Graph Coloring Lower Bounds from Decision Diagrams

Willem-Jan van Hoeve[0000−0002−0023−753X]⋆

Carnegie Mellon University, Pittsburgh PA 15213, USA
vanhoeve@andrew.cmu.edu

Abstract. We introduce an iterative framework for computing lower bounds to graph coloring problems. We utilize relaxed decision diagrams to compactly represent an exponential set of color classes, or independent sets, some of which may contain edge conflicts. Our procedure uses minimum network flow models to compute lower bounds on the coloring number 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 provide an experimental evaluation on benchmark instances, and report an improved lower bound for one open instance.

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 [26, 16, 3, 18]. The most efficient exact solution methods are the Randall-Brown algorithm using the Dsatur vertex ordering [23, 8, 22], integer linear programming [15], and column generation (branch-and-price) [20, 19, 14, 13, 21].

A major challenge for exact graph coloring methods is to find 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, 7, 6].

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

⋆ Partially supported by Office of Naval Research Grant No. N00014-18-1-2129 and National Science Foundation Award #1918102.

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.

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 columns.

Contributions. The main contributions of this work include 1) the introduction of a new framework for obtaining graph coloring lower 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 lower bounds on benchmark instances, with an improved lower bound for one open instance.

2 Graph Coloring by Independent Sets

We first present a formal definition of graph coloring [24]. 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 $\{1, \dots, n\}$. 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 variables 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 y_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 $i \in I$. The graph coloring problem can then be formulated as the following integer program:

$$\begin{aligned} \min\ & \textstyle\sum_{i \in I} y_i \\ \text{s.t. } & \textstyle\sum_{i \in I} a_{ij} y_i = 1 \quad \forall j \in V, \\ & y_i \in \{0, 1\} \qquad \forall i \in I, \end{aligned} \tag{1}$$

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 in [20]. 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 start with all subsets sets I and iteratively remove sets that contain adjacent vertices.

3 Decision Diagrams

Decision diagrams were originally developed to represent switching circuits and, more generally, Boolean functions [17, 1, 25]. They became particularly popular after the introduction of efficient compilation methods for Reduced Ordered Binary Decision Diagrams [9, 10], and have been applied widely to verification and configuration problems. More recently, decision diagrams have been applied to solve optimization problems [6], which is the context we follow in this paper.

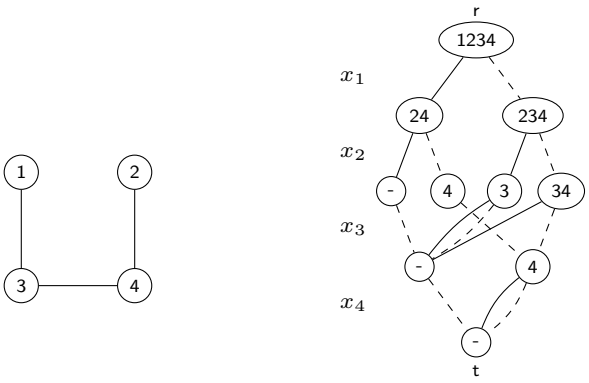
Definitions For our purposes, a decision diagram will represent the set of solutions to an optimization problem P defined on an ordered set of decision variables $X = \{x_1, \ldots, x_n\}$. In this paper, we assume that each variable is binary. The feasible set of P is denoted by $\text{Sol}(P)$.

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. 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 $\text{Sol}(D)$ the collection of variable assignments for all r-t paths in D.

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

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 [9]. Nonetheless, even reduced decision diagrams may be exponentially large to represent all solutions for a given problem.

Exact Compilation In this work we apply top-down compilation methods that depend on state-dependent information (similar to state variables in dynamic



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.

programming models). We limit the exposition to the compilation of decision diagrams for independent set problems [4, 5]. 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 $S(r) = V$, $S(t) = \varnothing$. 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

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

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. In our case, two nodes u and v are equivalent if $S(u) = S(v)$. This top-down compilation procedure yields the unique reduced decision diagram for representing all independent sets (for a given ordering), as shown in [4, 5].

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.

Compilation by Separation As an alternative to exact compilation, we can apply *constraint separation* to iteratively construct the decision diagram [11, 6]. We will apply this method to compile *relaxed* decision diagrams, and again describe it in the context of independent sets.

