Decision Diagrams in Graph Coloring

Timo Brand

Geboren am 26. November 1999 in München, Deutschland

30. Juli 2021

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Stephan Held

Zweitgutachter: Prof. Dr. Stefan Hougardy

Forschungsinstitut für Diskrete Mathematik

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Introduction

Abstract

In this thesis we will introduce the foundations of graph coloring and give an overview of previous effort on algorithmic approaches to this problem. Based on van Hoeve's work [Hoe21], detailing a new approach to the graph coloring problem, we present decision diagrams and how they can be used in this context. These decision diagrams can compactly represents all possible color classes, with some possibly containing a conflict. Such conflicts are identified by solving a constrained minimum network flow problem and separated to yield a decision diagram with a better approximation to our solution set, i.e. color classes with less or even no conflicts, resulting in a valid minimal coloring. We go over all the algorithms of this concept and discuss our implementation details, as well as evaluating our code on the DIMACS instances and against another graph coloring program.

Our contribution given in this thesis will consist of

- giving a survey on the graph coloring problem as well as heuristic and exact methods,
- repeating van Hoeve's proofs of the network flow model for a modified formulation,
- characterizing the relaxation of the network flow model on exact decision diagrams,
- adapting a method to obtain safe lower bounds from the linear programs we solve,
- extending van Hoeve's algorithm by further ideas,
- computing an improved lower bound for instance `DSJR500.9` and
- reporting the chromatic number of the previously open instance `r1000.1c`.

Acknowledgments

First I want to thank Prof. Stephan Held for the, even if sadly only digital, informative and helpful counseling, for making time for all my questions and assigning me such interesting and recent ideas to write about. Further thanks is due to Prof. Stefan Hougardy for taking on the second correction of this thesis. I also thank the institute for discrete mathematics for allowing me access to their computers to complete all my Benchmarks, especially Michael Schreiner for setting everything up for me.

For proofreading, helping me with my implementation and for listening to the presentations about my thesis i want to thank Carlotta, Daniela, Emil, Martin, Moritz and Thomas, and thank them also for making my studies an enjoyable time. A very special thank you, however, goes to my family for the years of never ending support.

Auszug

In dieser Arbeit geben wir eine Einführung in die Graphenfärbung und stellen verschiedene algorithmische Ansätze vor die versuchen dieses Problem zu lösen. Basierend auf der Arbeit von van Hoeve [Hoe21], in der ein neuartiger Ansatz vorgestellt wird, definieren wir Entscheidungsdiagramme und zeigen wie diese in dem Zusammenhang der Graphenfärbung benutzt werden. Mithilfe von Entscheidungsdiagrammen können wir kompakt alle Farbklassen darstellen, manche dieser Farbklassen enthalten dabei möglicherweise einen Konflikt. Solche Konflikte werden mit einem minimalem Flussproblem identifiziert und aufgelöst, um ein Entscheidungsdiagramm mit weniger bzw. keinen Konflikten zu erhalte, dessen Farbklassen dann einer minimalen Färbung entsprechen. Wir präsentieren die entsprechenden Algorithmen zu diesem Konzept und vergleichen die Performance unser Implementierung auf den DIMACS Instanzen sowie im direkten Vergleich mit einem anderen Programm zu Graphenfärbung.

Der Eigenbeitrag dieser Arbeit besteht aus folgenden Leistungen:

- Übersicht über die Graphenfärbung, sowie heuristischen und exakten Methoden,
- wiederholen von Beweisen von van Hoeve für ein modifiziertes Flussproblem,
- charakterisierung der linearen relaxierung des Flussproblems auf exakten Entscheidungsdiagrammen,
- vorstellen einer Methode um sichere Schranken aus den von uns benutzten linearen Programmen zu erhalten,
- erweitern von van Hoeve's Ansatz mit weiteren Ideen,
- verbessern der unteren Schranke für die Instanz `DSJR500.9` und
- berechnen der chromatischen Zahl auf der bisher ungelösten Instanz `r1000.1c`.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Graph Coloring

In this chapter we discuss the basics of graphs, colorings and previous algorithms and heuristics used to compute the chromatic number.

1.1 Definitions

Let $G = (V, E)$ be a simple undirected graph on the set of vertices V with the set of edges $E \subseteq \{\{v, w\} \mid v, w \in V \text{ and } v \neq w\}$. Further let $|V| = n$ and $|E| = m$, we label the vertices either as $V = \{v_1, \dots, v_n\}$ or consider them as the set of integers $V = [n] \coloneqq \{1, \dots, n\}$. With that we define $V_j = \{1, \dots, j\} \subseteq V$ and the set of outgoing edges of a vertex as $\delta(v) \coloneqq \{e \in E \mid e = \{v, w\} \text{ for some } w \in V\}$. The set of neighbors of v is denoted as $N(v) \coloneqq \{w \in V \mid \{v, w\} \in E\}$ while $N[v] \coloneqq N(v) \cup \{v\}$ is the closed neighborhood of v. As we will work only with one graph at a time, we omit mentioning the underlying graph in these definitions.

Before moving on to the problem, we also define *independent sets*, also called *stable sets*. These are subsets $U \subseteq V$ such that no two vertices in it are adjacent: for all $v, w \in U$ we have $\{v, w\} \notin E$. The opposite of stable sets are *cliques*, in a clique all vertices are pairwise adjacent: for all $v, w \in U$ we have $\{v, w\} \in E$. The largest clique in a graph G is called the *clique number* and denoted by $\omega(G)$.

Graph Coloring In graph coloring, we want to find an assignment of colors to each vertex such that no two neighboring vertices have the same color, or more formal:

A coloring of G is a map $f : V \to \mathbb{N}$ with $f(v) \neq f(w)$ for all $e = \{v, w\} \in E$.

Here we also say that a vertex v is assigned the $f(v)$-th color and if f maps only to k different values, we call it a *k-coloring* and refer to the subset of vertices with the same color as a *color class*. Finding some coloring is simple, assigning each vertex a different one results in a valid coloring. The more interesting problem is the following:

Given G, what is the smallest number k such that a k-coloring exists?

The resulting k is called the *chromatic number* of the graph and denoted by $\chi(G)$. We also say that G is *k-colorable*. Finding this k on the other hand is very far from being easy. In fact,

Theorem 1.1. *It is NP-complete to compute the chromatic number for a general graph.*

Karp showed this in his seminal paper on NP-complete problems in 1972, reducing the problem of finding the chromatic number to the satisfiability problem [Kar72]. Worse, for all $\epsilon > 0$, even approximating $\chi(G)$ with an approximation ratio of $n^{1-\epsilon}$ is NP-hard [Zuc06]. For this thesis, we work with the assumption that $P \neq NP$. This is mostly used for omitting "unless $P = NP$" when saying there does not exist a polynomial algorithm for a problem. With "This problem is hard" we mean to say that there is no polynomial algorithm for it (under the previous assumption).

Set Covering Another problem that will be useful to define for us is the following:

Given a set of elements $E = \{e_1, \dots, e_n\}$ called the universe and a collection S of subsets whose union is E, what is the smallest sub-collection $\hat{S} \subseteq S$ whose union equals E.

As the name *set covering* suggests, we are looking for the fewest sets that already suffice to cover our universe E. This problem can easily be formulated as an integer program with binary variables x_s indicating whether the set $s \in S$ is part of the minimal covering:

$$\begin{aligned} &\min \sum_{s \in S} x_s \\ &\text{s.t.} \sum_{s \in S | e \in s} x_s \geq 1 \quad \forall e \in E \\ &x_s \in \{0, 1\} \quad \forall s \in S \end{aligned} \tag{SC}$$

We will see an exact graph coloring method based on this integer programming formulation and use it for some important reductions in this thesis. Set covering was also shown to be NP-complete by Karp [Kar72].

1.2 Application

While the graph coloring problem is interesting already because of its NP-completeness, there are some useful real world applications too. One of these is in *scheduling*, where a number of jobs has to be assigned to a time slot. Jobs may be scheduled in any order but pairs of jobs may be in conflict in the sense that it is not allowed to assign them to the same time slot, possibly because they share a resource or have some other constraint.

An example can be found in making timetables and scheduling classes [Ly19; Owl]. Assume we are given classes that have to take place and conditions which classes can take place at the same time. Two classes might not be assigned to the same time slot because they take place in the same classroom or have a student in common. Then we can ask for the minimal number of time slots required to accommodate all classes. We model this as a graph coloring problem with a graph G that contains a vertex for each class taking place and model the constraints of classes not possibly taking place at the same time by edges in between those classes. This means they do not get assigned the same color/time slot. Our previous question for the numbers of required time slots is then answered by the chromatic number of the constructed graph (we also get an assignment to the time slots). This can be used on as large a scale as for example air traffic flow management [BB04].

Another use is found in register allocation: during the compilation of a program, variables will be assigned to a register on the processor to efficiently access them at runtime. Two variables needed at the same time can not be assigned to the same register, we proceed with a similar construction as before and ask for the minimum number of registers needed [Owl]. More applications include finding round-robin sport schedules where each of the n teams has to play every other team m times (typically $m = 1$ or $m = 2$), covering an art gallery space with the least amount of security cameras or frequency assignment of radio stations [Ahm12].

1.3 Survey Of Previous Effort

A tremendous amount of work has been done to find and prove theorems about the chromatic number as well as proposing efficient algorithms to compute it. For the latter

there are basically two ways: use a heuristic that will quickly find a valid coloring but may not be minimal or using an exact algorithm at the cost that it will be provably inefficient (remember: unless $P = NP$).

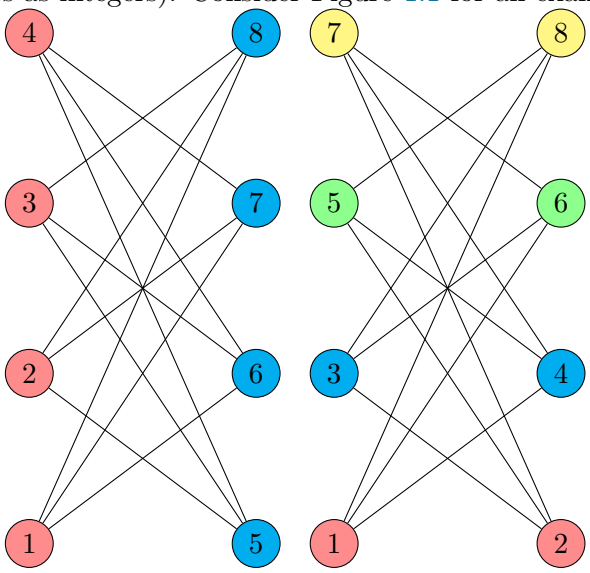
1.3.1 Heuristics

Heuristic algorithms greedily choose the next vertex and which color to give it according to some criteria. For certain classes of graphs, these heuristics can be enough to find the chromatic number but since their runtime is polynomial in $|V|$ and $|E|$ they can not be optimal on all classes of graphs. One such a simple and quite naive heuristic is the following:

Algorithm 1.1: Sequential greedy algorithm

Input: A graph G and some vertex ordering $v_1, \ldots, v_n$
Output: A coloring of G
1 **for** $i = 1, \ldots, n$ **do**
2 assign v_i the smallest color that is not used by any of the vertices in $N(v_i)$

The runtime of Algorithm 1.1 is $\mathcal{O}(n^2)$ because we iterate over the n vertices and for each have to check at most n neighbors. The solution quality of this algorithm is very susceptible to the used ordering, which normally is the lexicographic one (remember that we labeled our vertices as integers). Consider Figure 1.1 for an example.



(a) First ordering (b) Second ordering

Figure 1.1: Performance of the sequential greedy algorithm on a crown graph with two different vertex labels.

The *crown graphs* are bipartite graphs $V = \{u_1, \ldots, u_{n/2}\} \cup \{v_1, \ldots, v_{n/2}\}$ with edges from u_i to v_j if $i \neq j$. As the crown graphs are bipartite, they are 2-colorable and, if in the vertex ordering passed to the greedy all the vertices u_i come before the vertices v_i, it finds such an optimal coloring. On the other hand, if it alternates between u_i and then v_i, the next u_{i+1} will be assigned a new color, thus only finding a solution that uses $n/2$ colors. The example hints at the following proposition:

Proposition 1.2. *For any graph G there exists a good ordering such that the greedy algorithm finds an optimal coloring. Finding such a vertex ordering is an NP-complete problem in itself.*

Proof. Take some minimal coloring. Order the vertices according to their assigned colors i.e. first come the vertices of the first color, then of the second color and so on. Now the greedy goes through the vertices in that order and assigns each vertex the lowest color possible. By the coloring we found earlier, we know each vertex can be assigned at least its old color but it might be possible to select a lower color, which the greedy does. In total, the greedy algorithm finds a (possibly different) coloring where for each vertex the new color is smaller or equal to the old color. A new color is not used, so the found coloring is also an optimal one.

This immediately implies the second part of the proposition: finding such a good ordering for the greedy lets us then compute the chromatic number in polynomial time. But generally computing the chromatic number is NP-complete and thus the first step, finding the ordering of the vertices, must already be NP-complete. □

One can also ask about *bad orderings*, i.e. orderings such that the sequential greedy algorithm produces the worst coloring. The number of colors used is called the *Grundy number* [CS79]. It is interesting to observe that this problem is no easier than finding a good ordering. Determining for a given graph G and parameter k whether there exists an ordering causing the greedy to use k or more colors is NP-complete [Zak06]. In particular, it is difficult to find a bad ordering. The crown graph with n vertices for example has chromatic number 2 and a Grundy number of $n/2$.

A more sophisticated algorithm is the *Dsatur* heuristic formulated by Brélaz [Bré79]. To describe it, we define a *partial coloring* as a map $f' : U \to \mathbb{N}$ on a subset $U \subseteq V$ such that no two neighboring vertices in U are assigned the same color. For a graph G with partial coloring f', the *saturation degree* of a vertex is the number of different colors to which it is adjacent to, ignoring uncolored vertices. The algorithm then looks as follows:

Algorithm 1.2: Dsatur Algorithm

Input: A graph G
Output: A coloring of G

1 Arrange the vertices by decreasing order of degrees
2 Color a vertex with maximal degree with color 1
3 **while** *there exist uncolored vertices* **do**
4 Choose an uncolored vertex with maximal saturation degree. If there are multiple, break the tie by choosing a vertex with maximal degree in the uncolored subgraph
5 Color the vertex with the smallest color that is not used by any of its neighbors

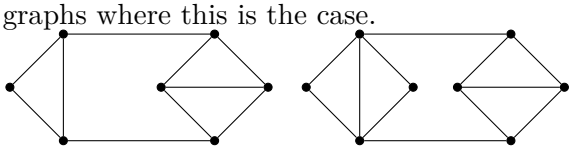
Notice that the algorithm does not depend on a given ordering of V, the next vertex is dynamically chosen according to their saturation degree in the partial coloring. Algorithm 1.2 can be efficiently implemented to run in time $\mathcal{O}(m \log n)$ [Tur88]. One improvement compared to Algorithm 1.1 becomes clear with the next result:

Proposition 1.3. *Algorithm 1.2 is exact for bipartite graphs.*

Proof. Let G be connected and bipartite on at least 3 vertices. If G were disconnected we could look at each connected component separately. We prove by contradiction: assume there exists a vertex v with saturation degree 2, i.e. it has two neighbors u_1 and u_2 with different colors. Starting from u_1 respectively u_2 we can build two paths which will have a common vertex w because G is connected (and finite). We have found a cycle containing v, u_1, u_2, w and possibly more nodes. Since the graph is bipartite, the cycle is of even length and the two neighbors u_1, u_2 of v must have the same color, which is a contradiction

to our assumption. This also yields an efficient method to determine whether a graph is bipartite. □

This is a positive result and better than our first sequential greedy approach but we know Dsatur can not find optimal solutions on all graphs. Even the hope that it will be optimal on small instances (say $n < 10$) is misguided: R. Janczewski et al. [Jan+01] found a graph with only 7 vertices that were not always colored optimally, that is, with **some** applications of Dsatur the coloring used a color more than necessary. There is also a graph with 8 vertices on which **all** applications of Dsatur (i.e. no matter what choice is made to break the ties further) do not find an optimal coloring. R. Janczewski et al. showed that these are the smallest graphs where this is the case.



(a) (b)

Figure 1.2: Depending on the random choice in the tie breaks, Dsatur does or does not solve (a) optimally, for (b) it never does.

A slight variant of this is the *Max-Connected-Degree*, heuristic which only differs by choosing the next vertex not according to their saturation degree, but by the number of colored neighbors, not necessarily colored differently. Note that Dsatur and Max-Connected-Degree produce a vertex ordering by remembering when each vertex is colored. These vertex orderings will be important later on.

1.3.2 Exact Methods

We have seen simple heuristic methods that were very fast but often do not give an optimal solution. With exact methods, we have the opposite situation. These algorithms are able to compute the chromatic number of any graph, but are in practice often too slow for even moderately sized instances of a few hundred vertices. Despite the known difficulty of the problem, a considerable amount of effort has been put into finding algorithms that, while they can not be polynomial, are still fast enough to be used in cases where the chromatic number is needed.

The brute force approach to test for k-colorability is to consider each of the k^n possible vertex assignments and check for each if it is legal. This alone is impractical, but to get the chromatic number one has to repeat this for $k = 1, \ldots, n-1$ until a single legal coloring is found.

