heuristic described above to redirect arcs in Algorithm 1. An initial evaluation of its performance indicated that this redirection strategy can be beneficial in some cases, but can be detrimental in other cases. For the experiments in Section 8, we therefore ran our procedure with and without redirecting arcs.

7.5 Initialization

We added two other features to streamline the solving process. First, before entering the iterative refinement procedure, we apply the Dsatur heuristic to initialize the upper bound (replacing n in line 6 of Algorithm 3).

Second, before running the LP and IP-based iterative refinement, we run a refinement procedure based on the longest path (with respect to 1-arcs) in the decision diagram. The intuition is that in the initial iterations, conflicts are likely to be found on paths with many 1-arcs. It is computationally much cheaper to identify such paths using a longest path algorithm (which runs in linear time in the size of the decision diagram), than using the LP or IP model. Once a longest path is found, we return the first conflict that is detected and separate it, similar to the paths found by the flow decomposition. If the longest path does not contain a conflict, or if a maximum number of iterations is reached, we terminate this process. We apply longest path-based conflict separation to obtain the results in Section 8, for a maximum of 100 iterations.

8 Experimental Results

We implemented our method in C++, and performed an experimental evaluation on the 137 DIMACS graph coloring benchmark instances [20]. We use CPLEX 12.9 as integer and linear programming solver, using a single thread and the Barrier Method

as root LP algorithm. All reported experiments are run on a machine with an Intel Xeon E5345@2.33GHz CPU running Ubuntu 18.04.

As mentioned in Section 7, in each run the iterative refinement procedure deploys the LP relaxation prior to the IP model, separates multiple conflicts per iteration, and initializes the procedure by running the Dsatur heuristic to obtain an upper bound, as well as a longest path-based conflict separation for at most 100 iterations.

8.1 Size of Exact and Relaxed Decision Diagrams

The aim of our first experiment is to compare the performance of the iterative refinement procedure and the exact compilation. In particular, we wish to understand whether small relaxed decision diagrams can still be as effective as exact decision diagrams for proving optimality. For this experiment, we apply the Max-Connected-Degree variable ordering, and we do not apply the arc redirection strategy. We consider all instances that are solved to optimality by either method within a time limit of 1 hour, and a maximum decision diagram size of 1,000,000 nodes. There were 52 instances solved to optimality under these settings: 46 were solved within one minute, and 36 within one second. Iterative refinement was able to solve 50 instances, whereas the exact decision diagram solved 39 instances. Table 1 reports the performance of both methods on these 52 instances in terms of running time and decision diagram size. Two instances that are solved by the exact decision diagram could not be solved within the given time limit by the relaxed decision diagram. Conversely, 13 instances were solved by the relaxed decision diagram but not the exact decision diagram.

Figure 3 presents a scatter plot comparing the size of the relaxed and the exact decision diagrams for the 52 instances that were solved optimally. A first observation is that the exact decision diagram can be remarkably small for some instances. Perhaps even more remarkable is that the relaxed diagram can sometimes be orders of magnitude smaller than the exact diagram for proving optimality, demonstrating the value of Theorem 7 in practice. As an example, for the Max-Connected-Degree variable ordering, instance `DSJR500.1` ($n = 500$, $m = 3,555$) requires an exact decision diagram of at least 1M nodes, whereas the relaxed decision diagram only needs 627 nodes to prove optimality.

8.2 Detailed Comparison of Algorithmic Settings

We next present the performance of the algorithm under different parameter settings. In each case, the algorithm is executed with a time limit of 1 hour and a maximum of 1,000,000 nodes. We evaluate each of the variable orderings (Lexicographic, Dsatur, and Max-Connected-Degree) for the iterative refinement procedure, without the option of redirecting arcs. For the most effective ordering (Max-Connected-Degree) we also evaluate the performance with redirecting arcs. In addition, we compile the exact decision diagram for each of the variable orderings. Together with the initial upper bound provided by the Dsatur heuristic, this gives a total of eight algorithmic settings that can be responsible for the best lower and upper bound that we report for each DIMACS instance:

			Relaxed DD				Exact DD				
Instance	n	m	LB	UB	Size	Time	LB	UB	Size	Time	R/E
1-FullIns_3	30	100	4	4	250	0.06	4	4	748	0.09	0.33
2-FullIns_3	52	201	5	5	1,233	0.83	5	5	12,867	2.73	0.10
3-FullIns_3	80	346	6	6	6,058	13.15	6	6	435,083	581.71	0.01
david	87	406	11	11	247	0.01	11	11	37,030	5.07	0.01
DSJC125.9	125	6,961	44	44	9,435	25.11	44	44	9,869	1.19	0.96
DSJR500.1c	500	121,275	85	85	138,048	1,358.57	85	85	145,777	5.62	0.95
fpsol2.i.1	496	11,654	65	65	6,947	10.33	65	65	8,296	0.37	0.84
fpsol2.i.2	451	8,691	30	30	958	0.21	30	30	10,168	0.56	0.09
fpsol2.i.3	425	8,688	30	30	971	0.25	30	30	10,258	0.76	0.09
huck	74	301	11	11	787	0.23	11	11	1,078	0.07	0.73
inithx.i.1	864	18,707	54	54	3,994	3.17	54	54	15,805	0.68	0.25
inithx.i.2	645	13,979	31	31	22,340	121.09	31	31	24,589	4.01	0.91
inithx.i.3	621	13,969	31	31	22,503	163.70	31	31	24,551	2.37	0.92
jean	80	254	10	10	292	0.01	10	10	5,252	0.73	0.06
miles1000	128	3,216	42	42	4,105	1.87	42	42	8,032	0.33	0.51
miles1500	128	5,198	73	73	2,698	1.10	73	73	4,008	0.16	0.67
miles250	128	387	8	8	295	0.01	8	8	2,813	0.46	0.10
miles500	128	1,170	20	20	362	0.04	20	20	15,273	1.71	0.02
miles750	128	2,113	31	31	711	0.12	31	31	13,154	0.96	0.05
mulsol.i.1	197	3,925	49	49	1,108	0.40	49	49	2,488	0.07	0.45
mulsol.i.2	188	3,885	31	31	794	0.19	31	31	2,612	0.09	0.30
mulsol.i.3	184	3,916	31	31	790	0.19	31	31	2,622	0.09	0.30
mulsol.i.4	185	3,946	31	31	807	0.19	31	31	2,637	0.09	0.31
mulsol.i.5	186	3,973	31	31	808	0.19	31	31	2,650	0.09	0.30
myciel3	11	20	4	4	60	0.04	4	4	63	0.02	0.95
myciel4	23	71	5	5	453	7.03	5	5	460	0.85	0.98
queen5_5	25	160	5	5	195	0.01	5	5	561	0.03	0.35
queen6_6	36	290	7	7	1,811	2.35	7	7	2,687	0.36	0.67
queen7_7	49	476	7	7	3,303	2.77	7	7	13,839	0.88	0.24
queen8_8	64	728	9	9	28,742	310.31	9	9	81,575	107.59	0.35
r125.1	125	209	5	5	340	0.02	5	5	921	0.05	0.37
r125.1c	125	7,501	46	46	3,570	2.08	46	46	4,008	0.13	0.89
r125.5	125	3,838	36	36	20,065	212.90	36	36	23,243	2.03	0.86
r250.1c	250	30,227	64	64	16,426	31.19	64	64	20,323	0.61	0.81
zeroin.i.1	211	4,100	49	49	1,025	0.32	49	49	2,770	0.12	0.37
zeroin.i.2	211	3,541	30	30	775	0.17	30	30	3,471	0.17	0.22
zeroin.i.3	206	3,540	30	30	770	0.20	30	30	3,458	0.19	0.22
4-FullIns_3	114	541	7	7	17,295	99.58	0	-	856,849	timeout	0.02
5-FullIns_3	154	792	8	8	61,201	1,233.00	0	-	$>$ 1M	-	$\leq$ 0.06
anna	138	493	11	11	357	0.01	0	-	$>$ 1M	-	$\leq$ 0.00
DSJR500.1	500	3,555	12	12	627	0.03	0	-	$>$ 1M	-	$\leq$ 0.00
games120	120	638	9	9	36,092	92.74	0	-	$>$ 1M	-	$\leq$ 0.04
le450_25a	450	8,260	25	25	944	0.17	0	-	$>$ 1M	-	$\leq$ 0.00
le450_25b	450	8,263	25	25	828	0.14	0	-	$>$ 1M	-	$\leq$ 0.00
le450_5d	450	9,757	5	5	701	0.03	0	-	$>$ 1M	-	$\leq$ 0.00
r1000.1	1,000	14,378	20	20	1,235	0.20	0	-	$>$ 1M	-	$\leq$ 0.00
r250.1	250	867	8	8	11,974	7.32	0	-	$>$ 1M	-	$\leq$ 0.01
school1	385	19,095	14	14	2,366	1.67	0	-	$>$ 1M	-	$\leq$ 0.00
school1_nsh	352	14,612	14	14	7,406	15.59	0	-	$>$ 1M	-	$\leq$ 0.01
wap05a	905	43,081	50	50	19,431	15.80	0	-	$>$ 1M	-	$\leq$ 0.02
2-Insertions_3	37	72	3	4	2,656	timeout	4	4	2,964	930.37	0.90
r250.5	250	14,849	65	67	123,585	timeout	65	65	232,727	134.95	0.53

Table 1 Comparing the performance of relaxed and exact decision diagrams on instances that were optimally solved by either method. For each instance we list the number of nodes (n) and edges (m). We report the lower bound (LB), upper bound (UB), solving time (in seconds), and the size of the decision diagram. The last column (R/E) represents the ratio of the relaxed and exact diagram sizes. The time limit was set to 3,600s, and the maximum size was set to 1 million nodes.

A: Lexicographic ordering;
B: Dsatur ordering;
C: Max-Connected-Degree ordering;

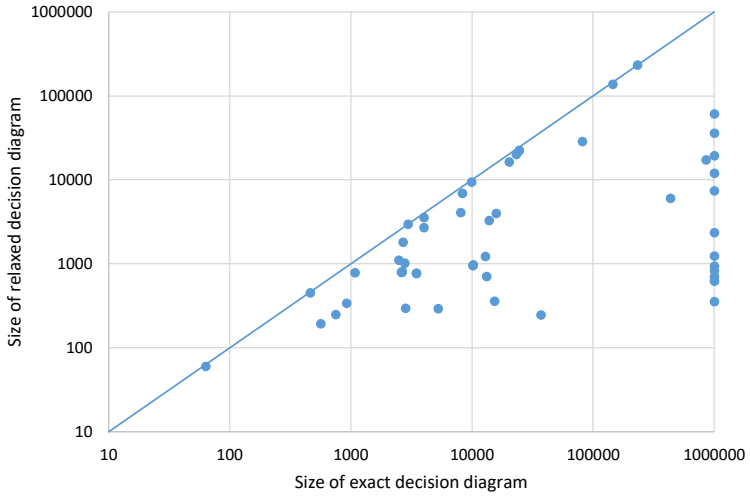


Fig. 3 Comparing the size of exact and relaxed decision diagrams for 52 DIMACS instances that were solved to optimality by either method.

D: Max-Connected-Degree ordering with redirecting arcs;
E: Lexicographic ordering with exact compilation;
F: Dsatur ordering with exact compilation;
G: Max-Connected-Degree ordering with exact compilation;
H: Dsatur heuristic (provides upper bound only).

In Table 2 we provide the best lower and upper bound found by any method for each of the DIMACS instances, and indicate for each bound which method(s) obtained that bound. However, if the best upper bound was found by the initial Dsatur heuristic (setting 'H'), we only report that.