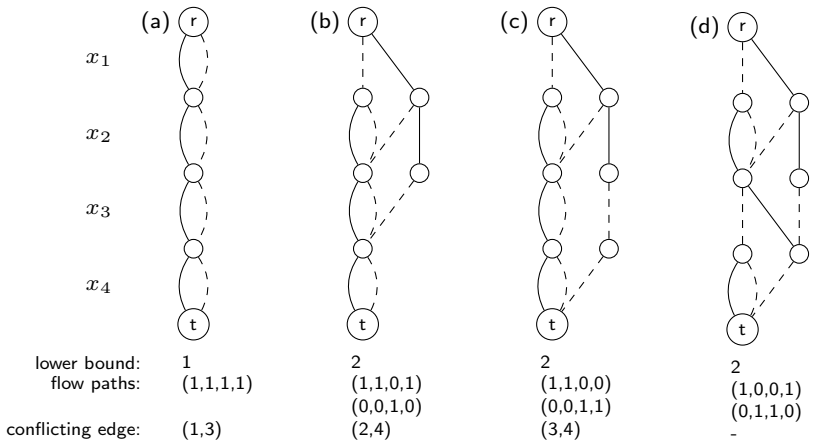


Fig. 2. Applying constraint separation to the input graph of Example 1. The initial relaxed diagram (a) is iteratively refined until the optimal solution (the flow paths) no longer contains infeasible color classes.

We initialize each layer j of D as a single node u_j, with state information $S(u_j) = \{j, \ldots, n\}$, for $j = 1, \ldots, n$, and $S(u_{n+1}) = \varnothing$. We define a 0-arc and a 1-arc between nodes u_j and u_{j+1}, for $j = 1, \ldots, n$. The input to our separation algorithm is a relaxed decision diagram D together with a path $p = u_j, u_{j+1}, \ldots, u_k$ with associated arc labels $l_j, l_{j+1}, \ldots, l_{k-1}$, and a conflict, i.e., the edge between vertices j and k, where $j < k$. The goal of the separation algorithm is to resolve the conflict along path p by splitting nodes, and arcs, appropriately. This is described in Algorithm 1.

We represent D as a two-dimensional vector D[][] of nodes, such that D is a vector of 'layers' and 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 D[]. The root is represented as D[1][1] and the terminal as D[$n+1$][1]. Each node u = D[j][i] (u is the i-th node in layer j) has state information D[j][i].S $= S(u)$, a reference to the node in layer $j + 1$ that represents the endpoint of its 1-arc D[j][i].oneArc, and a similar reference for its 0-arc D[j][i].zeroArc. If the 1-arc does not exist, the reference holds value -1.

Algorithm 1 considers each node D[i][u_i] along the path in sequence (line 2) and splits off the next node in the path. This is done by first creating a temporary node w (lines 3-5). If an equivalent node already exists in layer $i + 1$, we direct the path to its index (lines 6-7). Otherwise we complete the definition of w by copying the outgoing arcs of node D[$i + 1$][u_{i+1}] (lines 9-11), and we add w to layer $i + 1$ (lines 12-13). Lastly, we redirect the path from u_i to the new node with index t (lines 15-17). Fig. 2 gives an illustration of the algorithm. 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).

Algorithm 1: Compilation by separation.