When Brélaz introduced his Dsatur heuristic, which we have seen in the previous section, he also put forward an exact algorithm called *DSATUR* [Bré79]. It works by dividing the graph coloring instance into a collection of subproblems, each consisting of a different partial coloration of the graph. Throughout the algorithm, an upper bound UB for the chromatic number is stored. If a subproblem uses $k \geq UB$ colors, it can be discarded since it can not yield a better upper bound or the chromatic number. But if in a subproblem every node is colored and $k < UB$ colors are used, a better coloring has been found and we can set UB to k. We begin with the uncolored graph as the first subproblem. For each subproblem, if it is not completely colored and so far uses $k < UB$ colors, we create new subproblems. Some uncolored vertex i is chosen for branching and for each of the k colors used that are feasible for i, meaning there are no adjacent vertices of i with that color, a new subproblem is created by assigning i that feasible color. Additionally, a subproblem is created where i receives color $k+1$ and the initial subproblem with i uncolored is

marked as done. The algorithm terminates when no subproblems are left, then UB is the chromatic number [MT96]. The choice of node i that is branched on can have a large effect on the number of subproblems generated. Brélaz suggested choosing i according to the Dsatur heuristic, which is, choose next the vertex with the highest saturation degree. This will reduce the number of subproblems created as there are less feasible colors for i.

Another, and one for this thesis quite important approach, is based on the following observation: in a color class, no two vertices are incident to each other i.e. it is an independent set of G. Conversely, we can assign to all vertices of an independent set the same color without conflict. This enables us to formulate the graph coloring problem as an integer program: let I be the set of all independent sets in G and z_i be a binary variable for all $i \in I$. We use z_i to represent whether i is used as a color class and the binary coefficients a_{ij} will represent whether vertex $j \in V$ lies in the subset i, that is $a_{ij} = 1$ if $j \in i$ and $a_{ij} = 0$ else. The integer program is then:

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

This is a vertex cover formulation of the problem, a special case of (SC) in which we chose $E = V$ as universe and all independent sets as the collection I of subsets. It finds the chromatic number by assuring that each vertex of the graph is contained in at least one independent set i. These independent sets are by the correspondence described above also color classes, thus minimizing the number used over all stable sets yields the chromatic number. If a vertex happens to be assigned to more than one stable set, we simply remove it from all but one to obtain disjoint stable sets. This formulation was first discarded for use in the graph coloring problem since a graph can have exponentially many stable sets which would mean an integer program with that many variables.

Nevertheless, Merothra and Trick [MT96] used it as the basis for their *column generation* approach to graph coloring. Instead of considering all independent sets at once, they begin with $\bar{I} \subseteq I$ and solve the linear relaxation of (VCIP) using only independent sets from $\bar{I}$. The relaxation is achieved by simply substituting the integrality constraints of the z_i with $0 \leq z_i \leq 1 \; \forall i \in \bar{I}$. Beginning with this linear program of a manageable size, we obtain a feasible solution to the linear relaxation of (VCIP) and a dual value π_j for each constraint. With this, we determine if it would lead to a better solution to expand $\bar{I}$ by adding more independent sets. This is done by using the dual variables π_j in the following *weighted independent set* problem:

$$\begin{aligned} \alpha_\pi(G) \coloneqq \max & \sum_{j \in V} \pi_j y_j \\ \text{s.t.} & \; y_j + y_k \leq 1 \quad \forall \{j,k\} \in E \\ & \; y_j \in \{0,1\} \quad \forall j \in V \end{aligned} \tag{MWIS}$$

which is also called the *weighted stability number* [HCS12]. If the optimal value of (MWIS) is greater than 1, the y_j with value 1 correspond to an independent set that should be added to $\bar{I}$. If $\alpha_\pi(G) \leq 1$, there exists no more improving independent sets and solving the linear relaxation of (VCIP) over $\bar{I}$ is the same as solving it over $\bar{I}$. This is the main idea of Mehrothra and Trick's column generation algorithm, but there are still some difficulties

one has to consider. Namely (MWIS) is itself a difficult problem and the resulting solution to the relaxation of (VCIP) may have some z_i that are not integers. Mehrothra and Trick present efficient ways to compute $\alpha_\pi(G)$ and branching rules to ensure integrality [MT96].

The algorithm introduced by van Hoeve [Hoe20; Hoe21] and discussed in this thesis will also be an exact method. Additionally, we get improving lower and upper bounds throughout the algorithm, thus one may terminate the program early because of time constraints and still obtain strong bounds. The approach will be somewhat opposite to that we have just seen. In column generation, one starts with only a few independent sets and adds more until the used sets are enough to determine an optimal solution. Our approach uses decision diagrams to compactly represent all subsets of the graph and iteratively removes sets that are not stable sets. This is done until our solution contains no more conflicts and is optimal, i.e. the sets that our solution uses are actually independent sets. Before we introduce decision diagrams, we briefly look at the fractional chromatic number of a graph.

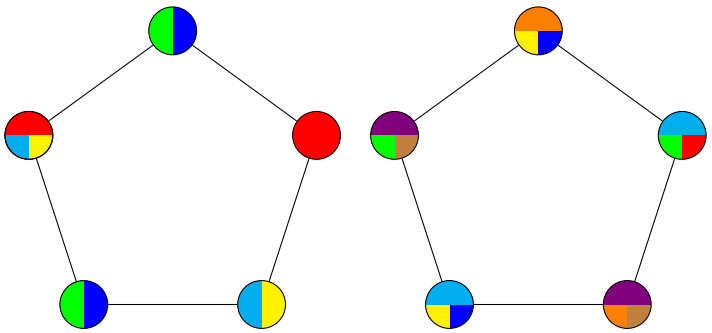
1.4 Fractional Chromatic Number

As a generalization of coloring a graph, instead of assigning each vertex only a single color and adjacent vertices being colored differently, one might assign each vertex a set of colors. The requirement for adjacent vertices stays unchanged, they must have no common color. The fractional chromatic number will only be important in Section 2.5 of this thesis, therefore reading of this section may be deferred until then. In precise terms we generalize:

Definitions A *fractional coloring* is a map $f : V \to \mathcal{P}(\mathbb{N})$, assigning each vertex a set of colors such that adjacent vertices receive disjoint sets of colors, that is, $f(v) \cap f(w) = \emptyset$ for all $\{v, w\} \in E$. $\mathcal{P}(X)$ is the power set of X and we write $\binom{X}{k}$ for the set of all subsets of X with k elements. A *b-fold coloring* is a map $f : V \to \binom{\mathbb{N}}{b}$ of a Graph G assigns each vertex a set of b colors, again with the condition that adjacent vertices receive disjoint sets of colors. We say that G is *$a:b$-colorable* if it has a b-fold coloring where the colors are drawn from a palette of size a. In other words, a b-fold coloring $f : V \to \binom{[a]}{b}$ exists. The *b-fold chromatic number* $\chi_b(G)$ of a graph G is the smallest a such that an $a:b$-coloring exists, so $\chi_1(G) = \chi(G)$. Finally, the *fractional chromatic number* of G is defined as:

$$\chi_f(G) := \lim_{b \to \infty} \frac{\chi_b(G)}{b} = \inf_b \frac{\chi_b(G)}{b} \tag{1.1}$$

The limit exists because the b-fold chromatic number is subadditive, and there is a b such that $\chi_f(G) = \frac{\chi_b(G)}{b}$ [SU11]. See Figure 1.3 for some examples to these definitions. From Figure 1.3(b) we can conclude that the 3-fold chromatic number of that graph is 8 by computing $2.5 = \chi_f(G) \leq \frac{\chi_3(G)}{3}$. If $\chi_3(G)$ were equal to 7 we would have $\frac{\chi_3(G)}{3} = \frac{7}{3} < 2.5\dot{4}$. While we assumed here that we know $\chi_f(G) = 2.5$, this valid 3-fold coloring already brings attention to the fact that the chromatic and fractional chromatic number must not always be equal.



(a) A fractional coloring (b) A 3-fold coloring

Figure 1.3: Two possible fractional colorings of a simple graph with 5 vertices. From (b) we know that G is $8:3$-colorable.

This is a somewhat cumbersome definition and does not give any tools to actually compute $\chi_f(G)$. Instead, we look at

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

the linear relaxation of (VCIP) [HCS12]. As all coefficients are integral, there exists a rational optimal solution to this (VCLP) which can be transform into a fractional coloring, and a fractional coloring can be transformed into a solution of (VCLP) [Dvo18].

With this formulation it is immediately clear that $\chi_f(G) \leq \lceil \chi_f(G) \rceil \leq \chi(G)$, and the fractional chromatic number gives a lower bound for the chromatic number. While we can solve linear programs in polynomial time, (VCLP) still has exponentially many variables. At first, glance computing the fractional chromatic number seems like a hard problem, and this turns out to be correct. Similarly to Theorem 1.1 we have:

Theorem 1.4. *It is NP-hard to compute the fractional chromatic number for a general graph. Again, for all $\epsilon > 0$ even approximating $\chi_f(G)$ with a ratio of $n^{1-\epsilon}$ is NP-hard [Zak06].*

Now that we know it is also a hard problem, one might hope the fractional chromatic number is at least a good lower bound and approximation for the chromatic number, i.e. the integrality gap for (VCIP) and its relaxation (VCLP) is small.

An interesting family of graphs to consider for this question are the *Mycielskians* or *Mycielski graphs*, based on the Mycielski transformation $\mu(G)$ that produces a graph such that $\chi(\mu(G)) = \chi(G) + 1$ [Myc55]. For the family of Mycielski graphs, we begin with the graph M_2 on two vertices with a single edge and iteratively apply the transformation to obtain $M_i = \mu(M_{i-1})$. These were of interest to Mycielski because his transformation left the clique number of a graph unchanged, while the number of needed colors strictly increased. Combining $\omega(M_i) = \omega(M_{i-1})$ and $\chi(M_i) = \chi(M_{i-1}) + 1$ when beginning with $\omega(M_2) = 2$ and $\chi(M_2) = 2$ showed that there exists a family of graphs such that the clique number and the chromatic number grow arbitrarily apart; the first stays constant while the latter continuously grows.

To us, the Mycielskians are of additional interest because of their fractional chromatic

number. Larsen et al. [LPU95] showed that the fractional chromatic number of $\mu(G)$ can be derived from that of G:

$$\chi_f(\mu(G)) = \chi_f(G) + \frac{1}{\chi_f(G)} \tag{1.2}$$

For the M_i we begin with $\chi_f(M_2) = 2$. The first transformation $M_3 = \mu(M_2)$ is the cycle on five vertices of which we have seen a fractional and a 3-fold coloring in Figure 1.3. Equation (1.2) now justifies our previous assumption of $\chi_f(M_3) = 2.5$ by computing $\chi_f(M_3) = \chi_f(M_2) + \chi_f(M_2)^{-1} = 2 + \frac{1}{2}$. Further, the sequence $a_n = a_{n-1} + \frac{1}{a_{n-1}}$ grows asymptotic to $\sqrt{2n}$ while the chromatic number grows linearly. This way the Mycielski graphs produce a nice way to show that $\chi(G) - \chi_f(G)$ can grow infinitely large [LPU95; SU11]. The same holds for $\chi_f(G) - \omega(G)$. More generally:

Theorem 1.5. *[Lov75] The integrality gap between $\chi(G)$ and $\chi_f(G)$ is $\mathcal{O}(\log n)$.*

This is not a small gap or a good approximation and thus a negative answer to our question about the quality of $\chi_f(G)$ as a lower bound to the chromatic number.

Chapter 2

Graph Coloring With Decision Diagrams

2.1 Decision Diagrams

We mentioned before that this approach to graph coloring uses *decision diagrams*, which we will formally introduce in this chapter. They were extensively used in the context of representing boolean functions, though we will use a slightly modified concept. There are two methods of building decision diagrams for discrete optimization problems [Tja18]:

- *Exact compilation techniques* construct a decision diagram from scratch, typically using a dynamic programming formulation. There are two common compilation approaches, top-down and depth-first. We only discuss the former.
- *Refinement techniques* begin and improve upon an existing decision diagram to better approximate the set of feasible solutions or reduce its size, typically by removing infeasible solutions. There are many techniques but we focus on constraint/conflict separation.

Following van Hoeves Work [Hoe20; Hoe21], we explore and apply both techniques to our graph coloring problem.

2.1.1 Definitions

We focus on decision diagrams representing the set of feasible solutions of a discrete optimization problem, either exactly or approximately depending on the technique used. We keep with the notation and description used by van Hoeve [Hoe21]. For an optimization problem P defined on the ordered set of variables $X = \{x_1, x_2, \dots, x_n\}$ we denote its feasible set of solutions by $\text{Sol}(P)$. We assume the x_i are binary and call them *decision variables*.

To P we associate a layered directed acyclic graph $D = (N, A)$ with nodes N and arcs A which will be what we call a *decision diagram* for P. D has $n+1$ layers $L_1, \dots, L_{n+1}$ which contain the nodes and arcs go from nodes in L_i only to nodes in L_{i+1}. Note that $N = \bigcup_{i=1}^{n+1} L_i$. The first layer L_1 consists of a single *root node* r, likewise the last layer L_{n+1} consists only of a single *terminal node* t. A layer L_j is a collection of nodes of D which we associate to the decision variable $x_j \in X$, for $j \leq n$. For a node $u \in L_j$ we denote its layer by $L(u)$ and associate to it some *state information* $S(u)$ depending on the underlying optimization problem. For an arc a from node u in layer j to node v in layer $j+1$ we have an associated label $l(u, v) \coloneqq l(\{u, v\})$ which is either 0 or 1 and we refer to them as 0-arcs respectively 1-arcs. For each node, except the terminal node, we have exactly one outgoing 0-arc and at most 1-arc. We require that each node and each arc must belong to a path from r to t. If we specify an r-t path $p = (a_1, a_2, \dots, a_n)$ by its arcs, we can define a variable assignment of X by letting $x_j = l(a_j)$ for $j = 1, \dots, n$. As D itself is not an optimization problem, we slightly abuse notation to denote with $\text{Sol}(D)$ the collection of variable assignments obtained by all r-t paths. Having defined the set

of solutions, we introduce two further terms: a decision diagram for problem P is called *exact* if $\text{Sol}(P) = \text{Sol}(D)$ and *relaxed* if $\text{Sol}(P) \subseteq \text{Sol}(D)$.

For our purposes, P will be the optimization problem of finding a maximum independent set on a graph G. Note that we speak of vertices and edges in G and of nodes and arcs in D. We have for each vertex $j \in V$ a decision variable x_j which denotes whether the vertex is part of the independent set ($x_j = 1$) or not ($x_j = 0$). Having fixed P, we specify the state information we keep at each node in D. For our problem of independent sets, we store a set of "eligible vertices", meaning the set of vertices that we can still add to the independent sets represented by all the paths into that node (a partial path in D also gives a partial assignment of decision variables). We will expand on this to describe the compilation of exact decision diagrams for our specific optimization problem P.

2.1.2 Exact compilation

To build an exact decision diagram to our independent set problem we apply top-down compilation. For that we construct the layers $L_1, \ldots, L_{n+1}$ in order, adding one node after the other to the current layer. The state information $S(u)$ we keep for each node is a subset of the set of vertices V and is recursively derived from the state information of the nodes parents, i.e. the nodes whose arcs end in u. We begin with $S(r) = V$ and for a node $u \in L_j$ with known state information, we distinguish two cases. In the simple case, when $j \notin S(u)$, we insert a node v with $S(u) = S(v)$ in layer $j + 1$ and add a 0-arc from u to v. If $j \in S(u)$, we insert two nodes into layer $j + 1$ and define both a 0-arc and a 1-arc out of u. The new node(s) will have the following state information:

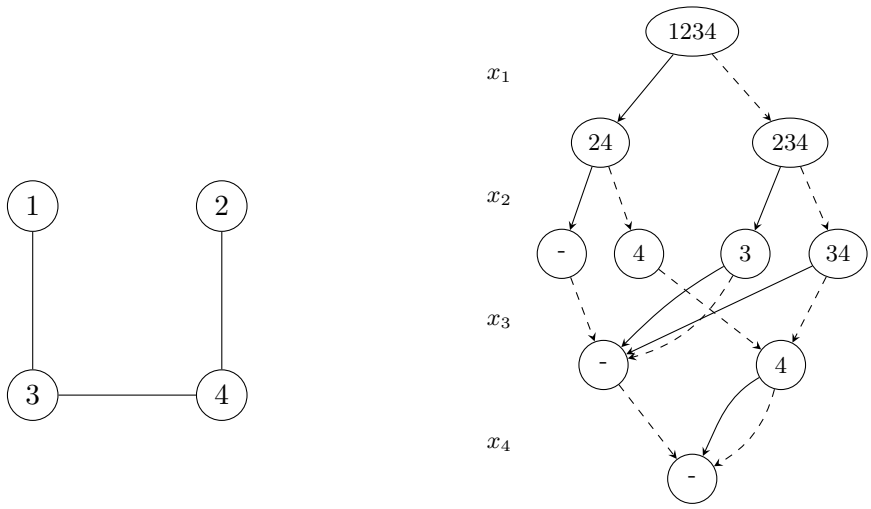
$$S(v) = \begin{cases} S(u) \backslash N[j] & \text{if } (u, v) \text{ is a 1-arc,} \\ S(u) \backslash \{j\} & \text{if } (u, v) \text{ is a 0-arc.} \end{cases} \tag{2.1}$$

Starting at the root node, we create all nodes in the next layer according to the state transition given in Equation (2.1) and then repeat this for the next layer. Because we keep track of the eligible vertices and only insert 1-arcs if this state information tells us it is possible, no node or arc is inserted that corresponds to an infeasible partial solution. One further nice property specific to our problem of finding maximal independent sets is that *equivalent nodes* are easily identified. We say that two nodes u and v are equivalent if all partial solutions obtained by paths from the root node r to u respectively v have the same set of feasible completions, i.e. continuing paths from u respectively v to the terminal node t give the same set of variable assignments. Bergman et al. [Ber+12a] observed that in the case of the independent set problem, the state information of two nodes is enough to determine the equivalence of nodes:

Proposition 2.1. *Two nodes u and v in layer L_j of a decision diagram D are equivalent if and only if $S(u) = S(v)$.*

A decision diagram in which no two nodes of a layer are equivalent is called *reduced*. As we can merge equivalent nodes without changing $\text{Sol}(D)$, we easily get a reduced exact decision diagram from our top-down compilation method by checking whether there already exists a node with the same state information before inserting a new one. For a given variable ordering of P, there always exists a unique reduced decision diagram [Bry86]; the described exact compilation method yields the unique reduced decision diagram representing all independent sets. But even reduced the decision diagram might be exponentially large to represent all solutions for a given problem.