Out of the 137 instances, 54 instances were solved to optimality in total. The performance of the algorithmic settings are summarized in the following table, which reports the number of instances for which that setting obtained the optimal solution (#Optimal), the best lower bound (#Best LB), and the best upper bound (#Best UB):

Setting	#Optimal	#Best LB	#Best UB
A	22	59	10
B	46	113	18
C	50	117	19
D	49	113	20
E	26	30	10
F	40	46	12
G	39	46	9
H	-	-	109

						Decision Diagram				
Instance	n	m	d	$\underline{\chi}$	$\overline{\chi}$	LB	Setting	UB	Setting	
1-FullIns_3	30	100	0.23	4	4	**4**	ABCDEFG	**4**	H	*
1-FullIns_4	93	593	0.14	5	5	4	ABCDG	**5**	H	
1-FullIns_5	282	3247	0.08	6	6	4	ABCD	**6**	H	
1-Insertions_4	67	232	0.10	5	5	3	ABCDFG	**5**	H	
1-Insertions_5	202	1227	0.06	6	6	3	ABCD	**6**	H	
1-Insertions_6	607	6337	0.03	4	7	3	ABCD	**7**	H	
2-FullIns_3	52	201	0.15	5	5	**5**	ABCDEFG	**5**	H	*
2-FullIns_4	212	1621	0.07	6	6	5	C	**6**	H	
2-FullIns_5	852	12201	0.03	7	7	4	ABCD	**7**	H	
2-Insertions_3	37	72	0.11	4	4	**4**	EFG	**4**	H	*
2-Insertions_4	149	541	0.05	5	5	3	ABCD	**5**	H	
2-Insertions_5	597	3936	0.02	6	6	3	ABCD	**6**	H	
3-FullIns_3	80	346	0.11	6	6	**6**	CDEFG	**6**	H	*
3-FullIns_4	405	3524	0.04	7	7	5	ABCD	**7**	H	
3-FullIns_5	2030	33751	0.02	8	8	5	ABCD	**8**	H	
3-Insertions_3	56	110	0.07	4	4	3	ABCDEFG	**4**	H	
3-Insertions_4	281	1046	0.03	5	5	3	ABCD	**5**	H	
3-Insertions_5	1406	9695	0.01	4	6	3	ABCD	**6**	H	
4-FullIns_3	114	541	0.08	7	7	**7**	C	**7**	H	*
4-FullIns_4	690	6650	0.03	8	8	7	C	**8**	H	
4-FullIns_5	4146	77305	0.01	9	9	7	C	**9**	H	
4-Insertions_3	79	156	0.05	4	4	3	ABCDG	**4**	H	
4-Insertions_4	475	1795	0.02	5	5	3	ABCD	**5**	H	
5-FullIns_3	154	792	0.07	8	8	**8**	C	**8**	H	*
5-FullIns_4	1085	11395	0.02	9	9	8	C	**9**	H	
abb313GPIA	1557	53356	0.04	9	9	8	ABCD	10	H	
anna	138	493	0.05	11	11	**11**	BCD	**11**	H	*
ash331GPIA	662	4181	0.02	4	4	**4**	ABCD	5	H	
ash608GPIA	1216	7844	0.01	4	4	**4**	B	5	H	
ash958GPIA	1916	12506	0.01	4	4	**4**	B	5	H	
C2000.5	2000	999836	0.50	99	145	20	CD	208	H	
C2000.9	2000	1799532	0.90	98	400	85	BD	563	H	
C4000.5	4000	4000268	0.50	107	259	20	CD	381	H	
david	87	406	0.11	11	11	**11**	BCDFG	**11**	H	*
DSJC1000.1	1000	49629	0.10	10	20	6	BCD	26	H	
DSJC1000.5	1000	249826	0.50	73	82	19	CD	119	H	
DSJC1000.9	1000	449449	0.90	216	222	86	D	304	H	
DSJC125.1	125	736	0.09	5	5	**5**	BCD	6	H	
DSJC125.5	125	3891	0.50	17	17	16	EFG	20	EFG	
DSJC125.9	125	6961	0.90	44	44	**44**	ABCDEFG	**44**	ABCDEFG	*
DSJC250.1	250	3218	0.10	6	8	5	BCD	10	H	
DSJC250.5	250	15668	0.50	26	28	16	BCD	36	H	
DSJC250.9	250	27897	0.90	72	72	**72**	ABF	**72**	AF	*
DSJC500.1	500	12458	0.10	9	12	5	BCD	16	H	
DSJC500.5	500	62624	0.50	43	47	18	CD	68	H	
DSJC500.9	500	112437	0.90	123	126	**123**	EFG	136	EF	
DSJR500.1	500	3555	0.03	12	12	**12**	BCD	**12**	H	*
DSJR500.1c	500	121275	0.97	85	85	**85**	BCDEFG	**85**	BCDEFG	*
DSJR500.5	500	58862	0.47	122	122	115	D	126	BC	
flat1000_50_0	1000	245000	0.49	50	50	19	CD	115	H	
flat1000_60_0	1000	245830	0.49	60	60	19	C	114	H	
flat1000_76_0	1000	246708	0.49	72	81	19	C	116	H	

Table 2 Lower and upper bounds obtained by the various settings of the decision diagram procedure (A, B, …, H) for all DIMACS instances. For each instance we report the number of nodes and edges, the density, and the best known lower bound ($\underline{\chi}$) and upper bound $\overline{\chi}$). Bounds in bold meet the best known bounds. Instances solved to optimality are indicated with an asterisk.