Input: relaxed decision diagram D ($\mathsf{D[j][i]}$ represents the ith node in layer j), path node indices $u_j, \ldots, u_{k-1}$, path arc labels $l_j, \ldots, l_{k-1}$, conflict (j, k), and list of neighbors N_i for each $i \in V$
for $i = j, \ldots, k-1$ **do**
  create node w // goal is to split the path towards node w
  $w.\mathsf{S} \leftarrow \mathsf{D}[i][u_i].\mathsf{S} \setminus \{i\}$ // copy the state and remove i
  if $l_i = 1$ **then** $S(w) \leftarrow S(w) \setminus N_i$ // remove N_i for 1-arc
  $t \leftarrow -1$ // t is index of the new node in layer $i+1$
  if $\exists k$ *such that* $\mathsf{D}[i+1][k].\mathsf{S} = w.\mathsf{S}$ **then** $t \leftarrow k$ // equivalent node
  if $t = -1$ **then**
    if $i+1 \in w.\mathsf{S}$ **then** $w.\mathsf{oneArc} = \mathsf{D}[i+1][u_{i+1}].\mathrm{oneArc}$ // copy 1-arc
    else $w.\mathsf{oneArc}$ = -1
    $w.\mathsf{zeroArc} = \mathsf{D}[i+1][u_{i+1}].\mathrm{zeroArc}$ // copy 0-arc
    $\mathsf{D}[i+1].\mathrm{add}(w)$ // append w as new node to layer $i+1$
    $t \leftarrow |\mathsf{D}[i+1]|$ // update t to last index of layer $i+1$
  end
  if $l_i = 1$ **then** $\mathsf{D}[i][u_i].\mathsf{oneArc} = t$ // re-direct path in case of 1-arc
  else $\mathsf{D}[i][u_i].\mathsf{zeroArc} = t$ // re-direct path in case of 0-arc
  $u_{i+1} = t$ // update path index
end

Compilation by constraint separation proceeds in iterations by gradually refining the relaxed decision diagram. For our model, it has some useful properties:

Lemma 1. *Each decision diagram that is compiled by separation is reduced, i.e., no two nodes on any layer are equivalent.*

Lemma 2. *Compilation by separation will terminate with an exact decision diagram if each possible conflict, along any path, is separated.*

These two lemmas follow from the fact that for independent set problems, the state information is sufficient to prove equivalence [4, 5]. Lemma 1 will ensure that our approach produces the smallest decision diagram at each iteration, while Lemma 2 will guarantee termination and optimality.

4 Network Flow Model

We next formulate the graph coloring problem based on independent sets as a network flow problem on the decision diagram. 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:

$$(F) = \min \sum_{a \in \delta^+(r)} y_a \tag{2}$$

$$\text{s.t.} \sum_{a=(u,v) | L(u)=j, \ell(a)=1} y_a = 1 \qquad \forall j \in V \tag{3}$$

$$\sum_{a \in \delta^-(u)} y_a - \sum_{a \in \delta^+(u)} y_a = 0 \qquad \forall u \in N \setminus \{r, t\} \tag{4}$$

$$y_a \in \{0, 1, \ldots, n\} \qquad \forall a \in A \tag{5}$$

The objective function (2) minimizes the total amount of flow. Constraints (3) define that in each layer exactly one 1-arc is selected. Constraints (4) ensure flow conservation. Constraints (5) make sure the flow is integer.

Lemma 3. *A solution to model (F) corresponds to a partition of vertex set V.*

Proof. The partition can be found by decomposing the flow into paths. For vertex $i \in V$ there is exactly one arc $a = (u, v)$ for which $y_a = 1$, by constraints (3). By constraints (4) 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. Let P be the collection of such r-t paths; the cardinality of P is given by the objective function (2). Each path $(a_1, a_2, \ldots, a_n) \in P$ corresponds to a subset of vertices $\{i \mid \ell(a_i) = 1\}$. By constraints (3), these subsets are disjoint. □

Theorem 1. *If the decision diagram is exact, model (F) finds an optimal solution to the graph coloring problem.*

Proof. Since each path in the exact decision diagram corresponds to an independent set, the theorem follows from Lemma 3. □

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 graph coloring problem.*

One may wonder whether model (F) can be solved in polynomial time, since the NP-hardness of graph coloring may be accounted for by the worst-case exponential size of the decision diagram. The answer, however, is negative:

Theorem 2. *Solving model (F) for an arbitrary decision diagram is NP-hard.*

The proof follows from a reduction from minimum set partitioning and is given in the appendix. Note that paths in the solution are not explicitly given, but instead follow from the arc flow values. Moreover, paths can share nodes (and 0-arcs). We can therefore restrict the proof of Theorem 2 to be paths corresponding to a solution to the minimum set partitioning problem:

Corollary 2. *If the decision diagram is relaxed, deciding whether a given solution to model (F) corresponds to a feasible graph coloring solution is NP-hard.*

5 Iterative Refinement Procedure

While the worst-case complexity results in the previous section are negative, we can, however, efficiently identify a conflict in a given solution:

Algorithm 2: Iterative refinement by conflict detection and separation.

```
Input: decision diagram D and input graph data
foundSol ← false
while foundSol = false do
    solve model (F) with decision diagram D
    lowerBound ← obj(F)
    if flow decomposition algorithm finds no conflict then foundSol ← true
    else separate conflict in D using Algorithm 1
end
```

Theorem 3. *For a given solution to (F) we can in polynomial time (in the size of the decision diagram) either determine feasibility or identify a subset of vertices that creates an infeasibility.*

The proof, given in the appendix, is constructive and based on a path decomposition of the network flow. We apply this path decomposition to identify and separate conflicts inside an iterative refinement procedure, described at a high-level in Algorithm 2. The algorithm repeatedly solves the flow model, the objective of which provides a lower bound. If the solution contains a conflict, the decision diagram is refined, and we repeat the process. Otherwise, we found an optimal solution and terminate. Note that depending on the flow decomposition, the same solution to (F) might either return a feasible solution or a conflict.

Example 2. 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 (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 diagram (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 (d) and identify flow paths without conflicts.

Example 2 shows that iterative refinement may yield relaxed decision diagram that are smaller than exact decision diagrams, to prove optimality. This is formalized in the following key result, the proof of which is given in the appendix:

Theorem 4. *The iterative refinement procedure can find a provably optimal solution with a relaxed decision diagram that is exponentially smaller than the exact diagram that is defined on the same variable ordering.*

6 Implementation and Experimental Results

We implemented our method in C++, and performed an experimental evaluation on the 137 graph coloring benchmark instances obtained from [12], which includes all DIMACS graph coloring instances [16]. We use CPLEX 12.9 as integer and linear programming solver. The decision diagrams have a given maximum size (either 100,000 or 1,000,000 nodes).

IP > LP+IP	2
LP+IP > IP	31
LP+IP = IP	104
(LP+IP)/IP	0.53 (s.d.=0.38)

a. Impact of adding LP to IP

single > multiple	3
multiple > single	43
multiple = single	91
multiple/single	0.36 (s.d.=0.39)

b. Single vs. multiple conflicts

Table 1. Evaluating the impact of solving linear programming models before solving the integer programming models (a) and single conflict and multiple conflict resolution (b). In each case, we first list the number of instances for which either method finds a better bound. For those cases with equal bounds, we report the ratio of the average time to the best solution and the standard deviation.

Implementation Details The single most important parameter that influences the performance of the algorithm is the variable ordering of the decision diagram. We apply the following variable ordering heuristic, which dominated all other heuristics we tested: 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.

Second, observe that solving the continuous linear programming relaxation of model (F) provides a valid lower bound, as well as a path decomposition that can be used to identify conflicts. While the LP bound may be weaker than the IP bound, it is faster to compute and may therefore speed up the overall process. We implemented this by starting with LP-based iterative refinement, which is continued until a conflict-free LP optimum solution is found. After that, we continue with the IP-based iterative refinement to prove integer optimality.

Third, instead of resolving a single conflict (separation) in each iteration, it is possible to resolve multiple conflicts; one for each path in the decomposition (if it exists). The validity of identifying and separating multiple conflicts per iteration relies on the specific state information for independent sets, as well as on the indexing of our data structure D[][].

We ran experiments to assess the impact of adding the LP-based iterative refinement and multiple conflicts. The results are shown in Table 1. For each feature, we ran our algorithm with and without that feature, on all 137 benchmark instances for a maximum of 300s per instance. The impact is measured by the quality of the bound (number of instances with a better bound), and when the bounds are equal, by the computation time to the best bound. Both using LP and multiple conflicts have a substantial positive impact on the performance, reducing the time to the best bound by a factor 0.53 (LP), resp. 0.38 (multiple conflicts).

We added two other features to streamline the solving process. After reading in the data, we run the Dsatur heuristic to quickly find an upper bound, to help prove optimality in some cases. Furthermore, before running the LP and IP-based iterative refinement, we run a refinement procedure based on the longest path (with respect to 1-arcs) on the decision diagram, for at most 100 iterations.