For some illustrations of the definitions we have given, consider in Figure 2.1 the very elegant example graph and its exact decision diagram found by van Hoeve [Hoe21]. To



(a) Example input graph

(b) Exact decision diagram

Figure 2.1: A graph on 4 vertices and its exact decision diagram. The paths in (b) represent independent sets of the input graph (a). Solid arcs represent 1-arcs, dashed arcs represent 0-arcs and in each node the state information is written.

the graph in Figure 2.1(a) we show the exact decision diagram obtained by the top-down compilation in Figure 2.1(b).

2.1.3 Conflict separation

This method starts with a decision diagram D' that is a relaxation of the exact decision diagram D: $\text{Sol}(P) = \text{Sol}(D) \subseteq \text{Sol}(D')$. Thus D' encodes all the solutions of P, but perhaps also infeasible solutions. The strategy here is to find such (partial) variable assignments that are parts of an infeasible solution and *separate* that conflict in D'. That is, changing the node and arc set of D' to remove the infeasible solution. In this section we describe an algorithm that is given a relaxed decision diagram and a conflict in the form of a partial variable assignment as input and removes that path from the decision diagram. For the problem P of finding a maximal independent set, a conflict will consist of a path where the corresponding variable assignment respectively the set of vertices will contain two adjacent vertices. To separate this conflict, a new node is inserted in each layer along the arc-specified path of the partial variable assignment except in the last layer of the path where we forbid the 1-arc. This removes the path containing the conflict from the decision diagram.

We need to begin with some relaxed decision diagram so we chose the simplest one, which we call the *initial diagram*. In each layer we have only a single node and a node in layer j has a 0-arc and a 1-arc going into the single node in layer $j+1$. In the relaxed diagram we also store state information. The root r has $S(r) = V$ and a node $u \in L_j$ has $S(u) = \{j, \dots, n\}$ for $j = 1, \dots, n$ and $S(t) = \emptyset$ as there are no outgoing arcs of t. Since in each layer of this diagram we can chose the 0-arc or the 1-arc, all 2^n binary tuples and thus all subsets of V are represented. Of course this includes the independent sets of G, therefore it is a valid relaxed decision diagram.

To talk about the algorithm for conflict separation formulated by van Hoeve [Hoe21], we first describe how we represent a decision diagram D in code. We use a two-dimensional vector, denoted D[ ][ ], of nodes where D is a collection of layers, D[j] is the collection of nodes in the j-th layer, i.e. L_j, and D[j][i] is the i-th node in layer j. We start the indexing

of vectors with 1. We have $n+1$ layer thus D has size $n+1$ whereas the size of each layer $\mathrm{D}[i]$ is updated dynamically. $\mathrm{D}[1][1] = r$ is the root node, $\mathrm{D}[n+1][1] = t$ the terminal node and for a node $u = \mathrm{D}[i][j]$ we further store the state information as $\mathrm{D}[j][i].\mathrm{S} = S(u)$ and keep a reference to the endpoints of the 0-arc and 1-arc in layer $j+1$ as $\mathrm{D}[j][i].\mathrm{zeroArc}$ respectively $D[j][i].\mathrm{oneArc}$. If no 1-arc exists, we set $D[j][i].\mathrm{oneArc} = -1$.
What we need to separate a conflict in a decision diagram are

- ▷ the reduced diagram D in which we wish to separate the conflict,
- ▷ an edge conflict $(j,k) \in E$ with $j < k$ and
- ▷ an associated path and labels to the conflict, consisting of the nodes $u_j, \ldots, u_k$ and the labels $l_j, \ldots, l_{k-1}$ of the arcs from u_i to u_{i+1}. The given path and arc labels must not contain an edge conflict (j', k') such that $j \leq j' < k' < k$.

In the above mentioned situation we say that the edge conflict (j,k) is a *minimal conflict*.

In Algorithm 2.1, we look at a conflict on a given path and we redirect that path by inserting nodes and/or changing the endpoint of arcs. For each node on the path (line 1), we create a temporary node w (lines 2-4) and if a node equivalent to that already exists we redirect the path to go through that node (line 5-6). Otherwise, we copy the outgoing arcs (lines 8-10) of the next node on the path ($\mathrm{D}[i+1][u_{i+1}]$) to w and add the temporary node w (lines 11-12). To complete an iteration, we alter the path to go through either the equivalent node we found or the newly inserted one and update the arcs accordingly (line 13-15). The intuitive idea of the algorithm is that we have a path with arc labels $l_j, \ldots, l_{k-2}, 1$, we alter the path by creating new nodes and at the last node of the path we forbid the 1-arc, thus only the $l_j, \ldots, l_{k-2}, 0$ path exists in the new diagram. The algorithm is characterized by Lemma 2.2 and it is shown that it removes exactly the specified conflict from the reduced decision diagram.

Algorithm 2.1:
Separating an edge conflict (j,k) in the reduced diagram D along a path

Input: A reduced decision diagram D represented as two-dimensional vector D[ ][ ], a path via node indices $u_j, \ldots, u_k$ and arc labels $l_j, \ldots, l_{k-1}$ with minimal conflict (j,k) and a list of neighbors $N(i)$ for $i \in V$
Output: reduced decision diagram in which the conflict (j,k) along the speciefied path has been eliminated

```
1  for i = j, j+1, ..., k-1 do
2      create a temporary node w
3      w.S ← D[i][u_i].S \ {i}
4      if l_i = 1 then w.S ← w.S \ N(i)
5      t ← −1
6      if ∃r such that D[i][r].S = w.S then t ← r
7      else
8          if i+1 ∈ w.S then  w.oneArc ← D[i+1][u_{i+1}].oneArc
9          else w.oneArc ← −1
10         w.zeroArc ← D[i+1][u_{i+1}].zeroArc
11         D[i+1].add(w)
12         t ← |D[i+1]
13     if l_i = 1 then D[i+1][u_i].oneArc ← t
14     else D[i+1][u_i].zeroArc ← t
15     u_{i+1} ← t
```

Lemma 2.2. *Application of Algorithm 2.1 with a reduced decision diagram D and a minimal conflict (j,k) along a path on the nodes $u_j, \ldots, u_k$ with labels $l_j, \ldots, l_{k-1}, 1$ as input yields another reduced decision diagram, in which only the arc-specified path $l_j, \ldots, l_{k-1}, 0$ starting at u_j exists, but not $l_j, \ldots, l_{k-1}, 1$.*

Proof. First observe that we need to select vertex j, otherwise we do not have a conflict with vertex k. Therefore $l_j = 1$ and when we start in line 1 with $i = j$ we eliminate $k \in N(j)$ from the state $\mathrm{S}(w)$ of the temporary node w (line 2-4). In further iterations where $i = j + 1, \dots, k - 1$, node w inherits its state from its parent node which by the previous observation never includes k.

The state update in line 4 removes $N(i)$ from the sate of w if $l_i = 1$, which potentially eliminates those 1-arcs from the given path. But since we require that the conflict is minimal, there are no adjacent j', k' on the path with $j \leq j' < k' < k$, so no 1-arc other than the last one will be removed from the path. It follows that in each iteration i, there exists an arc a from the node in the arc label specified path with label $l(a) = l_i$. As equivalent nodes have the same state information and thus the same outgoing arcs, merging of those nodes has no impact on the arc labels. In the last step of the iteration where $i = k - 1$, vertex k does not appear in $\mathrm{S}(w)$ so w as node in layer k has only a 0-arc and no outgoing 1-arc, which eliminates the conflicting assignment. The path $l_j, \dots, l_{k-1}, 1$ no longer exists after the termination of the algorithm but $l_j, \dots, l_{k-1}, 0$ does, the conflict (j, k) along the given path has been separated. □

Given a conflict, Algorithm 2.1 is polynomial only in the size of the decision diagram. Namely, we have $j - k \leq n$ iterations (line 1) and we might have to loop over at most $n - 1$ neighbors (line 4) but in each layer we have to check all other nodes for equivalence (line 6). Thus the runtime is $\mathcal{O}(n^2 * W)$ where W is the size of the largest layer in the diagram. We also call W the *width* of the decision diagram. While this is not a good runtime guarantee, we do have the following essential property given in [Hoe21]:

Theorem 2.3. *Given a reduced decision diagram D as input and an oracle that provides us with minimal edge conflicts and associated paths in D, repeated application of Algorithm 2.1 yields the unique exact decision diagram after a finite amount of steps.*

This idea, the repeated application of Algorithm 2.1 to separate conflicts, is the basis for van Hoeve's approach to graph coloring with relaxed decision diagrams. In the next section we will describe how to find the conflicts in a diagram and in Section 2.3 we formulate the iterative algorithm. First we will prove the theorem, which is an easy consequence from the next lemma(s).

Lemma 2.4.

(i) Algorithm 2.1 does not introduce new arc-specified paths to the decision diagram, the solution set becomes strictly smaller.

(ii) Given a reduced diagram as input, the output of Algorithm 2.1 is also a reduced diagram.

(iii) Given an edge conflict (j, k) with $j < k$ as input, Algorithm 2.1 increases the size of the diagram by at most $k - j$ nodes.

Proof. (i) New paths and therefore new solutions can only be introduced by re-directing arcs or by merging non-equivalent nodes. Since we copy the outgoing arcs from u_{i+1} to the new node w (lines 8-10), redirecting the path from u_{i+1} to w maintains the original paths. As for merging nodes, only equivalent ones are merged during the algorithm (line 6) and the same set of paths is maintained.
(ii) In Proposition 2.1 we showed that the state information $\mathrm{S}(u)$ is enough to determine the equivalence of two nodes in each layer. Therefore line 6 correctly merges if the temporary node is equivalent to an existing one and the decision diagram stays reduced after application of the algorithm.
(iii) In each iteration at most one node is created (lines 1-2) and inserted into the diagram, there are only $j - k$ iterations. □

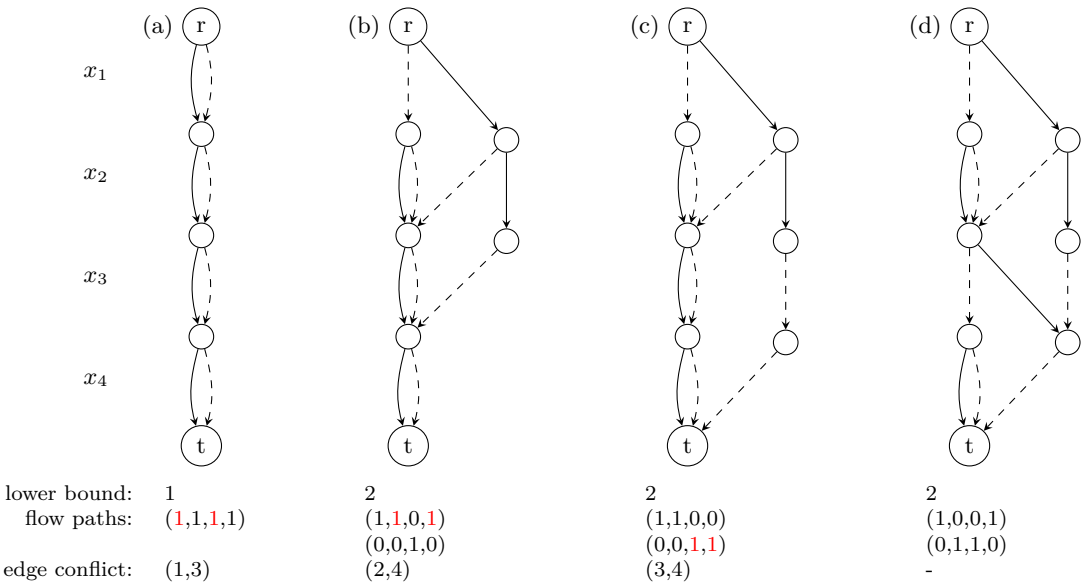


Figure 2.2: Iteratively, a conflict with arc labels is identified and separated to yield the next diagram until in (d) there is no conflict on the selected paths.

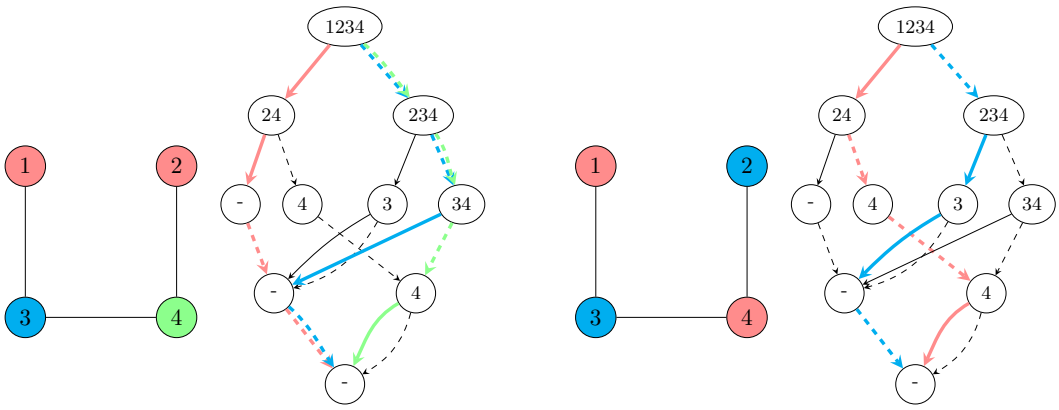
Proof of Theorem 2.3. Lemma 2.2 and Lemma 2.4.(i) guarantee that in each application of Algorithm 2.1 the set of solution becomes strictly smaller and Lemma 2.4.(ii) tells us that the diagram always stays reduced. By only removing conflicts, we eventually end with the exact decision diagram which is reduced and unique. □

In Figure 2.2 we look at a small example of the separation method. The diagram in (a) is the initial diagram of the simple path from Figure 2.1(a) for which we already considered the exact decision diagram. There are five layers and we have the arcs from layer i going to layer $i+1$ corresponding to assignments of the variable x_i. Additionally we are given the conflict $(1,3)$ and the arc specified path $(1,1,1,1)$. In (a), before separating the conflict, the partial path $(1,1,1)$ has a conflict but if we follow the first two 1-arcs in (b), we land in a node with no outgoing 1-arc. The conflict was redirected into a node that forbids the 1-arc so that the refined diagram does not contain the conflict anymore.

Notice that $(1,3)$ on path $(1,0,1,1)$ is a conflict between the same vertices and still exists in (b), we only separated the conflict on a single path. Thus we may need to separate the same conflict on several distinct paths, potentially leading to an exponential amount of conflicts to be separated.

2.2 Graph Coloring Formulated As Network-Flow Problem

Having defined decision diagrams for the discrete optimization problem of finding an independent set, it is still open how we obtain a graph coloring. The key observation to make the connection is that a graph coloring corresponds to a partition of the vertex set into independent sets by the condition that vertices with the same color can not be adjacent. Vice versa, such a partition yields a valid graph coloring. For now, have another look at the exact decision diagram we have seen in Figure 2.1(b). As it is exact, i.e. $\mathrm{Sol}(P) = \mathrm{Sol}(D)$, every path in the diagram gives us a variable assignment that corresponds to the vertices of a stable set. Now say that we take a first path in the diagram given by the labels $(1,1,0,0)$, another by the labels $(0,0,1,0)$ and a last one by the labels $(0,0,0,1)$. This yields the



(a) A partition of the vertex set into three independent sets via paths in the diagram.

(b) ... into two independent sets via paths in the diagram.

Figure 2.3: Extracting independent sets and by that a coloring from an exact decision diagram.

vertex sets $\{1,2\},\{3\}$ and $\{4\}$ which, taken together, give a valid 3-coloring of the graph, see Figure 2.3(a). If we care about the number of independent sets and thus the number of colors used, we could instead choose paths $(1,0,0,1)$ and $(0,1,1,0)$, corresponding to the independent sets $\{1,4\}$ and $\{2,3\}$ and a valid 2-coloring, see Figure 2.3(b).

Now the question is how to find these paths that give us independent sets and how to find them such that we use the minimal amount of independent sets. To answer this, we formulate a constrained network flow problem on the decision diagram $D = (N, A)$. For each arc $a \in A$ we introduce a variable y_a and denote by $\delta^+(u)$ and $\delta^-(u)$ the outgoing respectively incoming arcs of a node $u \in N$. We define the following integer program:

$$(F) = \min \sum_{a \in \delta^+(r)} y_a \tag{2.2}$$

$$\text{s.t.} \sum_{a=(u,v) \mid L(u)=j, \ell(a)=1} y_a \geq 1 \qquad \forall j \in V \tag{2.3}$$

$$\sum_{a \in \delta^-(u)} y_a - \sum_{a \in \delta^+(u)} y_a = 0 \qquad \forall u \in N \setminus \{r,t\} \tag{2.4}$$

$$y_a \in \{0, \ldots, n\} \qquad \forall a \in A \tag{2.5}$$

The objective function of (F) minimizes the amount of flow we send from the root r to the terminal t, Equation (2.4) is the flow conservation constraint and Equation (2.5) ensures the flow is integral. Equation (2.3) is the constraint that in each layer of the decision diagram at least one 1-arc is chosen. To a solution of (F) we denote with obj(F) the value of the objective function. We differ from van Hoeve's work by having (2.3) be "$\geq$" constraints and not "$=$" constraints, leading to a set covering formulation instead of a set partitioning one. The observed results are the same, but require slightly modified arguments. We consider this formulation since we will use this in our later implementation and it is more compatible with a reduction we will see in a later proof. We try to characterize model (F) by the following lemma:

Lemma 2.5. *A solution to (F) corresponds to a set covering of the vertex set V.*

Proof. A flow on a graph as given by a solution to (F) can be decomposed into flow on cycles and flow on paths. Since D is directed and acyclic, we can decompose the computed

flow into only flow on paths. Now a set covering of V can be found by the following algorithm that decomposes the flow into paths:

- ▷ Initialize a set of unselected vertices $V' = V$, a collection $P = \emptyset$ of subsets of V and a residual flow $y'_a = y'_a$ for every arc $a \in A$.
- ▷ While V' is not empty: select any $i \in V'$. By the constraint of Equation (2.3), there exists at least one arc $a = (u, v)$ in layer i, i.e. $u \in L_i$, such that $y'_a \geq 1$. Because of the flow conservation given by Equation (2.4) there exist both an r-u-path and a v-t-path with flow at least 1, together with a this forms an r-t-path p. Define the set $S = \{j \mid a = (u, v) \in p, l(u) = j, l(a) = 1\}$ as the layers of all 1-arcs on p and update $V' = V' \backslash S, P = P + S$ and $y'_a = y_a - 1$ for all $a \in p$.