Instance	n	m	d	$\underline{\chi}$	$\overline{\chi}$	Decision Diagram LB	Setting	UB	Setting	
flat300_20_0	300	21375	0.48	20	20	16	CD	42	H	
flat300_26_0	300	21633	0.48	26	26	16	CD	43	H	
flat300_28_0	300	21695	0.48	28	28	16	CD	44	H	
fpsol2.i.1	496	11654	0.09	65	65	**65**	BCDFG	**65**	H	*
fpsol2.i.2	451	8691	0.09	30	30	**30**	BCDFG	**30**	H	*
fpsol2.i.3	425	8688	0.10	30	30	**30**	BCDFG	**30**	H	*
games120	120	638	0.09	9	9	**9**	BCD	**9**	H	*
homer	561	1629	0.01	13	13	10	BCD	**13**	H	
huck	74	301	0.11	11	11	**11**	BCDFG	**11**	H	*
inithx.i.1	864	18707	0.05	54	54	**54**	BCDFG	**54**	H	*
inithx.i.2	645	13979	0.07	31	31	**31**	BCDFG	**31**	H	*
inithx.i.3	621	13969	0.07	31	31	**31**	BCDFG	**31**	H	*
jean	80	254	0.08	10	10	**10**	BCDFG	**10**	H	*
latin_square_10	900	307350	0.76	90	97	**90**	ABCD	130	H	
le450_15a	450	8168	0.08	15	15	**15**	BCD	17	BD	
le450_15b	450	8169	0.08	15	15	**15**	BCD	17	H	
le450_15c	450	16680	0.17	15	15	**15**	BCD	25	H	
le450_15d	450	16750	0.17	15	15	**15**	BCD	25	H	
le450_25a	450	8260	0.08	25	25	**25**	BCD	**25**	H	*
le450_25b	450	8263	0.08	25	25	**25**	BCD	**25**	H	*
le450_25c	450	17343	0.17	25	25	**25**	BCD	29	H	
le450_25d	450	17425	0.17	25	25	**25**	BCD	28	H	
le450_5a	450	5714	0.06	5	5	**5**	BCD	10	H	
le450_5b	450	5734	0.06	5	5	**5**	BCD	9	H	
le450_5c	450	9803	0.10	5	5	**5**	BCD	8	H	
le450_5d	450	9757	0.10	5	5	**5**	BCD	**5**	H	*
miles1000	128	3216	0.40	42	42	**42**	BCDEFG	**42**	H	*
miles1500	128	5198	0.64	73	73	**73**	ABCDEFG	**73**	H	*
miles250	128	387	0.05	8	8	**8**	BCDFG	**8**	H	*
miles500	128	1170	0.14	20	20	**20**	BCDFG	**20**	H	*
miles750	128	2113	0.26	31	31	**31**	BCDFG	**31**	H	*
mug100_1	100	166	0.03	4	4	3	ABCD	**4**	H	
mug100_25	100	166	0.03	4	4	3	ABCD	**4**	H	
mug88_1	88	146	0.04	4	4	3	ABCD	**4**	H	
mug88_25	88	146	0.04	4	4	3	ABCD	**4**	H	
mulsol.i.1	197	3925	0.20	49	49	**49**	ABCDEFG	**49**	H	*
mulsol.i.2	188	3885	0.22	31	31	**31**	ABCDEFG	**31**	H	*
mulsol.i.3	184	3916	0.23	31	31	**31**	ABCDEFG	**31**	H	*
mulsol.i.4	185	3946	0.23	31	31	**31**	ABCDEFG	**31**	H	*
mulsol.i.5	186	3973	0.23	31	31	**31**	ABCDEFG	**31**	H	*
myciel3	11	20	0.36	4	4	**4**	ABCDEFG	**4**	H	*
myciel4	23	71	0.28	5	5	**5**	ABCDEFG	**5**	H	*
myciel5	47	236	0.22	6	6	5	ABCDEFG	**6**	H	
myciel6	95	755	0.17	7	7	4	ABCDF	**7**	H	
myciel7	191	2360	0.13	8	8	4	ABCD	**8**	H	
qg.order100	10000	990000	0.02	100	100	**100**	ABCD	112	B	
qg.order30	900	26100	0.06	30	30	**30**	ABCD	32	ACD	
qg.order40	1600	62400	0.05	40	40	**40**	ABCD	45	H	
qg.order60	3600	212400	0.03	60	60	**60**	ABCD	64	ACD	
queen10_10	100	1470	0.30	11	11	10	ABCD	14	H	
queen11_11	121	1980	0.27	11	11	**11**	AB	15	BCD	
queen12_12	144	2596	0.25	12	12	**12**	AB	16	H	
queen13_13	169	3328	0.23	13	13	**13**	AB	17	H	
queen14_14	196	4186	0.22	14	14	**14**	AB	19	H	
queen15_15	225	5180	0.21	15	15	**15**	AB	21	H	
queen16_16	256	6320	0.19	16	17	**16**	AB	21	A	

Table 2 (Continued)

						Decision Diagram				
Instance	n	m	d	$\underline{\chi}$	$\overline{\chi}$	LB	Setting	UB	Setting	
queen5_5	25	160	0.53	5	5	**5**	ABCDEFG	**5**	H	*
queen6_6	36	290	0.46	7	7	**7**	ABCDEFG	**7**	ABCDEFG	*
queen7_7	49	476	0.40	7	7	**7**	ABCDEFG	**7**	ABCDEFG	*
queen8_12	96	1368	0.30	12	12	**12**	AB	**12**	H	*
queen8_8	64	728	0.36	9	9	**9**	ABCDEFG	**9**	ACDEFG	*
queen9_9	81	1056	0.33	10	10	**10**	ACD	11	AF	
r1000.1	1000	14378	0.03	20	20	**20**	BCD	**20**	H	*
r1000.1c	1000	485090	0.97	96	98	88	B	110	H	
r1000.5	1000	238267	0.48	234	234	214	BD	244	B	
r125.1	125	209	0.03	5	5	**5**	BCDFG	**5**	H	*
r125.1c	125	7501	0.97	46	46	**46**	ABCDEFG	**46**	H	*
r125.5	125	3838	0.50	36	36	**36**	BCDEFG	**36**	BCDEFG	*
r250.1	250	867	0.03	8	8	**8**	BCD	**8**	H	*
r250.1c	250	30227	0.97	64	64	**64**	ABCDEFG	**64**	ABCDEFG	*
r250.5	250	14849	0.48	65	65	**65**	BCDEFG	**65**	DEFG	*
school1	385	19095	0.26	14	14	**14**	BCD	**14**	BCD	*
school1_nsh	352	14612	0.24	14	14	**14**	BCD	**14**	BCD	*
wap01a	2368	110871	0.04	41	43	40	BCD	46	BCD	
wap02a	2464	111742	0.04	40	42	**40**	BCD	45	H	
wap03a	4730	286722	0.03	40	47	**40**	BCD	53	BCD	
wap04a	5231	294902	0.02	40	42	**40**	BCD	48	BCD	
wap05a	905	43081	0.11	50	50	**50**	CD	**50**	BCD	*
wap06a	947	43571	0.10	40	40	**40**	BCD	43	CD	
wap07a	1809	103368	0.06	40	41	**40**	B	46	BCD	
wap08a	1870	104176	0.06	40	42	**40**	B	45	H	
will199GPIA	701	6772	0.03	7	7	6	BCD	**7**	H	
zeroin.i.1	211	4100	0.19	49	49	**49**	ABCDEFG	**49**	H	*
zeroin.i.2	211	3541	0.16	30	30	**30**	ABCDEFG	**30**	H	*
zeroin.i.3	206	3540	0.17	30	30	**30**	ABCDEFG	**30**	H	*

Table 2 (Continued)

The best performing settings are C and D (the Max-Connected-Degree ordering without and with redirecting arcs), and B (the Dsatur ordering). Note also the strong performance of the Dsatur heuristic which finds the best upper bound for 109 instances.