Instance	n	m	d	$\underline{\chi}$	$\overline{\chi}$	Relaxed DD LB	Relaxed DD TTB
abb313GPIA	1,557	53,356	0.04	8	10	7	1.71
C2000.9	2,000	1,799,532	0.90	0	400	**98**	1,826.92
DSJC250.1	250	3,218	0.10	6	8	5	0.01
latin_square_10	900	307,350	0.76	90	97	90	0.73
wap01a	2,368	110,871	0.04	41	48*	40	1.20
wap02a	2,464	111,742	0.04	40	45*	40	1.61
wap03a	4,730	286,722	0.03	40	56*	40	0.88
wap04a	5,231	294,902	0.02	40	42	40	1.05
wap06a	947	43,571	0.10	40	54*	40	0.15
wap07a	1,809	103,368	0.06	40	41	39	705.66
wap08a	1,870	104,176	0.06	40	45*	39	107.23

Table 2. Performance of the relaxed decision diagram on a selection of open 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 ($\overline{\chi}$). For the relaxed decision diagram, we report the lower bound (LB) and time to best bound (TTB). The time limit was set to 3,600s (the maximum size of 1,000,000 was never exceeded). Upper bounds marked with an asterisk were computed with the Dsatur heuristic.

Experimental Analysis The aim of our first experiment is to compare the performance of the iterative refinement and the exact compilation. We consider all instances that are solved to optimality by either method—47 instances in total. The details can be found in Table 3 in the Appendix. The exact decision diagram can be remarkably small, which allows solving 37 instances directly. 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 4 in practice.

The second set of experiments, presented in Table 2, investigates the quality of the bounds of the iterative refinement procedure. The table considers a selection of open instances.[1] We report the lower bound (LB) and the time to the best lower bound in seconds (TTB). We were able to improve the lower bound for instance C2000.9 (marked in bold).

7 Conclusion

We introduced a new approach for obtaining lower bounds to graph coloring problems, by solving a minimum network flow problem defined over a relaxed decision diagram. By separating conflicts in the network flow solution, the relaxed decision diagram is iteratively refined, which results in stronger bounds. We showed both theoretically and experimentally that relaxed decision diagrams can be orders of magnitude smaller than exact diagrams when proving optimality. This allowed to find an improved lower bound for one open benchmark instance.

[1] According to the website with benchmark results [12] – accessed November 29, 2019.

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. B. 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.
5. B. 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.
6. D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. *Decision Diagrams for Optimization*. Springer, 2016.
7. 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.
8. D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
9. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.
10. R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24:293– 318, 1992.
11. A. A. Cire and J. N. Hooker. The separation problem for binary decision diagrams. In *Proceedings of ISAIM*, 2014.
12. S. Gualandi and M. Chiarandini. Graph Coloring Benchmarks. https://sites.google.com/site/graphcoloring/home.
13. 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.
14. 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.
15. 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.
16. 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.
17. C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.
18. 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.
19. E. Malaguti, M. Monaci, and P. Toth. An exact approach for the Vertex Coloring Problem. *Discrete Optimization*, 8:174–190, 2011.

20. A. Mehrotra and M. A. Trick. A Column Generation Approach for Graph Coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.
21. 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.
22. J. Peemöller. A correction to Brelaz's modification of Brown's coloring algorithm. *Communications of the ACM*, 26(8):595–597, 1983.
23. J. Randall-Brown. Chromatic scheduling and the chromatic number problem. *Management Science*, 19(4):456–463, 1972.
24. A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.
25. 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.
26. D. C. Wood. A technique for coloring a graph applicable to large-scale timetabling problems. *The Computer Journal*, 12(4):317–322, 1969.

Graph Coloring Lower Bounds from Decision Diagrams: Appendix

Proof of Theorem 2. 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 polynomial-size 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 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. An optimal solution to model (F) directly corresponds to solving the minimum set partitioning problem. □