Here, the $+$ operator indicates the addition of a set to a set, instead of the union. Because of Equation (2.3), each vertex $i \in V$ must be part of at least one path so the collection of sets P is a set covering of V. The objective value is the amount of flow used. Since the flow was integral, this is equal to the number of paths used and we have $|P| = \text{obj}(F)$. □

We can go a little further if the decision diagram is exact:

Theorem 2.6. *If the decision diagram is exact, a solution to model (F) can be transformed into an optimal solution of the graph coloring problem and we have obj$(F) = \chi(G)$. In the relaxed case, obj(F) yields a lower bound for the chromatic number.*

Proof. In the proof of Lemma 2.5 we observed that we can decompose the flow of a solution to (F) into flow on paths corresponding to a set cover of the vertices V. By exactness, each path in the decision diagram corresponds to an independent set. These might not be disjoint but if a vertex is contained in multiple of these independent sets we can remove it from all but one and obtain a vertex partition into independent sets. These represent a valid coloring of the graph and since (F) minimizes the amount of flow, i.e. the number of paths, it minimizes the number of colors needed, therefore $\text{obj}(F) = \chi(G)$. For a relaxed decision diagram D, $\text{obj}(F) \leq \chi(G)$ follows immediately from $\text{Sol}(P) \subseteq \text{Sol}(D)$. □

Another way to show that, on exact decision diagrams, (F) computes the chromatic number is by transforming an optimal solution of (F) into an optimal solution of (VCIP) and vice versa. For that, identify a path of the flow decomposition with a stable set i and the corresponding variable z_i in (VCIP) and set z_i to the amount of flow on the path. Conversely, to a stable set i of variable z_i there exists a path in the exact decision diagram representing that stable set, add z_i flow along that path. We will do such a proof in detail when looking at the linear relaxation of (F) in Section 2.5.

We previously remarked that computing the chromatic number and thus also finding a solution to (F) is NP-hard but one might ask whether that is due to the exponential number of constraints one has because of an exponential sized decision diagram, or whether the integer program is already hard to solve on polynomial sized instances. The following theorem given in [Hoe21] proves the latter:

Theorem 2.7. *Solving model (F) for on an arbitrary decision diagram is NP-hard.*

Proof. Not too surprising after the proof of Lemma 2.5, we proceed by reduction to the NP-hard set covering problem (SC). Given an universe of elements $E = \{e_1, \ldots, e_n\}$ for any fixed ordering of the e_i, and a collection S of subsets of E, we need to find a subset of S with minimum cardinality such that each element in E belongs to at least one set. Given such an instance, we define a decision diagram with $|E| + 1$ layers with layer i representing e_i. For each set $s \in S$ we define an r-t-path by inserting nodes and arcs between layers i and $i + 1$ with label 1 if $e_i \in s$ and 0 otherwise. For that, we iteratively apply the following steps for each $s \in S$:

- ▷ We begin at the root node; set $u = r$
- ▷ For $i = 1, \dots, |E|$: If there does not exist an arc $a = (u, v)$ with label $l(a) = \mathbb{I}[i \in s]$, then insert such a node v to layer $i + 1$. Update $u = v$ and continue.

This will construct a diagram with $|E| + 1$ layers, each one with at most $|S|$ nodes with either one or two outgoing arcs, and is therefore of size $\mathcal{O}(|E||S|)$. Now finding an optimal solution of (F) is the same as solving the minimum set cover problem. □

Note that one can be "lucky" in the case of a relaxed decision diagram, meaning that while not all paths in the diagram correspond to independent sets, maybe those that we choose in the path decomposition do. Then we could terminate with the chromatic number even if the diagram is not exact; we still obtained a valid minimal coloring. Generally we want to use relaxed diagrams because we hope that they will allows us to compute $\chi(G)$ with a decision diagram smaller than the exact one. However, as we only have flow given from solutions to (F) and no explicit paths, we have to decompose the flow into paths ourselves. In an exact decision diagram this decomposition does not matter since all paths represent stable sets. For a relaxed diagram, one could want to look for such a path decomposition that, if possible, only selects stable sets as well and thus finds a feasible solution if one exists. But it turns out that this is generally a difficult thing to do, as formulated and proved in [Hoe21] by reducing it to the graph coloring problem:

Theorem 2.8. *Given a relaxed decision diagram, it is NP-complete to decide whether a solution to the integer program (F) corresponds to a feasible graph coloring solution, i.e. deciding if there exists such a path decomposition into only stable sets giving a feasible coloring. This holds true even when obj(F) is the chromatic number of the input graph.*

When we apply the path decomposition to a solution of model (F) to get flow paths and thus vertex sets, these are often not all stable sets if we work with a relaxed decision diagram. We only have $\text{obj}(F) \leq \chi(G)$ but not a valid coloring. If we find a path that corresponds to a set that is not independent, we can obtain a conflict, that is two adjacent vertices and node path and arc labels, which we can use as input for Algorithm 2.1. This is the idea of Algorithm 2.2 proposed by van Hoeve which either yields such a conflict or certifies that the flow solution corresponds to a feasible graph coloring solution.

Theorem 2.9. *For a given solution to (F) and decision diagram D, Algorithm 2.2 either finds an edge conflict with node paths and arc labels or determines that the solution represents a feasible graph coloring solution. Additionally, it runs in time $\mathcal{O}(n^3)$.*

Proof. The algorithm implements a path decomposition. Since it is a flow model and by constraints given in Equation (2.3), each flow path carries a flow of value at least 1. Because we want to minimize the flow (Equation (2.2)), we further know that the flow is exactly 1, as otherwise we could reduce the flow on the path while still satisfying the constraints, lowering the optimal objective value. While there is still flow (line 1) and thus flow paths, we identify such an r-t-path. In temporary vectors P and L we store the node indices in D[ ][ ] and the arc labels respectively. With the vector S we keep track of all the layers in which we chose an 1-arc, i.e. all the vertices that correspond to the path so far. When we encounter a new 1-arc (line 6), we check if it is adjacent to one of the previous vertices of the path (lines 7-8). If so, we terminate and return the edge conflict we have found. Otherwise we continue and update P, L, S depending on whether it was a 1-arc (lines 10-12) or a 0-arc (lines 13-15). If we finish an r-t-path without conflict, we decrease the remaining flow value and the flow on all arcs along the path by 1 (lines 16-19). If we eliminated all flow and found no conflict on any of the paths, that means the path

Algorithm 2.2: Detecting an edge conflict via flow decomposition

Input: A reduced decision diagram D represented as two-dimensional vector D[ ][ ], a solution to (F) with optimal objective value B and a list of neighbors $N(i)$ for $i \in V$
Output: An edge conflict (j, k) with associated node path and arc labels, or an empty conflict if the solution is feasible
definitions: vector with node indices P, vector with arc labels L, vector of selected vertices S

```
1  while B ≥ 1 do
2      P, L, S ← ∅
3      P.add(1)
4      for i = 1, ..., n do
5          u ← P.last
6          if D[i][u].oneArcFlow ≥ 1 then
7              for j ∈ S do
8                  if j ∈ N(i) then
9                      return conflict (j, i) with node path P_j, ..., P_i and arc labels
                           L_j, ..., L_{i-1}
10             P.add(D[i][u].oneArc)
11             L.add(1)
12             S.add(i)
13         else
14             P.add(D[i][u].zeroArc)
15             L.add(0)
16     for i = j, j + 1, ..., k − 1 do
17         if L_i = 0 then D[i][P_i].zeroArcFlow ← D[i][P_i].zeroArcFlow −1
18         else D[i][P_i].oneArcFlow ← D[i][P_i].oneArcFlow −1
19     B = B − 1
20 return empty conflict
```

decomposition is a decomposition into only stable sets and represents a feasible coloring, we return the empty conflict.

Identifying one r-t-path takes $\mathcal{O}(n)$ (line 4) and checking for adjacency also takes time at most $\mathcal{O}(n)$. That means we need time $\mathcal{O}(n^2)$ for finding a single path and we have at most $B \leq n$ paths, ensuring the algorithm terminates in $\mathcal{O}(n^3)$. □

We want to point out that our observations about Algorithm 2.2 are in no contradiction to Theorem 2.8: the algorithm often returns a conflict even if some path decomposition exists that proves it feasible. Algorithm 2.2 only implements some path decomposition, not one that is always optimal.

2.3 Iterative Refinement

We now want to tie together all that we have seen to describe an iterative refinement procedure on relaxed decision diagrams that computes the chromatic number.

In Algorithm 2.3 we begin with the initial relaxed diagram (line 3) and while we have not found a solution that we know is feasible (line 4) we solve the flow model (F) on the decision diagram D (line 5). The objective value of an optimal solution gives us a lower bound for the chromatic number (line 6). We run Algorithm 2.2 on the flow solution to obtain a conflict (line 7) which we separate in the decision diagram using Algorithm 2.1

Algorithm 2.3: Iterative refinement by conflict detection and separation

Input: An input grapg $G = (V, E)$ and the list of neighbors $N(i)$ for $i \in V$
Output: The chromatic number of G

```
1  lowerBound ←0
2  foundSol ←false
3  D ← initial decision diagram with |V| + 1 layers
4  while foundSol=false do
5      solve model (F) for decision diagram D
6      lowerBound ← obj(F)
7      apply Algorithm 2.2 to the solution of (F) to determine a conflict (j, k) with node path
         P and arc labels L
8      if determined the solution is feasible/ no conflict detected then
9          foundSol ←true
10     else D ← output decision diagram of Algorithm 2.1 with input (j, k) and P, L
11 return lowerBound
```

(line 10). If we only found the empty conflict, i.e. determined that the solution corresponds to a feasible graph coloring (lines 8-9), we exit and return the lower bound of which we know it is the chromatic number (line 12).

Van Hoeve proposed that this algorithm correctly computes the chromatic number:

Theorem 2.10. *Given a graph G, Algorithm 2.3 terminates and computes the chromatic number of G.*

Proof. In Theorem 2.6 we saw the validity of the lower bound in line 6. In case Algorithm 2.2 does not find a conflict, Theorem 2.9 guarantees that we found a feasible solution and that the objective value is the chromatic number. If a conflict is found, Algorithm 2.2 returns the first adjacent $j, k \in V$ such that in layer j and k the 1-arc is used along the flow paths P and L. Therefore the edge conflict (j, k) is minimal, i.e. no edge conflict (j', k') with $j \leq j' < k' < k$ exists on the path and the conflict (j, k), P and L satisfy the input conditions for Algorithm 2.1, which correctly separates that conflict according to Lemma 2.2. With this, termination is guaranteed by Theorem 2.3. □

Take a second look at Figure 2.2 where we have seen how a conflict is separated in a diagram and a new one is identified with methods described in Section 2.2. We end with diagram (d) where two paths are found that contain no conflicts and we terminate with the chromatic number of 2. Observe that the final decision diagram is smaller than the exact one we have seen in Figure 2.1(b).

This hints at the reason why we consider this iterative refinement procedure with relaxed decision diagrams, we hope to compute the chromatic number on a decision diagram not as large as the exact one. Van Hoeve proved the following theoretical theorem. Later we will see practical results as well.

Theorem 2.11. *[Hoe21] For a graph G and given variable ordering, the iterative refinement procedure can compute the chromatic number with a relaxed decision diagram of size polynomial in n, while the exact decision diagram required is exponentially large.*

Proof. We find a family of graphs that displays the wanted behavior. Consider a path from vertex 1 to n, i.e. the simple graph $G = (V, E)$ with vertex labels $V = \{1, \ldots, n\}$ and edge set $E = \{(i, i + 1) \mid i \in \{1, \ldots, n - 1\}\}$. We define a variable ordering on this graph that produces an exact decision diagram with an exponential amount of nodes in n:

- For layers $i = 1, \ldots, \lfloor n/3 \rfloor$ we associate to variable x_i the vertex $1 + 3(i-1)$.
- For layers $i = \lfloor n/3 \rfloor + 1, \ldots, n$ we associate with x_i arbitrarily any of the unassigned vertices.

For an idea of how this ordering looks, refer again to Figure 2.1(a), where the labels in the nodes represent the decision variable x_i they correspond to.

Notice that

1. up to layer $\lfloor n/3 \rfloor$ none of the associated vertices are adjacent meaning each vertex appears in all states of its corresponding layer. Therefore, each node up to layer $\lfloor n/3 \rfloor$ has the 0-arc but also the 1-arc as outgoing edges.
2. the vertices up to layer $\lfloor n/3 \rfloor$ neither have any common neighbors. Consider the transition of a node u with state $S(u)$ into state $S(u')$ according to the transition rule Equation (2.1). Up to layer $\lfloor n/3 \rfloor$, the transition along a 0-arc eliminates the associated vertex and the transition along the 1-arc removes the vertex as well as its neighbors from $S(u)$ to yield $S(u')$. But, since all the neighbors were disjoint, so are the arising states when expanding layer i into layer $i+1$ for $i < \lfloor n/3 \rfloor$.

With 1. implying that up to layer $\lfloor n/3 \rfloor$ each node has two outgoing arcs and 2. implying that each arc ends in a node with a unique state, we conclude that layer $\lfloor n/3 \rfloor$ alone already has $\mathcal{O}(2^{\lfloor n/3 \rfloor})$ nodes.

In the iterative refinement procedure, we begin with the initial relaxed diagram. The first identified solution to model (F) consists of all 1-arcs. After separating a first conflict, the lower bound becomes 2 which is also the chromatic number of our simple graph. For this decision diagram there exists an optimal solution to (F) and associated path decomposition such that the two paths contain no conflicts, i.e. it determines a valid coloring. One path has the 1-arc for each x_i with odd i and the other all the 1-arcs for even i. The procedure therefore terminates with a relaxed decision diagram of size $\mathcal{O}(n)$. □

2.4 Primal Heuristic

As we have seen with Theorem 2.8, it is a difficult problem to decide whether a solution to model (F) corresponds to a feasible graph coloring solution. Thus we might continue separating conflicts even when our lower bound is already equal to the chromatic number. Another way to find out if our lower bound is optimal is to find an upper bound on the chromatic number, if they are equal we can terminate as well. For that, van Hoeve additionally introduced a primal heuristic using the optimal flow solution computed on the relaxed decision diagrams and a similar path decomposition as seen in Algorithm 2.2 to obtain a coloring and thus an upper bound for the chromatic number.

The primal heuristic as presented in Algorithm 2.4 utilizes a similar path decomposition as in Algorithm 2.2. Until all vertices have been assigned to a color class (line 2), a path is identified in the decision diagram. If the flow on the next 1-arc is at least as great as the flow on the 0-arc, the 1-arc is chosen (lines 7-10), otherwise we pick the 0-arc (16-18). If the 1-arc is selected, we add the vertex to the color class of the path (line 12) and to the set of colored vertices (line 13), as well ass adding $N[i]$ to the set of neighbors (line 14). If the color class of the identified path is not empty, i.e. colorUsed=true, we add it to the coloring (lines 19-20) and adjust the residual flow on all edges of the path (lines 22-24). If, after the path decomposition found a path with empty color class, there are still uncolored vertices (line 25), we find the lowest non-conflicting color class for such a vertex and add it (lines 26-30), or create a new color class containing only that vertex (lines 31-32). At the end of the algorithm every vertex has been assigned a color and a valid coloring is returned (line 34).

Algorithm 2.4: Flow-based primal heuristic

Input: A decision diagram D represented as two-dimensional vector D[ ][ ], a solution to (F) and a list of neighbors $N(i)$ for $i \in V$
Output: A coloring of G
definitions: vector with node indices P, vector with arc labels L, vector of selected vertices S, set of neighbors NS and a collection of color classes C

```
1  colorUsed ← true
2  while |S| < n and colorUsed=true do
3      colorClass ← ∅; colorUsed ←false , P, L, NS ← ∅, val ← n
4      P. add(1)
5      for i = 1, . . . , n do
6          u ← P. last
7          if D[i][u]. oneArcFlow ≥ D[i][u]. zeroArcFlow then
8              val ←min(val, D[i][u]. oneArcFlow)
9              P. add(D[i][u]. oneArc)
10             L. add(1)
11             if i ∉ NS then
12                 colorClass. add(1)
13                 S ← S ⋃ {i}
14                 NS ← NS ⋃ N[i]
15                 colorUsed ←true
16         else
17             P. add(D[i][u]. zeroArc)
18             L. add(0)
19     if colorUsed=true then
20         C. add(colorClass)
21         if val > 0 then
22             for i = j, j + 1, . . . , k − 1 do
23                 if L_i = 0 then D[i][P_i]. zeroArcFlow ← D[i][P_i]. zeroArcFlow −1
24                 else D[i][P_i]. oneArcFlow ← D[i][P_i]. oneArcFlow −1
25 if |S| < n then
26     for i ∉ S do
27         for c ∈ C do
28             if ∄j ∈ c | j ∈ N(i) then
29                 c. add(i)
30                 break
31         if i ∉ S then
32             C. add({i})
33         S ← S ⋃ {i}
34 return C
```

The difference to the path decomposition in Algorithm 2.2 is that 1-arcs are not always chosen if they have positive flow, but they can also be chosen if they have zero flow. And if a 1-arc is chosen, it is not added to the color class if it conflicts with previous 1-arcs.