8.3 Benchmark Comparison

Even though Table 2 provides a comparison with the state of the art in terms of the best lower and upper bounds from the literature, it is insightful to compare the running time of our approach to existing methods as well. As mentioned in Section 1, a number of approaches have been successfully applied to exact graph coloring. Among these, the branch-and-price method of Mehrotra and Trick [24] appears to be the most effective and robust general approach, in particular the implementation by Held, Cook, and Sewell [17]. Furthermore, both the Mehrotra-Trick approach and the decision diagram approach rely on the independent set formulation for graph coloring, which makes it natural to compare the two methods. We therefore selected the code by Held et al. [17] for our benchmark comparison.[2]

[2] The code has been downloaded from `https://github.com/heldstephan/exactcolors`.

					Held et al. [17]			Decision Diagram		
Instance	n	m	$\underline{\chi}$	$\overline{\chi}$	LB	UB	Time	LB	UB	Time
1-FullIns_3	30	100	4	4	**4**	**4**	0.01	**4**	**4**	0.06
1-FullIns_4	93	593	5	5	4	**5**	timeout	4	**5**	timeout
1-FullIns_5	282	3,247	6	6	4	**6**	timeout	4	**6**	timeout
1-Insertions_4	67	232	5	5	3	**5**	timeout	3	**5**	timeout
1-Insertions_5	202	1,227	6	6	3	**6**	timeout	3	**6**	timeout
1-Insertions_6	607	6,337	4	7	-	**7**	timeout	3	**7**	timeout
2-FullIns_3	52	201	5	5	**5**	**5**	0.01	**5**	**5**	0.83
2-FullIns_4	212	1,621	6	6	5	**6**	timeout	5	**6**	timeout
2-FullIns_5	852	12,201	7	7	5	**7**	timeout	4	**7**	timeout
2-Insertions_3	37	72	4	4	**4**	**4**	553.00	3	**4**	timeout
2-Insertions_4	149	541	5	5	3	**5**	timeout	3	**5**	timeout
2-Insertions_5	597	3,936	6	6	-	**6**	timeout	3	**6**	timeout
3-FullIns_3	80	346	6	6	**6**	**6**	0.03	**6**	**6**	13.15
3-FullIns_4	405	3,524	7	7	6	**7**	timeout	5	**7**	timeout
3-FullIns_5	2,030	33,751	8	8	6	**8**	timeout	5	**8**	timeout
3-Insertions_3	56	110	4	4	3	**4**	timeout	3	**4**	timeout
3-Insertions_4	281	1,046	5	5	-	**5**	timeout	3	**5**	timeout
3-Insertions_5	1,406	9,695	4	6	-	**6**	timeout	3	**6**	timeout
4-FullIns_3	114	541	7	7	**7**	**7**	0.05	**7**	**7**	99.58
4-FullIns_4	690	6,650	8	8	7	**8**	timeout	7	**8**	timeout
4-FullIns_5	4,146	77,305	9	9	-	**9**	timeout	7	**9**	timeout
4-Insertions_3	79	156	4	4	3	**4**	timeout	3	**4**	timeout
4-Insertions_4	475	1,795	5	5	-	**5**	timeout	3	**5**	timeout
5-FullIns_3	154	792	8	8	**8**	**8**	0.07	**8**	**8**	1233.00
5-FullIns_4	1,085	11,395	9	9	8	**9**	timeout	8	**9**	timeout
abb313GPIA	1,557	53,356	9	9	-	10	timeout	8	10	timeout
anna	138	493	11	11	**11**	**11**	0.02	**11**	**11**	0.01
ash331GPIA	662	4,181	4	4	**4**	6	timeout	**4**	5	timeout
ash608GPIA	1,216	7,844	4	4	-	6	timeout	3	5	timeout
ash958GPIA	1,916	12,506	4	4	-	6	timeout	3	5	timeout
C2000.5	2,000	999,836	99	145	-	207	timeout	20	208	timeout
C2000.9	2,000	1,799,532	-	400	-	550	timeout	84	563	timeout
C4000.5	4,000	4,000,268	107	259	-	376	timeout	20	381	timeout
david	87	406	11	11	**11**	**11**	0.01	**11**	**11**	0.01
DSJC1000.1	1,000	49,629	10	20	-	25	timeout	6	26	timeout
DSJC1000.5	1,000	249,826	73	82	-	114	timeout	19	119	timeout
DSJC1000.9	1,000	449,449	216	222	-	301	timeout	85	304	timeout
DSJC125.1	125	736	5	5	**5**	6	timeout	**5**	6	timeout
DSJC125.5	125	3,891	17	17	16	18	timeout	14	22	timeout
DSJC125.9	125	6,961	44	44	**44**	**44**	19.23	**44**	**44**	25.11
DSJC250.1	250	3,218	7	8	-	10	timeout	5	10	timeout
DSJC250.5	250	15,668	26	28	26	30	timeout	16	36	timeout
DSJC250.9	250	27,897	72	72	71	73	timeout	71	74	timeout
DSJC500.1	500	12,458	9	12	-	16	timeout	5	16	timeout
DSJC500.5	500	62,624	43	47	-	65	timeout	18	68	timeout
DSJC500.9	500	112,437	123	126	-	163	timeout	84	165	timeout
DSJR500.1	500	3,555	12	12	**12**	**12**	1173.00	**12**	**12**	0.03
DSJR500.1c	500	121,275	85	85	**85**	**85**	2428.00	**85**	**85**	1358.57
DSJR500.5	500	58,862	122	122	-	132	timeout	112	126	timeout
flat1000_50_0	1,000	245,000	50	50	-	113	timeout	19	115	timeout
flat1000_60_0	1,000	245,830	60	60	-	112	timeout	19	114	timeout
flat1000_76_0	1,000	246,708	72	81	-	115	timeout	19	116	timeout

Table 3 Comparing the performance of the branch-and-price implementation by Held et al. [17] and the decision diagram approach, using setting C. For each instance we list the number of nodes (n) and edges (m), and the best known lower bound ($\underline{\chi}$) and upper bound ($\overline{\chi}$). For each method, we report the lower bound (LB), upper bound (UB), and solving time (in seconds). The time limit was set to 3,600s. The decision diagram size never exceeded the limit of 1 million nodes. Bounds in bold meet the best known bounds.