Proof of Theorem 3. We apply a flow decomposition algorithm that iteratively finds r-t paths with flow value 1, starting from the root. The algorithm maintains the set of vertices i for which $y_a = 1$, $\ell(a) = 1$ and $L(a) = i$. Each time a vertex i is added to this set, we inspect whether there exists an edge $(i, j) \in E$ with j in the set. If so, we terminate and report that the current (partial) path violates the edge constraint for (i, j). Otherwise, when the r-t path is completed, we subtract the minimum flow value along the path from each of its arcs, and repeat the process until all arcs out of the root have flow value zero. If in the process none of the paths violates an edge constraint, we report that the solution is feasible. Finding one r-t path takes linear time (in the size of the decision diagram, $|D|$). The edge inspection takes $O(n)$ time per event, which makes the total time for identifying a single path $O(n \cdot |D|)$. Since there are at most n paths, the total time is $O(n^2|D|)$. □

Proof of Theorem 4. 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, where n is an odd integer. We define the following fixed variable ordering to compile the decision diagrams. For layers $i = 1, \ldots, \lceil n/2 \rceil$, we associate vertex i if i is odd, and vertex $\lceil n/2 \rceil + i$ if i is even. The remaining layers are defined in the 'reverse order'; layer $i = \lceil n/2 \rceil + 1, \ldots, n$ is associated with vertex $n - i + 2$ if i is odd, and vertex $n - i + \lceil n/2 \rceil$ if i is even.
Observation 1: up to layer $\lceil n/2 \rceil - 1$, vertex i appears in each state of layer $i - 1$ since the vertices associated with these layers are not adjacent in G. Therefore, each of these states has two outgoing arcs, the 0-arc and the 1-arc.
Observation 2: up to layer $\lceil n/2 \rceil - 1$, each 1-arc eliminates one element from the set $\{\lceil n/2 \rceil + 1, \ldots, n\}$. Therefore, the states of each layer, up to layer $\lceil n/2 \rceil - 1$, are distinct. These two observations imply that the exact decision diagram requires at least $O(2^{\lceil n/2 \rceil})$ states (the size of layer $\lceil n/2 \rceil$).

Without loss of generality, we assume that the iterative refinement procedure applies a lexicographic search in each iteration to find an optimal solution to model (F), and refines the decision diagram whenever an edge conflict is detected. In iteration i, the algorithm will consider the conflict associated with edge $(i, i + \lceil n/2 \rceil)$. Each of these conflicts is refined by adding one more node to state $i + \lceil n/2 \rceil + 1$. After $\lceil n/2 \rceil$ iterations, no more conflicts are found. The iterative refinement procedure therefore terminates with a decision diagram of $O(n)$ size. □