Theorem 2.12. *Given graph G and a solution to model (F) on a decision diagram D, Algorithm 2.4 finds a coloring of G in $\mathcal{O}(n^3)$.*

Proof. Similar to Algorithm 2.2, identifying a path takes $\mathcal{O}(n^2)$. In each iteration (line 2) we color at least one vertex or terminate the path decomposition so we identify at most n paths. Then, if there are still uncolored vertices left, we assign each to the next available color class. Since the color classes are disjoint we have to check at most n edges (line 26-30) so this takes $\mathcal{O}(n)$ for each vertex. The overall complexity is then $\mathcal{O}(n^3) + \mathcal{O}(n^2) = \mathcal{O}(n^3)$. Since, each time we add a vertex to a color class, we check if that class contains adjacent vertices, we obtain a valid coloring without conflicts. □

We extend Algorithm 2.3 by changing the termination condition to using the upper bound obtained by the primal heuristic:

Algorithm 2.5: Iterative refinement with lower and upper bounds

Input: An input graph $G = (V, E)$ and the list of neighbors $N(i)$ for $i \in V$
Output: The chromatic number of G

```
1  lowerBound ←0
2  upperBound ←n
3  D ← initial decision diagram with |V| + 1 layers and width 1
4  while lowerBound < upperBound do
5      solve model (F) for decision diagram D
6      lowerBound ← obj(F)
7      apply Algorithm 2.4 to obtain a coloring C
8      upperBound ←min(upperBound, |C|)
9      apply Algorithm 2.2 to the solution to (F) to determine a conflict (j, k) with node path
         P and arc labels L
10     if determined the solution is feasible/ no conflict detected then
11         upperBound ← obj(F)
12     else D ← output decision diagram of Algorithm 2.1 with input (j, k) and P, L
13 return lowerBound
```

2.5 Linear Relaxation Of The Network Flow Problem

We have now seen a full description of the theoretical ideas behind the algorithm developed by van Hoeve but there are many implementation details to be considered. While in any case the algorithm will not run in polynomial time, there are many factors with a large impact on the practical runtime of an implementation. We will consider a number of these in Section 3.1, but first we take another look at model (F), specifically its linear relaxation:

$$(F') = \min \sum_{a \in \delta^+(r)} y_a \tag{2.6}$$

$$\text{s.t.} \sum_{a=(u,v) | L(u)=j, \ell(a)=1} y_a \geq 1 \qquad \forall j \in V \tag{2.7}$$

$$\sum_{a \in \delta^-(u)} y_a - \sum_{a \in \delta^+(u)} y_a = 0 \qquad \forall u \in N \setminus \{r, t\} \tag{2.8}$$

$$0 \leq y_a \leq n \qquad \forall a \in A \tag{2.9}$$

While we had the negative result of the integer program (F) being NP-complete to solve even on polynomial instances (Theorem 2.7), there exist polynomial time methods to solve linear programs. Modern state-of-the-art mixed-integer programming solvers can potentially solve instances with more than a hundred thousand variables and constraints in a reasonable amount of time. Still, solving model (F) is the runtime bottleneck of the iterative refinement procedure. Therefore, we look at the relaxation (F') and how it can be used during the algorithm. As relaxation, the optimal objective value of (F') is a lower bound for that of (F) and thus also a lower bound to the chromatic number. Additionally, a solution to (F') also yields a flow on the edges of the decision diagram and we can apply Algorithm 2.2 and Algorithm 2.4. The latter stays unchanged when using the relaxation, for the former we change the path decomposition to be similar: we select a 1-arc in the arc if it has at least as much flow as the 0-arc, otherwise we default to the 0-arc. Thus we can still identify conflicts and find a coloring via the primal heuristic, even if the flow is not integral.

Characterization

We had Theorem 2.6 characterizing the integer program, for the relaxation we discovered the following result:

Theorem 2.13. *If the decision diagram is exact, a solution to model (F') can be transformed into a solution of (VCLP), the relaxation of the set covering formulation (VCIP) to compute the chromatic number. Further we have obj$(F') = \chi_f(G)$.*

Proof. We show that an optimal solution $z_i, i \in I$ of (VCLP) can be transformed into an optimal solution $y_a, a \in A$ of (F') . Remember that I was the set of all independent sets of G and A the set of arcs in the decision diagram $D = (N, A)$.

Take a $z_i > 0$ in (VCLP) and construct the arc label specified path $(l_1, \dots, l_n)$ in the decision diagram, where $l_j = 1$ if $j \in i$ and $l_j = 0$ otherwise. Since i is an independent set and D is exact, the path corresponding to the arc labels exists in the decision diagram. Along this path, add z_i flow to each edge. Do this for all $i \in I$ with positive z_i. Any optimal solution uses at most n independent sets, this bound is sharp for the complete graph where every used set is a single vertex. Therefore, since we add at most 1 to the flow along a path ($z_i \leq 1$), constraints (2.9) are satisfied. Since we only add flow paths, the flow condition (2.8) is also fulfilled. For a vertex $j \in V$ it is contained in enough independent sets $\bar{I} \subseteq I$ with positive flow such that $\sum_{i \in \bar{I}} z_i \geq 1$. Translating this to the solution of (F'), this means that much flow is sent through 1-arcs in layer j of the decision diagram, thus the solution satisfies (2.7) as well and is a valid solution of (F').

For a transformation in the other direction, consider an optimal solution $y_a, a \in A$. This flow can be decomposed into flow along paths from r to t. Let $P_1, \dots, P_k$ be such a decomposition, all these paths correspond to an independent set i since D is exact. set x_i to the amount of flow sent along that path. Because the solution was an optimal one, the flow on a path is at most 1, otherwise we could reduce it to 1 and lower the value of the objective function. Therefore, $z_i \leq 1$ and the variable bounds are satisfied. Further, the amount of flow on 1-arcs in layer j is at least 1. This implies that all the independent sets together containing j sum up to at least 1 as well and the constructed z_i are a solution to (VCLP).

To see the optimality of the transformed solutions, consider what the objective function computes in each case: $\sum_{i \in I} z_i$ sums up the value of all used independent sets, these correspond to flow paths in the decision diagram, which in turn determine the amount of flow sent from s to t and therefore yield the flow value $\sum_{a \in \delta^+(r)} y_a$. Thus we conclude

that (F') on an exact decision diagram of a graph G computes the fractional chromatic number $\chi_f(G)$. □

From Theorem 1.4 we conclude that solving the linear relaxation of (F) is NP-hard. Further, this shows that computing the solution to the linear program is not always enough to determine the chromatic number, as we have seen the integrality gap between that and the fractional chromatic number to be in $\mathcal{O}(\log n)$ (Theorem 1.5). Worse, this means that the objective value on relaxed decision diagrams is then a lower bound to the lower bound $\chi_f(G) \leq \chi(G)$. But we can still use a solution of (F') to find valid conflicts to be separated during the iterative refinement procedure by slightly changing Algorithm 2.2 as described above. The linear program can generally be solved a lot faster than (F), we further discuss the use of (F') in our algorithm in Section 3.1.

Safe Lower Bounds

As we will use it however, we utilize the dual solution of (F') to obtain correct lower bounds. That is, correct in the face of floating-point errors that might be produced by the used LP or IP solver. Jansson [Jan04] gives a quite simple example of an explicit linear program on which a solver finds the optimum or reports it infeasible if different variable bounds are given, despite those bounds not changing the set of feasible solutions. Cook et al. [Coo+13] cite a numerically difficult instance where, depending on which solver and which settings are used, the reports range from being able to find an optimal solution, claiming the instance is unbounded, or finding no solution at all. For another instance coming from chip design verification, solving with three different solvers and settings, three different optimal values are reported. This raises some concerns as to the validity of the results obtained by those solvers, especially since the mentioned instance is known and was constructed to be infeasible. Two important factors for the ill conditioning of these problems are the large difference in magnitude between some of the coefficients and that some of the given input data may not even be representable exactly as floating-point number and rounding errors occur (for further reference about the conditioning of linear programs see [Hig02; Klo14]). In our linear program, all coefficients are the integers 0 or 1 so we do not expect as drastic results as feasible/not feasible. But we want to be certain of the correctness of the lower bound that is computed. If by some floating-point error we compute $31 + 10^8$ instead of the optimal value 31, we erroneously report $\lceil 31 + 10^8 \rceil = 32$ as lower bound.

A standard idea for this problem in the context of graph coloring is to fix an $\epsilon > 0$ and use the adjusted value $\lceil \chi_f(G) - \epsilon \rceil$ as a hopefully correct lower bound. If ϵ is chosen to be large enough, this can avoid floating-point errors on well conditioned problems, but its correctness depends on the solution tolerance produced by the used solver. Van Hoeve employed the linear relaxation (F') in his algorithm and chose to use $\lceil z^* - 10^5 \rceil$ as a lower bound, where z^* is the optimal objective value of (F').

We found a different approach of Neumaier [NS04] to transfer nicely to our problem and yield rigorous lower bounds. Adapting the method described in his paper, we can compute the ϵ specific to each instance of (F') and generally find an ϵ smaller than 10^5, though in our experimental tests we never observed $\lceil z^* - 10^5 \rceil$ to give a wrong lower bound. Only on one instance did we find that, in our implementation, $\lceil z^* - 10^5 \rceil$ gave a worse lower bound than the dynamically chosen ϵ. Neumaier proposed steps to postprocess a potentially approximate solution to obtain a rigorous bound for the objective function. He requires reasonable bounds on all variables, which applies to our problem. To a simple

linear program

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t. } Ax = b, \quad x \geq 0 \end{aligned}$$

and its dual

$$\begin{aligned} &\max\ b^T y \\ &\text{s.t. } A^T y \leq c \end{aligned}$$

we find an approximate solution y. Suppose

$$r \geq A^T y - c \tag{2.10}$$

is a rigorous upper bound for the residual $A^T y - c$. Then $c \geq A^T y - r$, hence

$$c^T x \geq (A^T y - r)^T x = y^T Ax - r^T x = y^T b - r^T x. \tag{2.11}$$

Assuming the upper bound $x \leq \bar{x}$ and writing $r_+ = max(r, 0), r_- = max(-r, 0)$, we find

$$c^T x \geq y^T b - r_+^T x \geq y^T b - r_+^T \bar{x}. \tag{2.12}$$

On systems and programming languages with directed rounding of floating-point numbers, switches `roundup` and `rounddown` can be used to control to which nearest floating-point representable number we round. The following code provides a rigourious lower bound μ for $c^T x$:

$$\begin{aligned} &\texttt{roundup}; \\ &r = A^T y - c; \\ &\delta = \bar{x}^T r_+; \\ &\texttt{rounddown}; \\ &\mu = y^T b - \delta; \end{aligned} \tag{2.13}$$

Because $A^T y \leq c$, we have $r \leq 0$ in exact arithmetic. Then we compute $r_+ = 0$ and $\delta = 0$, resulting in the real lower bound $y^T b$. Since we use floating-point arithmetic, we need to round in the required direction according to (2.13). The residual r and the δ are made marginally larger than necessary to fit into a floating-point number, the lower bound μ is made slightly smaller.

Since we use a covering formulation, we have constraints "$\geq$" next to the "$=$" flow constraints. Neumaier also discusses the most general case of linear programs $a \leq Ax \leq b$, which requires interval arithmetic and a lengthier description. We do not need to go that far for our linear program, we can slightly generalize from the simple case discussed above. First, we omit the variable bounds (2.9) since an optimal solution requires flow at most n and thus each arc has flow at most n, resulting in the same optimal solutions when removing the variable bounds. Be aware that we still use n as the rigorous upper bound when computing δ [1].Then our linear program is of the form

[1] We can even use $\chi(G) - 1$ as a stronger upper bound. An optimal solution has at most $\chi(G)$ flow, of which at least 1 must be on a 1-arc so this holds for 0-arcs. Else if a 1-arc lies on all paths, it must be an isolated vertex and can be removed from the graph in a pre-processing step.

$$\min\ c^T x \tag{2.14}$$
$$\text{s.t. } A_1 x \geq b_1 \tag{2.15}$$
$$A_2 x = b_2 \tag{2.16}$$
$$x \geq 0 \tag{2.17}$$

where A_1 is the constraint matrix to the covering constraints (2.7) and A_2 that of the flow constraints (2.8). We also write

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Its dual is

$$\max\ b^T y \tag{2.18}$$
$$\text{s.t. } A^T y \leq c \tag{2.19}$$
$$y_1 \leq 0 \tag{2.20}$$

where y_1 is the first part of y that corresponds to the constraints (2.15). Important is only that Equation (2.11) holds and it does, though the last equality is replaced by another "$\geq$" inequality. With this, Equation (2.12) is valid as well and applying (2.13) yields the desired rigorous lower bound.

We use Neumaier's method in our implementation whenever we solve the linear program (F'). An explicit dualisation of the flow problem (F') is given in Appendix A.1.

Chapter 3

Implementation Details And Experimental Results

3.1 Implementation Details

Van Hoeve presented several main features of his implementation and proposed important factors for the runtime [Hoe21]. We repeat those for our implementation and discuss further ideas and heuristics potentially improving the performance.

3.1.1 Main Features

Variable Ordering

As the most time consuming part of the algorithm is solving (F), we want to do so on as small a decision diagram as possible. The largest factor influencing the size, be it for relaxed or exact diagrams, is the variable ordering in the underlying optimization problem. For the independent set problem, which is also what we use, there has been a lot of effort put into trying to find the variable ordering that yields the smallest exact decision diagram. For example, a variable ordering was found that guarantees to bound the width of layer j by the j-th Fibonacci number [Ber+12b; Ber+14]. Similar to finding a good ordering that allows the sequential greedy algorithm to find an optimal coloring (Proposition 1.2), finding a variable ordering that yields the smallest exact decision diagram is also an NP-hard problem [Weg00]. In practice, one usually uses heuristic strategies, though the orderings in the cited works are applied to obtain the exact decision diagram by top-down compilation. In that case, one can use the state information at each layer to decide on the next variable to choose in the ordering [Ber+12b; Ber+14]. This allows for strategies such as the *k-look ahead*, where the next k variables are chosen as to minimize the size of the next layers. Since we begin with an initial diagram, we have to fix an ordering at the start. This is computed in a pre-processing step by one of the following heuristics:

- ▷ **Lexicographic:** The ordering is taken from the input graph, that is, variable x_i is associated to vertex i.
- ▷ **Dsatur:** Order the variables as they are selected in the Dsatur heuristic.
- ▷ **Max-Connected-Degree:** Order the variables as they are selected in the Max-Connected-Degree heuristic.

In our experimental results we will see that the lexicographic ordering is rarely a viable choice, on the DIMACS instances it is almost always outperformed by the other two orderings. Only on the `queen9_9` instance did it have a speed benefit, though it was also solved to optimality by the Max-Connected-Degree ordering. The latter two show a similar relative performance, with the Max-Connected-Degree ordering performing slightly better.

Relaxing The Network Flow Model

We discussed before the difficulty of solving (F) (Theorem 2.7) and observed that solving its linear relaxation, replacing constraints $y_a \in \{0, \dots, n\}$ by $0 \leq y_a \leq n$ for all $a \in A$, is also NP-hard. On potentially exponential sized instance of an exact decision diagram, this computes $\chi_f(G)$ (Theorem 2.13). Nonetheless, the linear program can generally be solved faster than the integer program and as we have seen can be used to identify conflicts too. Therefore, we begin with solving the linear program in every iteration (changing line 5 of Algorithm 2.5) until a path decomposition with no conflict is found. This yields the optimal value of the relaxation, that is, the fractional chromatic number. Since that lower bound is often not enough, we then switch to solving the integer program and continue as initially described. On a number of instances, using only the linear program is enough; if the primal heuristic finds a coloring with $\chi(G)$ colors and $\lceil \chi_f(G) \rceil = \chi(G)$, we terminate without solving a single integer program. With one of the settings we report on in Section 3.2, we are able to solve 50 of the 137 DIMACS instances to optimality within a time limit of 1 hour. For 45 of those 50 instances, the algorithm is able to report the chromatic number without solving a single integer program. In these cases, iterative refinement using the relaxation (F') is enough. As van Hoeve did in his paper after reporting its speed benefits, we employ the linear program as standard for the iterative refinement procedure, with the extension of methods described in Section 2.5 to obtain rigorous lower bounds.

Note that the objective value of (F) on relaxed decision diagrams can give a stronger lower bound on the chromatic number than the objective value of (F'), the second being again a lower bound of the first. On instances where $\lceil \chi_f(G) \rceil \neq \chi(G)$, this means that we need to solve the LP as long as no conflict-free path decomposition is found, even if solving the IP on the same relaxed decision diagram could already yield the chromatic number. Additionally, we observed in further tests that the primal heuristic can sometimes yield better colorings on flow from the integer program than from the linear relaxation. As we still like the speed advantage we get from using the linear relaxation but could potentially terminate the iterative refinement procedure earlier if we obtain better bounds, we combine the two instead of using them only in separate phases. That is, with some frequency k, we solve in every k-th iteration the integer program, otherwise the linear program. If the LP flow paths have no conflict, we switch to only solving the IP.