					Held et al. [17]			Decision Diagram		
Instance	n	m	$\underline{\chi}$	$\overline{\chi}$	LB	UB	Time	LB	UB	Time
flat300_20_0	300	21,375	20	20	-	42	timeout	16	42	timeout
flat300_26_0	300	21,633	26	26	-	42	timeout	16	43	timeout
flat300_28_0	300	21,695	28	28	**28**	33	timeout	16	44	timeout
fpsol2.i.1	496	11,654	65	65	**65**	**65**	1.13	**65**	**65**	10.33
fpsol2.i.2	451	8,691	30	30	**30**	**30**	1.00	**30**	**30**	0.21
fpsol2.i.3	425	8,688	30	30	**30**	**30**	1.00	**30**	**30**	0.25
games120	120	638	9	9	**9**	**9**	0.02	**9**	**9**	92.74
homer	561	1,628	13	13	**13**	**13**	1.00	10	**13**	timeout
huck	74	301	11	11	**11**	**11**	0.01	**11**	**11**	0.23
inithx.i.1	864	18,707	54	54	**54**	**54**	3.46	**54**	**54**	3.17
inithx.i.2	645	13,979	31	31	**31**	**31**	1.00	**31**	**31**	121.09
inithx.i.3	621	13,969	31	31	**31**	**31**	1.00	**31**	**31**	163.70
jean	80	254	10	10	**10**	**10**	0.01	**10**	**10**	0.01
latin_square_10	900	307,350	90	97	-	129	timeout	90	130	timeout
le450_15a	450	8,168	15	15	-	17	timeout	**15**	18	timeout
le450_15b	450	8,169	15	15	-	17	timeout	**15**	17	timeout
le450_15c	450	16,680	15	15	-	24	timeout	**15**	25	timeout
le450_15d	450	16,750	15	15	-	24	timeout	**15**	25	timeout
le450_25a	450	8,260	25	25	**25**	**25**	3.05	**25**	**25**	0.17
le450_25b	450	8,263	25	25	**25**	**25**	2.46	**25**	**25**	0.14
le450_25c	450	17,343	25	25	-	28	timeout	**25**	29	timeout
le450_25d	450	17,425	25	25	-	29	timeout	**25**	28	timeout
le450_5a	450	5,714	5	5	-	10	timeout	**5**	10	timeout
le450_5b	450	5,734	5	5	-	7	timeout	**5**	9	timeout
le450_5c	450	9,803	5	5	-	11	timeout	**5**	8	timeout
le450_5d	450	9,757	5	5	-	11	timeout	**5**	**5**	0.03
miles1000	128	3,216	42	42	**42**	**42**	1.00	**42**	**42**	1.87
miles1500	128	5,198	73	73	**73**	**73**	0.41	**73**	**73**	1.10
miles250	128	387	8	8	**8**	**8**	0.02	**8**	**8**	0.01
miles500	128	1,170	20	20	**20**	**20**	0.04	**20**	**20**	0.04
miles750	128	2,113	31	31	**31**	**31**	0.16	**31**	**31**	0.12
mug100_1	100	166	4	4	**4**	**4**	6.00	3	**4**	timeout
mug100_25	100	166	4	4	**4**	**4**	4.19	3	**4**	timeout
mug88_1	88	146	4	4	**4**	**4**	3.00	3	**4**	timeout
mug88_25	88	146	4	4	**4**	**4**	3.33	3	**4**	timeout
mulsol.i.1	197	3,925	49	49	**49**	**49**	0.30	**49**	**49**	0.40
mulsol.i.2	188	3,885	31	31	**31**	**31**	1.00	**31**	**31**	0.19
mulsol.i.3	184	3,916	31	31	**31**	**31**	0.08	**31**	**31**	0.19
mulsol.i.4	185	3,946	31	31	**31**	**31**	0.11	**31**	**31**	0.19
mulsol.i.5	186	3,973	31	31	**31**	**31**	0.16	**31**	**31**	0.19
myciel3	11	20	4	4	**4**	**4**	0.02	**4**	**4**	0.04
myciel4	23	71	5	5	**5**	**5**	12.00	**5**	**5**	7.03
myciel5	47	236	6	6	4	**6**	timeout	5	**6**	timeout
myciel6	95	755	7	7	4	**7**	timeout	4	**7**	timeout
myciel7	191	2,360	8	8	5	**8**	timeout	4	**8**	timeout
qg.order100	10,000	990,000	100	100	-	106	timeout	**100**	116	timeout
qg.order30	900	26,100	30	30	-	32	timeout	**30**	32	timeout
qg.order40	1,600	62,400	40	40	-	42	timeout	**40**	45	timeout
qg.order60	3,600	212,400	60	60	-	63	timeout	**60**	64	timeout
queen10_10	100	2,940	11	11	**11**	**11**	781.00	10	14	timeout
queen11_11	121	3,960	11	11	**11**	12	timeout	10	15	timeout
queen12_12	144	5,192	12	12	**12**	13	timeout	10	16	timeout
queen13_13	169	6,656	13	13	**13**	15	timeout	10	17	timeout
queen14_14	196	4,186	14	14	**14**	16	timeout	10	19	timeout
queen15_15	225	5,180	15	15	**15**	17	timeout	10	21	timeout
queen16_16	256	12,640	16	17	-	21	timeout	10	22	timeout

Table 3 (Continued)