				Relaxed DD				Exact DD				
instance	n	m	d	LB	UB	time	size	LB	UB	time	size	R/E
1-FullIns_3	30	100	0.23	4	4 *	0.03	249	4	4 *	0.04	747	0.33
2-FullIns_3	52	201	0.15	5	5 *	0.26	1,206	5	5 *	0.59	12,866	0.09
david	87	406	0.11	11	11 *	0.01	246	11	11 *	1.60	37,029	0.01
DSJC125.9	125	6,961	0.90	44	44 *	9.52	9,433	44	44 *	0.37	9,868	0.96
fpsol2.i.1	496	11,654	0.09	65	65 *	3.19	7,135	65	65 *	0.09	8,295	0.86
fpsol2.i.2	451	8,691	0.09	30	30 *	0.06	957	30	30 *	0.14	10,167	0.09
fpsol2.i.3	425	8,688	0.10	30	30 *	0.06	970	30	30 *	0.19	10,257	0.09
huck	74	301	0.11	11	11 *	0.09	786	11	11 *	0.02	1,077	0.73
inithx.i.1	864	18,707	0.05	54	54 *	0.83	3,993	54	54 *	0.21	15,804	0.25
inithx.i.2	645	13,979	0.07	31	31 *	34.57	21,787	31	31 *	1.05	24,588	0.89
inithx.i.3	621	13,969	0.07	31	31 *	51.19	22,777	31	31 *	0.79	24,550	0.93
jean	80	254	0.08	10	10 *	0.01	291	10	10 *	0.32	5,251	0.06
miles1000	128	3,216	0.40	42	42 *	0.58	4,104	42	42 *	0.11	8,031	0.51
miles1500	128	5,198	0.64	73	73 *	0.31	2,697	73	73 *	0.04	4,007	0.67
miles250	128	387	0.05	8	8 *	0.01	294	8	8 *	0.17	2,812	0.10
miles500	128	1,170	0.14	20	20 *	0.02	361	20	20 *	0.39	15,272	0.02
miles750	128	2,113	0.26	31	31 *	0.04	710	31	31 *	0.29	13,153	0.05
mulsol.i.1	197	3,925	0.20	49	49 *	0.12	1,107	49	49 *	0.02	2,487	0.45
mulsol.i.2	188	3,885	0.22	31	31 *	0.07	793	31	31 *	0.03	2,611	0.30
mulsol.i.3	184	3,916	0.23	31	31 *	0.06	789	31	31 *	0.03	2,621	0.30
mulsol.i.4	185	3,946	0.23	31	31 *	0.06	806	31	31 *	0.03	2,636	0.31
mulsol.i.5	186	3,973	0.23	31	31 *	0.06	807	31	31 *	0.03	2,649	0.30
myciel3	11	20	0.36	4	4 *	0.02	59	4	4 *	0.01	62	0.95
myciel4	23	71	0.28	5	5 *	2.31	454	5	5 *	0.35	459	0.99
queen5_5	25	160	0.53	5	5 *	0.01	194	5	5 *	0.01	560	0.35
queen6_6	36	290	0.46	7	7 *	1.03	1,927	7	7 *	0.11	2,686	0.72
queen7_7	49	476	0.40	7	7 *	0.89	3,267	7	7 *	0.23	13,838	0.24
queen8_8	64	728	0.36	9	9 *	164.56	30,606	9	9 *	38.82	81,574	0.38
r125.1	125	209	0.03	5	5 *	0.01	339	5	5 *	0.02	920	0.37
r125.1c	125	7,501	0.97	46	46 *	0.65	3,569	46	46 *	0.04	4,007	0.89
r125.5	125	3,838	0.50	36	36 *	57.31	18,924	36	36 *	0.49	23,242	0.81
r250.1c	250	30,227	0.97	64	64 *	11.54	18,059	64	64 *	0.15	20,322	0.89
zeroin.i.1	211	4,100	0.19	49	49 *	0.08	1,024	49	49 *	0.03	2,769	0.37
zeroin.i.2	211	3,541	0.16	30	30 *	0.05	774	30	30 *	0.05	3,470	0.22
zeroin.i.3	206	3,540	0.17	30	30 *	0.05	769	30	30 *	0.05	3,457	0.22
anna	138	493	0.05	11	11 *	0.01	356	0	11	2.80	$\geq$ 100k	$\leq$ 0.00
DSJR500.1	500	3,555	0.03	12	12 *	0.01	626	0	12	1.43	$\geq$ 100k	$\leq$ 0.01
games120	120	638	0.09	9	9 *	40.77	43,074	0	9	1.90	$\geq$ 100k	$\leq$ 0.39
le450_25a	450	8,260	0.08	25	25 *	0.05	943	0	25	1.90	$\geq$ 100k	$\leq$ 0.01
le450_25b	450	8,263	0.08	25	25 *	0.04	827	0	25	1.51	$\geq$ 100k	$\leq$ 0.01
le450_5d	450	9,757	0.10	5	5 *	0.02	700	0	5	2.45	$\geq$ 100k	$\leq$ 0.01
r1000.1	1,000	14,378	0.03	20	20 *	0.06	1,234	0	20	1.56	$\geq$ 100k	$\leq$ 0.01
r250.1	250	867	0.03	8	8 *	3.03	11,614	0	8	1.40	$\geq$ 100k	$\leq$ 0.11
school1	385	19,095	0.26	14	14 *	23.87	22,322	0	15	1.01	$\geq$ 100k	$\leq$ 0.21
school1_nsh	352	14,612	0.24	14	14 *	23.80	23,430	0	16	1.23	$\geq$ 100k	$\leq$ 0.21
2-Insertions_3	37	72	0.11	3	4	1,800	2,713	4	4 *	362.45	2,963	0.92
DSJC250.9	250	27,897	0.90	71	93	1,800	79,637	72	72 *	1,749.80	80,681	0.99

Table 3. A comparison of relaxed and exact decision diagrams on instances that were optimally solved by either method (indicated by *). For each instance we list the number of nodes (n) and edges (m), and edge density (d). 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 1,800s, and the maximum size was set to 100,000 nodes.