Separating More Than A Single Conflict Per Iteration

In Algorithm 2.2 we stopped as soon as a single conflict on a path was found. It is also possible to identify more conflicts from the same solution by considering all paths in the decomposition: instead of immediately returning the conflict, we add it to a list of conflicts, finish the path and look at the next path in the decomposition. When the residual flow is zero, the path decomposition is done and we return the list of conflicts. After that we apply Algorithm 2.1 in sequence to all the found conflicts. As separating a conflict changes arcs and inserts nodes in the decision diagram, we need to be able to guarantee that other conflicts we found and the node and arc paths they lie on are still valid in the separated diagram. But because we only add nodes to the end and not in the middle of each layer and no nodes are removed, the paths given by the node indices in each layer remain valid.

Van Hoeve noticed that finding and separating multiple conflicts at once in each iteration is beneficial in speeding up the iterative refinement. We observed the same result in similar tests; resolving several conflicts in each iteration reduces the number of times we need to solve the linear or integer program. Thus, we incorporate this as standard into our implementation of the algorithm, all conflicts that were found in the path decomposition

are separated at once.

Flow from a solution to the linear program (F') must not be integral and there can be paths with arbitrarily small flow value. A further idea is then to select several but not all found conflicts. One possible restriction is to only select the conflicts that lie on a flow path with a large enough flow value. The thought behind this is that conflicts on a path with larger flow correspond to "more important" or "harder to avoid" conflicts because in the case that the flow is small, only a small amount of flow would need to be diverted to avoid that conflict. We test this as an extra setting of the algorithm.

Improved Relaxation By Redirecting Arcs

The state information stored for each node in the decision diagram is the set of eligible vertices when considering the independent set problem. The transition rule (2.1) determines for a node u with state $S(u)$ in what state $S(u')$ the 0- and 1-arcs end in the next layer. During the exact compilation, the arcs are directed to a node with such state information or one is created. This is not done during the iterative refinement when separating a conflict. A node with the desired state information may not exist and the arc is redirected to some other node. In Algorithm 2.1 the original nodes along the path were chosen, i.e. the arcs of the previous node on the path are copied, ending in a node v. This ensures that $S(u') \subseteq S(v)$ and that we obtain a proper relaxation, but we could choose any such node v that its state satisfies the condition.

Van Hoeve proposes the natural heuristic of choosing the "most similar" node, that is, a node v which maximizes $S(u') \cap S(v)$. We implemented this in Algorithm 2.1 to allow for redirecting arcs to other nodes in the layer and do so according to the "most similar" heuristic. This is added as another setting to the algorithm and we ran tests with and without redirecting arcs; the standard setting will be to use no redirection.

Longest Path Refinement

Intuitively, we expect a large set of vertices to have a high chance of containing two adjacent vertices. Thus if we take any such large set it is likely that we get an edge conflict. If we find the set through taking some path in the relaxed decision diagram, we additionally get the node path and arc labels we need as input for Algorithm 2.1. This was what van Hoeve [Hoe21] observed and proposed to use a refinement procedure based on a longest path (i.e. containing the most 1-arcs) before the iterative refinement. As a special case of the Hamiltonian path problem, finding the longest path in a general graph is NP-hard [PM04]. Yet doing so on a directed acyclic graph can be done in linear time. Our decision diagrams fall into that category of graphs. This is much cheaper than solving (F) or (F') and thus we first look for a conflict on the longest path until no conflict is found or the maximum number of iterations for this procedure is reached. In his experimental results, thus being our standard as well, van Hoeve applies this longest path refinement procedure for at most 100 iterations.

We observed this to be a good number of iterations to use. It covers many of the obvious and necessary to separate conflicts, such as the all 1-arc path in the initial diagram. A lot of the early conflicts found through the flow problem are thus separated without the expense of solving model (F) or (F'). If run for too many iterations however, looking for conflicts only on the longest path can potentially be misguided. When no color class of a minimal coloring contains that many vertices, it is redundant to resolve those conflicts belonging to a large vertex set and it possibly makes the decision diagram unnecessarily large.

3.1.2 Further Ideas

Besides considering conflicts with large flow and mixing LP's and IP's in the first phase of the algorithm, we also contribute a few broader changes or extended ideas to van Hoeve's algorithm.

Using A Clique In The Variable Ordering

When considering previous methods for exact graph coloring in Section 1.3.2, we briefly discussed DSATUR. While at the time a novel approach to graph coloring, a number of improvements have been proposed [San12; FGT17]. Most notably, Sewell suggested to find a large clique in the graph and color these vertices first [Sew96], the idea being that the clique vertices will certainly need to be assigned different colors. Having fixed some vertices and colors already, DSATUR needs to process fewer subproblems.

This transfers to the heuristics we have discussed in Section 1.3.1. The vertices of a clique will definitely need different colors; coloring them first we might avoid some conflicts that could occur when coloring a vertex adjacent to the clique first. We test this as an additional 'heuristic only' setting in our benchmarks by fixing the colors of vertices in a clique before the normal Dsatur or Max-Connected-Degree heuristic is run.

Before finding a variable ordering and proving that it leads to exact decision diagrams with the width in each layer being bounded by the Fibonacci numbers, Bergman et al. [Ber+12b] observed the following for the special case of cliques:

Theorem. *Let G be a clique. Then, for any ordering of the vertices, the width of the exact reduced decision diagram obtained by top-down compilation will be 2.*

Inspired by this result and the improvements to DSATUR as well as the coloring heuristics obtained by fixing vertices of a clique, we employ this in the variable ordering as well. We first find a large clique and put these vertices as first in the variable ordering. After that they are selected as Dsatur or the Max-Connected-Degree heuristic chooses them next. This yields different variable orderings and we test their impact in both the iterative refinement and for the exact decision diagram. A clique is obtained employing a stable set heuristic from Held et al. [1] [HCS12] on the complement graph.

Improvements To The Coloring Heuristics

If we reach a lower bound equal to $\chi(G)$, we can terminate when we have an equal upper bound, or switch to solving the integer program and find a conflict free path decomposition. Running the algorithm with the primal heuristic to get an upper bound, we often terminate by reaching the first condition. Thus, as getting an upper bound equal to the chromatic number is an important termination condition, we explore two ideas to improve the quality of the employed heuristics. Van Hoeve proposed to initialize the lower bound in line 2 of Algorithm 2.5 by the size of a coloring obtained from Dsatur. If Dsatur finds an optimal coloring, we do not need to wait for the primal heuristic to produce one.

We put further effort into improving the bound obtained by either Dsatur or the primal heuristic. First, in the initialization of the upper bound in line 2, we take the size of the minimal coloring obtained by Dsatur, by the Max-Connected-Degree heuristic, or by a slightly modified version of Dsatur where ties are broken only by degree (not by degree in the uncolored subgraph). Each of these can produce different upper bounds and each of these can be the best for some instance.

[1] Code obtained from https://github.com/heldstephan/exactcolors

Second, we augmented both the initial coloring heuristics and the primal heuristic with the *recolor* technique [RA14; HK19]: suppose the next selected vertex v is given a new color k since it is adjacent to at least one vertex in all existing color classes. Thus the color class $C_k = \{v\}$ only contains vertex v. We attempt to reduce the number of colors by reassigning an adjacent vertex u that has been previously colored with a color i. If $C_i \cap N(v) = u$, C_i contains a single adjacent vertex of v. If we are able to remove it from the color class C_i, we could assign v to that color instead. For that we have to reassign u first, this is possible if there exists a color j such that $i < j < k$ and $C_j \cap N(u) = \emptyset$. This means that u can be given color j without any conflicts with its neighbors and v gets color i, thus leaving color k free. If it exists, a minimal j is chosen. The possibility of this arises from the sequential nature of the heuristics, selecting a vertex and assigning it to the lowest possible color. Take a vertex u with color i. Because we chose the lowest color for u every color $j < i$ contains some vertex adjacent to u. However, u could be given the colors $i + 1, \ldots, k$ since those colors were created after u was selected, allowing for the previously described reassignment. Avoiding to create a new color class leads to using less colors, so we use this as standard in our implementation. Specifically for the primal heuristic, we change lines 31-32 of Algorithm 2.4 to try the recoloring before adding the color class with a single vertex.

A New Path Decomposition

Given a flow solution to model (F) or (F'), there are often multiple possible path decompositions. One of them is described by Algorithm 2.2. We prefer to follow 1-arcs over 0-arcs if they have flow at least 1 in case we solved (F) and prefer to follow 1-arcs if their flow is at least as large as the flow on the 0-arcs if we solved (F'). This greedy method always selects the next 1-arc if possible, trying to find a path with many 1-arcs. While this is likely to return a lot of conflicts that we can separate in the decision diagram, it might be detrimental in the later stages of the iterative refinement where we could terminate if we find a path decomposition without conflicts.

Thus we implemented a slight change to Algorithm 2.2, resulting in an often different path decomposition. When choosing a 1-arc in line 6 and the test in lines 7-8 finding that this produces a conflict with a previously selected vertex, we try to avoid that 1-arc if possible. That is, if the 0-arc is also eligible because it has positive flow, we choose it instead. While this is a simple idea, avoiding only the next possible conflict if it can, we hope that this can lead to path decompositions with less or no conflicts where the standard decomposition would find more. We test this as a separate setting in our implementation.

Different Integer Programming Formulations

We took another look at model (F) and considered changes to the formulation that might yield benefits, either in the quality of the bound or the speed of the solving process. One ideas is to make only some variables integral and all others continuous. For example, one could make the 0- and 1-arc of the root node integral and all other variables continuous. Say z^* is the optimal value of the linear program (F'). Computing the flow problem with only the outgoing arcs of the root node integral yields $\lceil z^* \rceil$ as optimal value. This is because the integrality of the first arcs forces the flow and thus the objective value to be integral, but the flow on 1-arcs other than in the first layer can still be fractional, allowing for a potentially lower optimal value than would be computed on the integer program (F). Since we use $\lceil z^* \rceil$ as the bound anyways, this formulation has no benefit for us.

We can get a bound of better quality if we choose to make more variables integral,

namely all 1-arcs. We suppose that this is enough to compute the chromatic number (on exact decision diagrams). However, using this mixed formulation with about half of the variables integral and the other continuous, does not seem to be beneficial for CPLEX to use. CPLEX is able to reduce the pure integer program in a presolve routine to a smaller size with fewer constraints and variables than it can with the mixed integer programming instance, leading to no performance gain. We suppose that this is because the presolve methods are able to make more inferences between variables and more substitutions if they are all integers. Since we did not observe any benefit in this formulation either, we did not further investigate what is computed by the optimal value .

When considering the variable bounds $y_a \leq n, a \in A$, we generally observed that Cplex works better if it is given more freedom. In Section 2.5 we noted that we could also bound the variables by $\chi(G) - 1$ and obtain the same optimal solutions. Instead of this resulting in a better formulation, we observed that the linear and integer programs were solved faster when omitting any bounds on the variables. As we saw in Section 2.5, this does not change the set of optimal solutions. We use this as standard in our implementation.

Pre-processing The Graph

To reduce the size of the graph and therefore speed up the algorithm, we present two reductions as described by Lucet et al. [LMM04].

Dominated Vertices Let x, y be two vertices of a graph G that are not adjacent. If $\delta(y) \subseteq \delta(x)$, then y and its adjacent edges can be removed from the graph without changing the chromatic number. We call y a dominated vertex. To see this, suppose that $\chi(G) = k$ and $k - 1$ colors are needed to color the neighbors of x and vertex x can take the k-th color. Now x and y are not adjacent but all neighbors of y are colored with at most $k - 1$ colors so y can be given the k-th color as well. Thus, if $G \backslash \{y\}$ is k-colorable, then so is G. Therefore we can delete y and compute the chromatic number on the reduced graph. This can be applied recursively until no dominated vertex is found.

Peeling Suppose we have a lower bound k for the chromatic number of a graph G. Then for each vertex x with degree strictly smaller than k, we can erase x and its adjacent edges can be removed from the graph without changing the chromatic number. Assume x has $k - 1$ neighbors, all colored differently in the worst case. Then vertex x can take the k-th color without interfering with the rest of the coloring as all its neighbors have been colored already. Because we know we will be able to assign it a color that is different from all its neighbors, we can delete it from the graph. Coloring this reduced graph yields the chromatic number of the initial graph. Again we apply this recursively until we do not find a vertex with degree strictly smaller than k. In our case, the lower bound is the size of a large clique. This clique is obtained by using a stable set heuristic from Held et al. [HCS12] on the complement graph.

These two techniques can significantly reduce the size of a graph and thus the size of our problem we need to solve, especially on sparse graphs. On several DIMACS instances [JT96], the lower bound from a heuristic clique is enough to completely reduce the graph [LMM04]. That is, we remove vertices until none are left, showing that the lower bound from the size of the clique was actually the chromatic number. In general, these reductions can make moderate instances easier to solve while hard instances will stay difficult. On a lot of the benchmark instances, these reductions had no effect e.g. no vertex was removed. For this reason and since neither van Hoeve [Hoe21] nor Held et al. [HCS12] make use of this reduction in their implementation, we do not employ this in our benchmarks so as to not skew the results based on this pre-processing step.

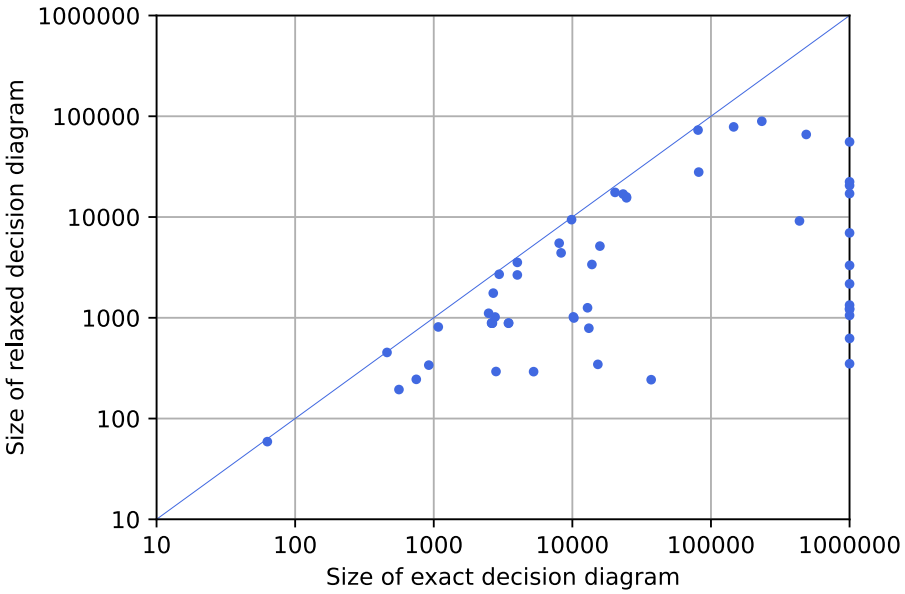


Figure 3.1: Comparison of the size of the relaxed and the exact decision diagram needed on the 54 instances that were solved to optimality by either method.

3.2 Experimental Results

Like van Hoeve, we implemented the method in C++ and performed an experimental evaluation on the 137 DIMACS graph coloring benchmark instances. We use CPLEX 12.6 as the integer and linear programming solver, running on a single thread. All reported experiments are done on a machine with an Intel Xeon X5690@3.47GHZ CPU running CentOS Linux 7. In Appendix A.2 we report performance data from the DFMAX benchmarks to allow for some meaningful–although approximate–comparison with past and future work.

As standard features of the algorithm we

- employ the linear program in a first phase before switching to the integer program,
- separate all conflicts found in a path decomposition,
- utilize the longest path refinement for at most 100 iterations,
- use three coloring heuristics to get an initial upper bound,
- try to recolor vertices to use fewer colors, both in the initial and the primal heuristic,
- solve the linear/integer program with no bounds on the variables.