					Held et al. [17]			Decision Diagram		
Instance	n	m	$\underline{\chi}$	$\overline{\chi}$	LB	UB	Time	LB	UB	Time
queen5_5	25	160	5	5	**5**	**5**	0.00	**5**	**5**	0.01
queen6_6	36	290	7	7	**7**	**7**	1.00	**7**	**7**	2.35
queen7_7	49	476	7	7	**7**	**7**	1.13	**7**	**7**	2.77
queen8_12	96	1,368	12	12	**12**	**12**	18.35	9	**12**	timeout
queen8_8	64	728	9	9	**9**	**9**	10.19	**9**	**9**	310.31
queen9_9	81	1,056	10	10	**10**	**10**	24.00	**10**	12	timeout
r1000.1	1,000	14,378	20	20	**20**	**20**	2.43	**20**	**20**	0.20
r1000.1c	1,000	485,090	96	98	-	107	timeout	82	110	timeout
r1000.5	1,000	238,267	234	234	-	248	timeout	213	246	timeout
r125.1	125	209	5	5	**5**	**5**	0.01	**5**	**5**	0.02
r125.1c	125	7,501	46	46	**46**	**46**	1.00	**46**	**46**	2.08
r125.5	125	3,838	36	36	**36**	**36**	34.55	**36**	**36**	212.90
r250.1	250	867	8	8	**8**	**8**	0.05	**8**	**8**	7.32
r250.1c	250	30,227	64	64	**64**	**64**	103.00	**64**	**64**	31.19
r250.5	250	14,849	65	65	**65**	**65**	592.00	**65**	67	timeout
school1	385	19,095	14	14	**14**	**14**	3065.00	**14**	**14**	1.67
school1_nsh	352	14,612	14	14	**14**	**14**	2463.00	**14**	**14**	15.59
wap01a	2,368	110,871	41	43	-	47	timeout	40	46	timeout
wap02a	2,464	111,742	40	42	-	46	timeout	40	45	timeout
wap03a	4,730	286,722	40	47	-	57	timeout	40	53	timeout
wap04a	5,231	294,902	40	42	-	46	timeout	40	48	timeout
wap05a	905	43,081	50	50	**50**	**50**	11.41	**50**	**50**	15.80
wap06a	947	43,571	40	40	-	44	timeout	**40**	43	timeout
wap07a	1,809	103,368	40	41	-	47	timeout	38	46	timeout
wap08a	1,870	104,176	40	42	-	44	timeout	39	45	timeout
will199GPIA	701	6,772	7	7	**7**	**7**	15.03	6	**7**	timeout
zeroin.i.1	211	4,100	49	49	**49**	**49**	0.17	**49**	**49**	0.32
zeroin.i.2	211	3,541	30	30	**30**	**30**	0.09	**30**	**30**	0.17
zeroin.i.3	206	3,540	30	30	**30**	**30**	0.10	**30**	**30**	0.20

Table 3 (Continued)

We compiled the code from Held, Cook, and Sewell on the same machine as our decision diagram implementation, and it uses the same version of CPLEX. We apply the code to all 137 DIMACS instances, with the same time limit as the decision diagrams (1 hour). For a fair comparison, we only report the results for a fixed setting (setting C) for the decision diagram approach. Table 3 reports the lower and upper bounds as well as the running times for both methods. A high-level summary shows that the methods are competitive (we refer to Held, Cook, and Sewell as 'HCS' and to the decision diagram approach as 'DD'):

- HCS solves more instances optimally (60) than DD (50).
- HCS is unable to return a lower bound (within the time limit) for 50 instances. DD returns a lower bound for all instances.
- The methods find a similar number of best known lower bounds (70 for HCS and 73 for DD) and best known upper bounds (82 for HCS and 80 for DD).
- In a relative comparison, HCS finds 21 better lower bounds than DD, while DD finds 51 better lower bounds than HCS. For 65 instances they find the same lower bound.
- HCS finds 37 better upper bounds than DD, while DD finds 13 better upper bounds than HCS. For 87 instances they find the same upper bound.

Instance	n	m	d	$\underline{\chi}$	$\overline{\chi}$	LB	TTLB	Setting	UB	TTUB	Setting
1-Insertions_6	607	6,337	0.03	4	7	3	0.0	A	7	0.0	H
3-Insertions_5	1,406	9,695	0.01	4	6	3	0.1	A	6	0.0	H
C2000.5	2,000	999,836	0.50	99	145	20	823.1	D	208	0.6	H
C2000.9	2,000	1,799,532	0.90	98	400	**145**	4.7 days	D	563	1.2	H
C4000.5	4,000	4,000,268	0.50	107	259	20	1,640.3	C	381	6.0	H
DSJC1000.1	1,000	49,629	0.10	10	20	6	3.1	B	26	0.0	H
DSJC1000.5	1,000	249,826	0.50	73	82	19	1,975.0	C	119	0.1	H
DSJC1000.9	1,000	449,449	0.90	216	222	86	3,290.7	D	304	0.2	H
DSJC250.1	250	3,218	0.10	6	8	5	0.0	B	10	0.0	H
DSJC250.5	250	15,668	0.50	26	28	16	594.5	C	36	0.0	H
DSJC500.1	500	12,458	0.10	9	12	5	0.1	C	16	0.0	H
DSJC500.5	500	62,624	0.50	43	47	18	1,317.3	C	68	0.0	H
DSJC500.9	500	112,437	0.90	123	126	123	27.3	F	132	10.7h	F
flat1000_76_0	1,000	246,708	0.49	72	81	19	3,052.6	C	116	0.1	H
latin_square_10	900	307,350	0.76	90	97	90	7.7	C	130	0.1	H
queen16_16	256	6,320	0.19	16	17	16	0.0	A	21	1.3	A
r1000.1c	1,000	485,090	0.97	96	98	88	2,985.7	B	110	0.1	H
wap01a	2,368	110,871	0.04	41	43	40	8.1	C	46	1.3	C
wap02a	2,464	111,742	0.04	40	42	40	3.1	B	45	0.2	H
wap03a	4,730	286,722	0.03	40	47	40	6.1	C	53	4.9	C
wap04a	5,231	294,902	0.02	40	42	40	7.3	B	48	2.4	C
wap07a	1,809	103,368	0.06	40	41	40	291.2	B	46	1.5	C
wap08a	1,870	104,176	0.06	40	42	40	3,224.2	B	45	0.1	H

Table 4 Performance of the decision diagram approach on the set of open DIMACS instances. For each instance we list the number of nodes (n) and edges (m), edge density (d), and the best known lower bound ($\underline{\chi}$) and upper bound ($\overline{\chi}$). We report the lower bound (LB), time to lower bound (TTB), upper bound (UB), time to upper bound (TTUB), and the settings obtaining the best times. For all instances except `C2000.9` and `DSJC500.9` the times are given in seconds and a time limit of 3,600s was imposed. The instances `C2000.9` and `DSJC500.9` were run for 4.7 days and 10.7 hours, respectively.

8.4 Results on Open Instances

The last set of experiments, presented in Table 4, investigates the quality of the bounds of the iterative refinement procedure and exact compilation on the set of open DIMACS instances. For each instance we report the best lower bound and upper bound, as well the time to find those bounds and the associated algorithmic setting. For instance `C2000.9`, we report an improved lower bound of value 145, after running the algorithm for 4.7 days.