3.2.1 Benefit Of The Iterative Refinement Procedure

We first formulated how to compute the chromatic number on an exact decision diagram and then introduced the iterative refinement procedure with the reasoning that we expect a smaller, relaxed decision diagram to be enough to compute $\chi(G)$. We saw that it is possible that a polynomial sized relaxed diagram is enough where the exact diagram would be exponentially large (Theorem 2.11). This was a theoretical result indicating their benefit, we now want to see if we observe the same in experimental results. For this first experiment we use the Max-Connected-Degree ordering without arc-redirection. Running the iterative refinement respectively solving model (F) on the exact decision diagram, we compare all instances that were solved to optimality by either method within a time limit of 1 hour, and allowing at most 1 million nodes for the decision diagrams. We report similar results to those of van Hoeve [Hoe21]. Of the 137 instances, 54 were

				Relaxed DD				Exact DD				
Instance	n	m	d	LB	UB	Size	Time	LB	UB	Size	Time	R/E
1-FullIns_3	30	100	0.23	4	4	245	0.02	4	4	748	0.02	0.32
2-FullIns_3	52	201	0.15	5	5	1257	0.18	5	5	12867	2.08	0.09
DSJC125.9	125	6961	0.90	44	44	9417	42.55	44	44	9869	0.64	0.95
david	87	406	0.11	11	11	243	0.00	11	11	37030	6.95	0.00
fpsol2.i.1	496	11654	0.09	65	65	4407	13.15	65	65	8296	0.35	0.53
fpsol2.i.2	451	8691	0.09	30	30	1021	0.58	30	30	10168	0.35	0.10
fpsol2.i.3	425	8688	0.10	30	30	995	0.55	30	30	10258	0.30	0.09
huck	74	301	0.11	11	11	811	0.14	11	11	1078	0.02	0.75
inithx.i.1	864	18707	0.05	54	54	5146	17.63	54	54	15805	1.25	0.32
inithx.i.2	645	13979	0.07	31	31	15509	95.48	31	31	24589	2.70	0.63
inithx.i.3	621	13969	0.07	31	31	15848	79.12	31	31	24551	1.95	0.64
jean	80	254	0.08	10	10	292	0.00	10	10	5252	0.41	0.05
miles1000	128	3216	0.40	42	42	5503	2.38	42	42	8032	0.10	0.68
miles1500	128	5198	0.64	73	73	2662	0.89	73	73	4008	0.07	0.66
miles250	128	387	0.05	8	8	293	0.01	8	8	2813	0.11	0.10
miles500	128	1170	0.14	20	20	345	0.04	20	20	15273	0.71	0.02
miles750	128	2113	0.26	31	31	788	0.15	31	31	13154	0.19	0.05
mulsol.i.1	197	3925	0.20	49	49	1109	0.60	49	49	2488	0.05	0.44
mulsol.i.2	188	3885	0.22	31	31	888	0.25	31	31	2612	0.05	0.33
mulsol.i.3	184	3916	0.23	31	31	884	0.24	31	31	2622	0.04	0.33
mulsol.i.4	185	3946	0.23	31	31	885	0.25	31	31	2637	0.05	0.33
mulsol.i.5	186	3973	0.23	31	31	886	0.24	31	31	2650	0.04	0.33
myciel3	11	20	0.36	4	4	59	0.02	4	4	63	0.01	0.93
myciel4	23	71	0.28	5	5	453	5.74	5	5	460	0.42	0.98
queen5_5	25	160	0.53	5	5	194	0.00	5	5	561	0.01	0.34
queen6_6	36	290	0.46	7	7	1757	0.99	7	7	2687	0.06	0.65
queen7_7	49	476	0.40	7	7	3381	2.01	7	7	13839	0.27	0.24
queen8_8	64	728	0.36	9	9	27917	334.29	9	9	81575	47.75	0.34
r125.1	125	209	0.03	5	5	339	0.01	5	5	921	0.03	0.36
r125.1c	125	7501	0.97	46	46	3547	1.42	46	46	4008	0.04	0.88
r125.5	125	3838	0.50	36	36	16907	267.88	36	36	23243	0.65	0.72
r250.1c	250	30227	0.97	64	64	17562	97.97	64	64	20323	0.18	0.86
zeroin.i.1	211	4100	0.19	49	49	1018	0.51	49	49	2770	0.06	0.36
zeroin.i.2	211	3541	0.16	30	30	888	0.23	30	30	3471	0.08	0.25
zeroin.i.3	206	3540	0.17	30	30	883	0.23	30	30	3458	0.08	0.25
3-FullIns_3	80	346	0.11	6	6	9143	14.98	0	-	435083	timeout	0.02
4-FullIns_3	114	541	0.08	7	7	20556	140.63	0	-	>1M	-	$\leq$ 0.02
5-FullIns_3	154	792	0.07	8	8	55704	2187.85	0	-	>1M	-	$\leq$ 0.05
DSJR500.1	500	3555	0.03	12	12	624	0.05	0	-	>1M	-	$\leq$ 0.00
anna	138	493	0.05	11	11	350	0.01	0	-	>1M	-	$\leq$ 0.00
games120	120	638	0.09	9	9	22379	14.77	0	-	>1M	-	$\leq$ 0.02
le450_25a	450	8260	0.08	25	25	1209	0.69	0	-	>1M	-	$\leq$ 0.00
le450_25b	450	8263	0.08	25	25	1056	0.55	0	-	>1M	-	$\leq$ 0.00
qg.order30	900	26100	0.06	30	30	1337	1.09	0	-	>1M	-	$\leq$ 0.00
queen9_9	81	1056	0.33	10	10	65978	2825.37	0	-	486777	timeout	0.13
r1000.1	1000	14378	0.03	20	20	1234	0.62	0	-	>1M	-	$\leq$ 0.00
r250.1	250	867	0.03	8	8	3312	1.64	0	-	>1M	-	$\leq$ 0.00
school1	385	19095	0.26	14	14	2172	1.70	0	-	>1M	-	$\leq$ 0.00
school1_nsh	352	14612	0.24	14	14	6954	17.57	0	-	>1M	-	$\leq$ 0.00
wap05a	905	43081	0.11	50	50	17088	44.11	0	-	>1M	-	$\leq$ 0.01
2-Insertions_3	37	72	0.11	3	4	2703	timeout	4	4	2964	700.85	$\leq$ 1
DSJC250.9	250	27897	0.90	69	82	72946	timeout	72	72	80682	159.23	$\leq$ 1
DSJR500.1c	500	121275	0.97	76	87	78436	timeout	85	85	145777	2.28	$\leq$ 1
r250.5	250	14849	0.48	65	66	89213	timeout	65	65	232727	26.18	$\leq$ 1

Table 3.1: Comparison of the performance of the iterative refinement procedure and the exact decision diagram on the DIMACS instances that were solved to optimality by either method. For each instance we list the number of vertices (n) the number (m) of edges, and report for each setting the obtained lower bound (LB), obtained upper bound (UB), the solving time (in seconds), and the size of the decision diagram needed. In the last column (R/E) we list the ration of the size of the relaxed and exact decision diagram sizes. The time limit was 3600s and at most 1 million nodes were allowed.

solved completely by either method. Of those, 49 were solved in under a minute and 36 within a second. Iterative refinement was able to solve 50 instances, 11 more than the 39 instances solved by the exact decision diagram. The first was able to solve 15 instances not solved by the latter and conversely, the exact decision diagram solved 4 instances that

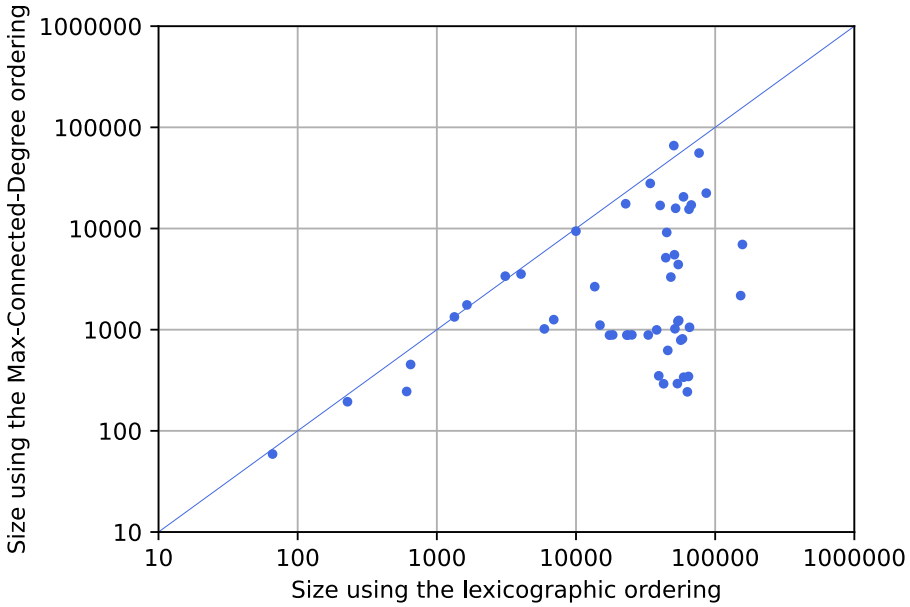


Figure 3.2: Comparison of the size of the relaxed decision diagram needed using the Max-Connected-Degree ordering and the lexicographic ordering on the 50 instances that were solved to optimality by either method.

the iterative method did not. Table 3.1 shows the running times and size of the decision diagrams needed by each method on the 54 solved instances.

Figure 3.1 shows a scatter plot of the sizes of the required decision diagram for the iterative method and the exact decision diagrams for graphs that were solved by either method. We observe that all points lie below the diagonal, meaning that the relaxed decision diagram is always smaller than the exact one, though in some cases the difference is marginal. More than that, we confirm the observation of van Hoeve that the relaxed diagrams required to prove optimality can be several orders of magnitude smaller than the exact decision diagram. This confirms the theoretical benefit of Theorem 2.11 also in our practical results. Two examples signifying this result are `DSRJ500.1` and `anna`, both requiring an exact decision diagram with at least 1 million nodes, but the iterative refinement can compute the chromatic number with a relaxed decision diagram of size 624 respectively 350.

We stated that the variable ordering is the most important factor in determining the size of the decision diagram, relaxed or exact. To see this in practice, we briefly do a similar comparison as we did for relaxed/exact but using the Max-Connected-Degree and the lexicographic ordering, both for the iterative refinement. Figure 3.2 shows a scatter plot of the sizes of the decision diagrams, a detailed table comparing the two can be found in Appendix A.3. With the Max-Connected-Degree ordering, 50 instances were solved, with the lexicographic ordering only 22. There was not a single instance solved by the latter that was not also solved by the first in the given time limit of 3600s. Only for the `queen9_9` instance did lexicographic ordering terminate significantly faster and with a smaller decision diagram than the Max-Connected-Degree ordering.

3.2.2 Comparison Of Van Hoeve's Settings

In Section 3.1.1, we described next to the standard behavior also various parameters influencing the performance of the algorithm, most notably the choice of a variable ordering and whether to use arc redirection. We evaluate each of the variable orderings for the iterative refinement procedure and for compiling the exact decision diagram (Lexicographic, Dsatur and Max-Connected-Degree). For the third ordering, the best performing one, we employ another setting with arc redirection and we use the coloring heuristics to only obtain upper bounds. These 8 settings were run on the DIMACS instances [JT96] with a time limit of 1 hour and using at most 1 million nodes in the decision diagram. The algorithmic settings are:

A: Lexicographic ordering
B: Dsatur ordering
C: Max-Connected-Degree ordering
D: Max-Connected-Degree ordering with arc redirection
E: Lexicographic ordering with exact compilation
F: Dsatur ordering with exact compilation
G: Max-Connected-Degree ordering with exact compilation
H: Coloring heuristic (upper bound only)

We repeat the same evaluation as van Hoeve did in his work. Here we only give a high-level summary of the performance of the settings in terms of number of instances solved and best lower respectively upper bound found. Table A.2 in Appendix A.3 shows the detailed performance of all settings on the DIMACS benchmark instances.

A general overview of the performance of the algorithm is given in the following table. Note that we only report a setting to have found a best upper bound if the upper bound was not also found by the coloring heuristic 'H', since the algorithm uses the heuristic to get an initial bound and does not produce a better bound through the primal heuristic. Of the 137 instances, 54 were solved to optimality by at least 1 setting. The table reports for each setting the number of instances that it solved to optimality and the number of instances where it was one of the settings that found the best lower respectively upper bound:

Setting	#Optimal	#Best LB	#Best UB
A	22	60	7
B	46	111	8
C	50	120	12
D	47	97	15
E	26	32	8
F	39	44	10
G	39	46	8
H	-	-	112

Setting 'C', using the Max-Connected-Degree ordering, performs best, though setting 'B' (Dsatur ordering) and 'D' (Max-Connected-Degree ordering with arc redirection) follow next with only slightly worse performance. Setting 'B' finds more lower best bounds than 'D' but less best upper bound. While our summarized results for the exact decision diagrams are similar, our setting 'D' with arc redirection finds fewer best lower bounds than the setting did for van Hoeve. Additionally, we report for each setting except 'H' a smaller number of best found upper bounds. A reason for this is the improved coloring heuristics we used, with setting 'H' finding the best upper bound on 112 instances.

3.2.3 Detailed Comparison Of Additional Settings

We described further ideas that could be utilized in the algorithm in Section 3.1.2, some of which were implemented as the standard and others resulting in new parameters for the algorithm. We consider these in this section and look at their performance relative to the previous best settings ('C' for the iterative refinement procedure, 'G' for the exact compilation and 'H' for the heuristic) to determine the benefits of the formulated ideas. We use the Max-Connected-Degree ordering as standard in all settings and again run each setting on the DIMACS instances for 1 hour with a limit of 1 million nodes. In Section 3.1.1 we suggested to only separate conflicts that lie on a path with at least some minimal amount of flow; we choose 10^{-4} as the limit. In another setting, we solve in every 100th iteration the integer program instead of the linear relaxation. In Section 3.1.2 we proposed a different path decomposition, and we test what impact using a clique in the variable ordering can have. These ideas yield the additional 6 settings:

- I: Only separate conflicts on path with flow at least 10^{-4}
- J: Solve (F) in every 100th iteration instead of (F')
- K: Use the path decomposition that tries to avoid conflicts
- L: Fix the vertices of a clique in the variable ordering
- M: Fix the vertices of a clique in the variable ordering with exact compilation
- N: Fix the vertices of a clique in the variable ordering with the coloring heuristics

We repeat the same evaluation as before, with Table 3.2 reporting for the DIMACS instances the best lower and upper bound found by any setting and listing which settings were able to obtain that bound. If the upper bound was found by the (alternative) coloring heuristic, we only list 'H' respectively 'N' or both.

Of the 137 instances, 55 were solved to optimality by at least 1 setting. We summarize the results for these new settings and settings 'C', 'G' and 'H' in the following table. It reports again the number of optimally solved instances and the number of instances where the setting computed the best found lower respectively upper bound:

Setting	#Optimal	#Best LB	#Best UB
C	50	110	13
G	39	46	8
H	-	-	106
I	50	111	13
J	50	110	11
K	51	110	11
L	50	127	12
M	39	46	9
N	-	-	106

Settings 'I', 'J' and 'K' perform about as good as the baseline setting 'C', only sometimes finding one more or less best bounds, and setting 'K' solving 1 instance more to optimality. For exact compilation with and without fixing the vertices of a clique in the variable ordering, there is no notable difference. In the coloring heuristic we see mixed results. While both 'H' and 'N' have found the same number of best upper bounds, they both individually find less than the 112 of setting 'H' when it was the only heuristic used. This is because on some instances fixing the vertices of a clique has an advantage while on other it is detrimental. Combining both heuristics yields an improved 115 best bounds. While not being able to solve more instances than setting 'C', setting 'L', the iterative refinement procedure using a clique in the variable ordering, is able to find 17 more best lower bounds.

						Decision Diagram				
Instance	n	m	d	$\underline{X}$	$\overline{X}$	LB	Setting	UB	Setting	
1-FullIns_3	30	100	0.23	4	4	4	CGIJKLM	4	HN	*
1-FullIns_4	93	593	0.14	5	5	4	CIJKL	5	HN	
1-FullIns_5	282	3247	0.08	6	6	4	CIJKL	6	HN	
1-Insertions_4	67	232	0.10	5	5	3	CGIJKLM	5	HN	
1-Insertions_5	202	1227	0.06	6	6	3	CIJKL	6	HN	
1-Insertions_6	607	6337	0.03	4	7	3	CIJKL	7	HN	
2-FullIns_3	52	201	0.15	5	5	5	CGIJKLM	5	HN	*
2-FullIns_4	212	1621	0.07	6	6	5	CIJ	6	HN	
2-FullIns_5	852	12201	0.03	7	7	4	CIJKL	7	HN	
2-Insertions_3	37	72	0.11	4	4	4	GKM	4	HN	*
2-Insertions_4	149	541	0.05	5	5	3	CIJKL	5	HN	
2-Insertions_5	597	3936	0.02	6	6	3	CIJKL	6	HN	
3-FullIns_3	80	346	0.11	6	6	6	CIJKL	6	HN	*
3-FullIns_4	405	3524	0.04	7	7	5	CIJKL	7	HN	
3-FullIns_5	2030	33751	0.02	8	8	5	CIJKL	8	HN	
3-Insertions_3	56	110	0.07	4	4	3	CGIJKLM	4	HN	
3-Insertions_4	281	1046	0.03	5	5	3	CIJKL	5	HN	
3-Insertions_5	1406	9695	0.01	4	6	3	CIJKL	6	HN	
4-FullIns_3	114	541	0.08	7	7	7	CIJKL	7	HN	*
4-FullIns_4	690	6650	0.03	8	8	7	CIJKL	8	HN	
4-FullIns_5	4146	77305	0.01	9	9	7	CIJKL	9	HN	
4-Insertions_3	79	156	0.05	4	4	3	CGIJKLM	4	HN	
4-Insertions_4	475	1795	0.02	5	5	3	CIJKL	5	HN	
5-FullIns_3	154	792	0.07	8	8	8	CIJKL	8	HN	*
5-FullIns_4	1085	11395	0.02	9	9	8	I	9	HN	
abb313GPIA	1557	53356	0.04	9	9	8	L	10	H	
anna	138	493	0.05	11	11	11	CIJKL	11	HN	*
ash331GPIA	662	4181	0.02	4	4	4	CIJK	5	HN	
ash608GPIA	1216	7844	0.01	4	4	3	CIJKL	5	HN	
ash958GPIA	1916	12506	0.01	4	4	4	JL	5	HN	
C2000.5	2000	999836	0.50	99	145	19	L	203	H	
C2000.9	2000	1799532	0.90	98	400	74	L	529	N	
C4000.5	4000	4000268	0.50	107	259	19	CIJKL	373	HN	
david	87	406	0.11	11	11	11	CGIJKLM	11	HN	*
DSJC1000.1	1000	49629	0.10	10	20	6	CIJKL	27	HN	
DSJC1000.5	1000	249826	0.50	73	82	18	L	115	N	
DSJC1000.9	1000	449449	0.90	216	222	68	K	289	H	
DSJC125.1	125	736	0.09	5	5	5	CIJKL	6	HN	
DSJC125.5	125	3891	0.50	17	17	16	GM	18	M	
DSJC125.9	125	6961	0.90	44	44	44	CGIJKLM	44	CGIJKLM	*
DSJC250.1	250	3218	0.10	6	8	5	CIJKL	10	HN	
DSJC250.5	250	15668	0.50	26	28	15	CIJKL	36	CIJKL	
DSJC250.9	250	27897	0.90	72	72	72	GM	72	GM	*
DSJC500.1	500	12458	0.10	9	12	6	L	16	H	
DSJC500.5	500	62624	0.50	43	47	17	CIJKL	63	CIJL	
DSJC500.9	500	112437	0.90	123	126	123	GM	131	GM	
DSJR500.1	500	3555	0.03	12	12	12	CIJKL	12	HN	*
DSJR500.1c	500	121275	0.97	85	85	85	GM	85	GM	*
DSJR500.5	500	58862	0.47	122	122	122	L	127	H	
flat1000_50_0	1000	245000	0.49	50	50	17	CIJKL	113	HN	
flat1000_60_0	1000	245830	0.49	60	60	17	L	112	N	
flat1000_76_0	1000	246708	0.49	72	81	17	CIJKL	114	H	
flat300_20_0	300	21375	0.48	20	20	15	CIJKL	39	CIJKL	
flat300_26_0	300	21633	0.48	26	26	15	CIJKL	40	CIJKL	
flat300_28_0	300	21695	0.48	28	28	15	CIJKL	40	CIJL	
fpsol2.i.1	496	11654	0.09	65	65	65	CGIJKLM	65	HN	*
fpsol2.i.2	451	8691	0.09	30	30	30	CGIJKLM	30	HN	*
fpsol2.i.3	425	8688	0.10	30	30	30	CGIJKLM	30	HN	*
games120	120	638	0.09	9	9	9	CIJKL	9	HN	*
homer	561	1629	0.01	13	13	13	L	13	HN	*
huck	74	301	0.11	11	11	11	CGIJKLM	11	HN	*
inithx.i.1	864	18707	0.05	54	54	54	CGIJKLM	54	HN	*
inithx.i.2	645	13979	0.07	31	31	31	CGIJKLM	31	HN	*
inithx.i.3	621	13969	0.07	31	31	31	CGIJKLM	31	HN	*
jean	80	254	0.08	10	10	10	CGIJKLM	10	HN	*
latin_square_10	900	307350	0.76	90	97	90	CIJKL	130	HN	

Table 3.2: Best lower and upper bounds obtained by any of the settings on all DIMACS instances. Next to the data of the graphs, we list the best known lower ($\underline{X}$) and upper bounds ($\overline{X}$). Fully solved instances are marked with an asterisk.