Instance `DSJC500.9` is an interesting case, because the exact decision diagram (using the Dsatur variable ordering; setting F) contains 779,179 nodes. While this yields a large integer program, the MIP presolve procedure of CPLEX substantially reduces its size to 3,553 rows, 25,445 columns, and 83,106 nonzeros. It quickly finds a lower bound of value 123, and an upper bound of 136 (the Dsatur heuristic finds an upper bound of value 165). It takes 10.7 hours for CPLEX to further improve the upper bound to value 132. Even after 2.7 days, no further bound improvements were reported, however.

9 Conclusion

We introduced a new approach for solving graph coloring problems, based on a decision diagram representation of the possible color classes. As exact decision diagrams may grow exponentially large, we proposed an iterative refinement scheme that operates on relaxed decision diagrams instead. By solving a constrained minimum network flow problem defined over the relaxed decision diagrams, we computed a lower bound on the chromatic number. We showed how the network flow solution can be decomposed into paths that are inspected for edge conflicts. These conflicts are then separated in the decision diagram, resulting in an iterative refinement procedure yielding increasingly stronger bounds. In addition, we developed a primal heuristic based, again, on a path decomposition of the network flow solution.

We showed both theoretically and experimentally that relaxed decision diagrams can be orders of magnitude smaller than exact diagrams when proving optimality. We demonstrated that decision diagrams can be used to solve 54 out of 137 DIMACS instances to optimality, of which 46 were solved within 1 minute and 36 within one second. Moreover, we compared our method to a state-of-the-art exact graph coloring solver based on branch-and-price, and obtained competitive results. Lastly, we computed an improved lower bound for the open instance `C2000.9`.

Acknowledgements

This work was partially supported by Office of Naval Research Grant No. N00014-18-1-2129 and National Science Foundation Award #1918102.

References

1. S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27:509–516, 1978.
2. H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A Constraint Store Based on Multivalued Decision Diagrams. In *Proceedings of CP*, volume 4741 of *LNCS*, pages 118–132. Springer, 2007.
3. N. Barnier and P. Brisset. Graph Coloring for Air Traffic Flow Management. *Annals of Operations Research*, 130:163–178, 2004.
4. D. Bergman and A. A. Cire. On Finding the Optimal BDD Relaxation. In *Proceedings of CPAIOR*, volume 10335 of *LNCS*, pages 41–50. Springer, 2017.
5. D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. *Decision Diagrams for Optimization*. Springer, 2016.
6. D. Bergman, A. A. Cire, W.-J. van Hoeve, and Hooker J. N. Variable Ordering for the Application of BDDs to the Maximum Independent Set Problem. In *Proceedings of CPAIOR*, volume 7298 of *LNCS*, pages 34–49. Springer, 2012.
7. D. Bergman, A. A. Cire, W.-J. van Hoeve, and Hooker J. N. Optimization Bounds from Binary Decision Diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014.
8. D. Bergman, A. A. Cire, W.-J. van Hoeve, and Hooker J. N. Discrete Optimization with Decision Diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.
9. D. Bergman, W.-J. van Hoeve, and J. N. Hooker. Manipulating MDD Relaxations for Combinatorial Optimization. In *Proceedings of CPAIOR*, volume 6697 of *LNCS*, pages 20–35. Springer, 2011.
10. D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
11. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

12. R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24:293– 318, 1992.
13. A. A. Cire and J. N. Hooker. The separation problem for binary decision diagrams. In *Proceedings of ISAIM*, 2014.
14. F. Furini, V. Gabrel, and I.-C. Ternier. An Improved DSATUR-Based Branch-and-Bound Algorithm for the Vertex Coloring Problem. *Networks*, 69(1):124–141, 2017.
15. M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
16. S. Gualandi and Malucelli F. Exact Solution of Graph Coloring Problems via Constraint Programming and Column Generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.
17. S. Held, W. Cook, and E. C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012.
18. W.-J. van Hoeve. Graph Coloring Lower Bounds from Decision Diagrams. In D. Bienstock and G. Zambelli, editors, *Proceedings of IPCO*, volume 12125 of *Lecture Notes in Computer Science*, pages 405–418. Springer, 2020.
19. A. Jabrayilov and P. Mutzel. New Integer Linear Programming Models for the Vertex Coloring Problem. In *Proceedings of LATIN*, volume 10807 of *LNCS*, pages 640–652. Springer, 2018.
20. D. S. Johnson and M. A Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.
21. C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.
22. R. Lewis and J. Thompson. On the application of graph colouring techniques in round-robin sports scheduling. *Computers & Operations Research*, 38(1):190–204, 2011.
23. E. Malaguti, M. Monaci, and P. Toth. An exact approach for the Vertex Coloring Problem. *Discrete Optimization*, 8:174–190, 2011.
24. A. Mehrotra and M. A. Trick. A Column Generation Approach for Graph Coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.
25. I. Méndez-Díaz and P. Zabala. A Branch-and-Cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154:826–847, 2006.
26. I. Méndez-Díaz and P. Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156:159–179, 2008.
27. D. R Morrison, E. C. Sewell, and S. H. Jacobson. Solving the Pricing Problem in a Branch-and-Price Algorithm for Graph Coloring Using Zero-Suppressed Binary Decision Diagrams. *INFORMS Journal on Computing*, 28(1):67–82, 2016.
28. J. Peemöller. A correction to Brelaz's modification of Brown's coloring algorithm. *Communications of the ACM*, 26(8):595–597, 1983.
29. G. Perez and J.-C. Régin. Constructions and In-Place Operations for MDDs Based Constraints. In *Proceedings of CPAIOR*, volume 9676 of *LNCS*, pages 279–293. Springer, 2016.
30. J. Randall-Brown. Chromatic scheduling and the chromatic number problem. *Management Science*, 19(4):456–463, 1972.
31. M. Römer, A. A. Cire, and L.-M. Rousseau. A Local Search Framework for Compiling Relaxed Decision Diagrams. In *Proceedings of CPAIOR*, volume 10848 of *LNCS*, pages 512–520. Springer, 2018.
32. P. San Segundo. A new DSATUR-based algorithm for exact vertex coloring. *Computers & Operations Research*, 39:1724–1733, 2012.
33. A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.
34. I. Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, 2000.
35. D. C. Wood. A technique for coloring a graph applicable to large-scale timetabling problems. *The Computer Journal*, 12(4):317–322, 1969.