						Decision Diagram				
Instance	n	m	d	$\underline{X}$	$\overline{X}$	LB	Setting	UB	Setting	
le450_15a	450	8168	0.08	15	15	15	CIJKL	16	N	
le450_15b	450	8169	0.08	15	15	15	CIJKL	16	HN	
le450_15c	450	16680	0.17	15	15	15	CIJKL	24	HN	
le450_15d	450	16750	0.17	15	15	15	CIJKL	24	HN	
le450_25a	450	8260	0.08	25	25	25	CIJKL	25	HN	*
le450_25b	450	8263	0.08	25	25	25	CIJKL	25	HN	*
le450_25c	450	17343	0.17	25	25	25	CIJKL	28	N	
le450_25d	450	17425	0.17	25	25	25	CIJKL	28	HN	
le450_5a	450	5714	0.06	5	5	5	CIJKL	9	N	
le450_5b	450	5734	0.06	5	5	5	CIJKL	7	H	
le450_5c	450	9803	0.10	5	5	5	CIJKL	8	I	
le450_5d	450	9757	0.10	5	5	5	CIJKL	9	CK	
miles1000	128	3216	0.40	42	42	42	CGIJKLM	42	HN	*
miles1500	128	5198	0.64	73	73	73	CGIJKLM	73	HN	*
miles250	128	387	0.05	8	8	8	CGIJKLM	8	HN	*
miles500	128	1170	0.14	20	20	20	CGIJKLM	20	HN	*
miles750	128	2113	0.26	31	31	31	CGIJKLM	31	HN	*
mug100_1	100	166	0.03	4	4	3	CIJKL	4	HN	
mug100_25	100	166	0.03	4	4	3	CIJKL	4	HN	
mug88_1	88	146	0.04	4	4	3	CIJKL	4	HN	
mug88_25	88	146	0.04	4	4	3	CIJKL	4	HN	
mulsol.i.1	197	3925	0.20	49	49	49	CGIJKLM	49	HN	*
mulsol.i.2	188	3885	0.22	31	31	31	CGIJKLM	31	HN	*
mulsol.i.3	184	3916	0.23	31	31	31	CGIJKLM	31	HN	*
mulsol.i.4	185	3946	0.23	31	31	31	CGIJKLM	31	HN	*
mulsol.i.5	186	3973	0.23	31	31	31	CGIJKLM	31	HN	*
myciel3	11	20	0.36	4	4	4	CGIJKLM	4	HN	*
myciel4	23	71	0.28	5	5	5	CGIJKLM	5	HN	*
myciel5	47	236	0.22	6	6	5	CGIJKLM	6	HN	
myciel6	95	755	0.17	7	7	4	CGIJKLM	7	HN	
myciel7	191	2360	0.13	8	8	4	CIJKL	8	HN	
qg.order100	10000	990000	0.02	100	100	100	CIJKL	102	HN	
qg.order30	900	26100	0.06	30	30	30	CIJKL	30	HN	*
qg.order40	1600	62400	0.05	40	40	40	CIJKL	41	HN	
qg.order60	3600	212400	0.03	60	60	60	CIJKL	62	HN	
queen10_10	100	1470	0.30	11	11	10	CIJKL	13	HN	
queen11_11	121	1980	0.27	11	11	11	CIL	14	HN	
queen12_12	144	2596	0.25	12	12	12	L	15	H	
queen13_13	169	3328	0.23	13	13	13	L	16	H	
queen14_14	196	4186	0.22	14	14	14	L	17	CIKL	
queen15_15	225	5180	0.21	15	15	15	L	18	N	
queen16_16	256	6320	0.19	16	17	16	L	20	HN	
queen5_5	25	160	0.53	5	5	5	CGIJKLM	5	HN	*
queen6_6	36	290	0.46	7	7	7	CGIJKLM	7	CGIJKLM	*
queen7_7	49	476	0.40	7	7	7	CGIJKLM	7	CGIJKLM	*
queen8_12	96	1368	0.30	12	12	12	L	13	HN	
queen8_8	64	728	0.36	9	9	9	CGIJKLM	9	CGIJKLM	*
queen9_9	81	1056	0.33	10	10	10	CIJKL	10	CIJK	*
r1000.1	1000	14378	0.03	20	20	20	CIJKL	20	HN	*
r1000.1c	1000	485090	0.97	96	98	85	L	104	HN	
r1000.5	1000	238267	0.48	234	234	234	L	239	N	
r125.1	125	209	0.03	5	5	5	CGIJKLM	5	HN	*
r125.1c	125	7501	0.97	46	46	46	CGIJKLM	46	HN	*
r125.5	125	3838	0.50	36	36	36	CGIJKLM	36	H	*
r250.1	250	867	0.03	8	8	8	CIJKL	8	HN	*
r250.1c	250	30227	0.97	64	64	64	CGIJKLM	64	HN	*
r250.5	250	14849	0.48	65	65	65	CGIJKM	65	GLM	*
school1	385	19095	0.26	14	14	14	CIJKL	14	CIJKL	*
school1_nsh	352	14612	0.24	14	14	14	CIJKL	14	N	*
wap01a	2368	110871	0.04	41	43	41	L	45	HN	
wap02a	2464	111742	0.04	40	42	40	CIJKL	44	N	
wap03a	4730	286722	0.03	40	47	40	CIJKL	50	N	
wap04a	5231	294902	0.02	40	42	40	CIJKL	46	HN	
wap05a	905	43081	0.11	50	50	50	CIJKL	50	HN	*
wap06a	947	43571	0.10	40	40	40	CIJKL	42	H	
wap07a	1809	103368	0.06	40	41	40	L	44	N	
wap08a	1870	104176	0.06	40	42	40	L	43	H	

Table 3.2: (continued)

						Decision Diagram				
Instance	n	m	d	$\underline{X}$	$\overline{X}$	LB	Setting	UB	Setting	
will199GPIA	701	6772	0.03	7	7	6	CIJKL	7	HN	
zeroin.i.1	211	4100	0.19	49	49	49	CGIJKLM	49	HN	*
zeroin.i.2	211	3541	0.16	30	30	30	CGIJKLM	30	HN	*
zeroin.i.3	206	3540	0.17	30	30	30	CGIJKLM	30	HN	*

Table 3.2: (continued)

We only observe setting 'L' to have a considerable advantage compared to setting 'C' and other new settings when it comes to finding the best lower bounds. Additionally we note that when both settings terminate or find the same lower bound, setting 'L' often does so in a considerably shorter amount of time and with a smaller decision diagram than setting 'C'. One interesting example, though it should be considered an exception, is instance `homer`. While the standard setting is unable to solve this instance in the time limit of 1 hour and is terminated with a decision diagram with 47284 nodes, using the clique in the vertex ordering allows the iterative refinement procedure to terminate in 0.16 seconds with a decision diagram of size 664. Only on the highly structured `queen` instances did setting 'C' have a speed benefit, even solving `queen9_9` to optimality that setting 'L' was unable to solve. A more detailed comparison between these two settings can be found in Appendix A.3.

3.2.4 Benchmark Against Exactcolors

Because it was run on a different platform, Table 3.2 does not provide an accurate comparison of our implementation with that of van Hoeve in terms of running time. Therefore, we repeat his benchmarks against a different algorithm to solve the graph coloring problem. We have discussed the approach of Merothra and Trick [MT96] based on column generation in Section 1.3.2, van Hoeve chose the implementation of this from Held et al. [HCS12] to compare against. As we pointed out, both the column generation and the decision diagram approach are based around finding a vertex covering into independent sets, though the former adds independent sets until enough for an optimal solutions are found, and the latter removes vertex sets that are not independent until an optimal solution is found.

We refer to the implementation of Held et al. by 'Exactcolors'[2] and to our implementation of van Hoeve's decision diagram approach by 'DD'. Both programs were compiled on the same machine and using the same version of CPLEX as mentioned in the beginning of this section. We run both methods on all 137 DIMACS instances with a limit of 1 hour and at most 1 million nodes for DD, using only a fixed setting for our code. We choose setting 'L' as it showed an improvement to the number of best bounds found. Table 3.3 reports the lower respectively upper bounds and the running time of both methods. Our benchmarks result in the following high-level comparison of the two methods:

- Exactcolors solves 61 instances to optimality, DD only 50.
- Exactcolors does not return a lower bound for 24 instances, while DD returns one for all instances.
- Exactcolors finds slightly more best known lower bound than DD (90 vs 83).
- Both methods report a similar number of best known upper bounds (84 for Exactcolors and 79 for DD).
- Exactcolors finds 27 better lower bounds whereas DD finds 25 better lower bound. On 85 instances they find the same lower bound.

[2]Code obtained from https://github.com/heldstephan/exactcolors

						Exactcolors [HCS12]			Decision Diagram		
Instance	n	m	d	$\underline{X}$	$\overline{X}$	LB	UB	Time	LB	UB	Time
1-FullIns_3	30	100	0.23	4	4	4	4	0.00	4	4	0.01
1-FullIns_4	93	593	0.14	5	5	4	5	timeout	4	5	timeout
1-FullIns_5	282	3247	0.08	6	6	4	6	timeout	4	6	timeout
1-Insertions_4	67	232	0.10	5	5	3	5	timeout	3	5	timeout
1-Insertions_5	202	1227	0.06	6	6	3	6	timeout	3	6	timeout
1-Insertions_6	607	6337	0.03	4	7	4	7	timeout	3	7	timeout
2-FullIns_3	52	201	0.15	5	5	5	5	0.00	5	5	0.19
2-FullIns_4	212	1621	0.07	6	6	5	6	timeout	4	6	timeout
2-FullIns_5	852	12201	0.03	7	7	5	7	timeout	4	7	timeout
2-Insertions_3	37	72	0.11	4	4	4	4	158.14	3	4	timeout
2-Insertions_4	149	541	0.05	5	5	3	5	timeout	3	5	timeout
2-Insertions_5	597	3936	0.02	6	6	3	6	timeout	3	6	timeout
3-FullIns_3	80	346	0.11	6	6	6	6	0.01	6	6	15.61
3-FullIns_4	405	3524	0.04	7	7	6	7	timeout	5	7	timeout
3-FullIns_5	2030	33751	0.02	8	8	6	8	timeout	5	8	timeout
3-Insertions_3	56	110	0.07	4	4	3	4	timeout	3	4	timeout
3-Insertions_4	281	1046	0.03	5	5	3	5	timeout	3	5	timeout
3-Insertions_5	1406	9695	0.01	4	6	-	6	timeout	3	6	timeout
4-FullIns_3	114	541	0.08	7	7	7	7	0.01	7	7	142.50
4-FullIns_4	690	6650	0.03	8	8	7	8	timeout	7	8	timeout
4-FullIns_5	4146	77305	0.01	9	9	7	9	timeout	7	9	timeout
4-Insertions_3	79	156	0.05	4	4	3	4	timeout	3	4	timeout
4-Insertions_4	475	1795	0.02	5	5	3	5	timeout	3	5	timeout
5-FullIns_3	154	792	0.07	8	8	8	8	0.02	8	8	1936.40
5-FullIns_4	1085	11395	0.02	9	9	8	9	timeout	7	9	timeout
abb313GPIA	1557	53356	0.04	9	9	-	10	timeout	8	15	timeout
anna	138	493	0.05	11	11	11	11	0.00	11	11	0.01
ash331GPIA	662	4181	0.02	4	4	4	6	timeout	3	5	timeout
ash608GPIA	1216	7844	0.01	4	4	-	6	timeout	3	5	timeout
ash958GPIA	1916	12506	0.01	4	4	-	6	timeout	4	6	timeout
C2000.5	2000	999836	0.50	99	145	-	207	timeout	19	207	timeout
C2000.9	2000	1799532	0.90	98	400	-	550	timeout	74	531	timeout
C4000.5	4000	4000268	0.50	107	259	-	376	timeout	19	374	timeout
david	87	406	0.11	11	11	11	11	0.00	11	11	0.00
DSJC1000.1	1000	49629	0.10	10	20	-	25	timeout	6	27	timeout
DSJC1000.5	1000	249826	0.50	73	82	-	114	timeout	18	115	timeout
DSJC1000.9	1000	449449	0.90	216	222	-	301	timeout	65	292	timeout
DSJC125.1	125	736	0.09	5	5	5	6	timeout	5	6	timeout
DSJC125.5	125	3891	0.50	17	17	16	18	timeout	14	20	timeout
DSJC125.9	125	6961	0.90	44	44	44	44	9.14	44	44	31.50
DSJC250.1	250	3218	0.10	6	8	7	10	timeout	5	10	timeout
DSJC250.5	250	15668	0.50	26	28	26	30	timeout	15	36	timeout
DSJC250.9	250	27897	0.90	72	72	71	72	timeout	65	82	timeout
DSJC500.1	500	12458	0.10	9	12	-	16	timeout	6	16	timeout
DSJC500.5	500	62624	0.50	43	47	43	65	timeout	17	63	timeout
DSJC500.9	500	112437	0.90	123	126	123	129	timeout	64	154	timeout
DSJR500.1	500	3555	0.03	12	12	12	12	536.37	12	12	0.11
DSJR500.1c	500	121275	0.97	85	85	85	85	1139.06	79	88	timeout
DSJR500.5	500	58862	0.47	122	122	122	132	timeout	122	127	timeout
flat1000_50_0	1000	245000	0.49	50	50	-	113	timeout	17	113	timeout
flat1000_60_0	1000	245830	0.49	60	60	-	112	timeout	17	112	timeout
flat1000_76_0	1000	246708	0.49	72	81	-	115	timeout	17	115	timeout
flat300_20_0	300	21375	0.48	20	20	20	42	timeout	15	39	timeout
flat300_26_0	300	21633	0.48	26	26	26	26	2797.04	15	40	timeout
flat300_28_0	300	21695	0.48	28	28	28	33	timeout	15	40	timeout
fpsol2.i.1	496	11654	0.09	65	65	65	65	0.64	65	65	2.61
fpsol2.i.2	451	8691	0.09	30	30	30	30	0.46	30	30	0.30
fpsol2.i.3	425	8688	0.10	30	30	30	30	0.54	30	30	0.27
games120	120	638	0.09	9	9	9	9	0.01	9	9	0.01
homer	561	1629	0.01	13	13	13	13	0.13	13	13	0.16
huck	74	301	0.11	11	11	11	11	0.00	11	11	0.00
inithx.i.1	864	18707	0.05	54	54	54	54	2.21	54	54	3.88
inithx.i.2	645	13979	0.07	31	31	31	31	1.89	31	31	0.84
inithx.i.3	621	13969	0.07	31	31	31	31	0.65	31	31	0.71
jean	80	254	0.08	10	10	10	10	0.00	10	10	0.00
latin_square_10	900	307350	0.76	90	97	90	129	timeout	90	130	timeout

Table 3.3: Comparing the performance of our implementation using setting 'L' and the column generation implementation of Held et al. [HCS12]. We list the graph data, including best known bounds, and report the lower and upper bounds obtained by each method as well as solving time, if it was within the time limit of 1 hour.

						Exactcolors [HCS12]			Decision Diagram		
Instance	n	m	d	$\underline{X}$	$\overline{X}$	LB	UB	Time	LB	UB	Time
le450_15a	450	8168	0.08	15	15	15	17	timeout	15	16	timeout
le450_15b	450	8169	0.08	15	15	15	17	timeout	15	16	timeout
le450_15c	450	16680	0.17	15	15	15	24	timeout	15	24	timeout
le450_15d	450	16750	0.17	15	15	15	24	timeout	15	24	timeout
le450_25a	450	8260	0.08	25	25	25	25	3.39	25	25	0.36
le450_25b	450	8263	0.08	25	25	25	25	3.58	25	25	0.36
le450_25c	450	17343	0.17	25	25	25	28	timeout	25	28	timeout
le450_25d	450	17425	0.17	25	25	25	29	timeout	25	28	timeout
le450_5a	450	5714	0.06	5	5	5	10	timeout	5	9	timeout
le450_5b	450	5734	0.06	5	5	-	7	timeout	5	10	timeout
le450_5c	450	9803	0.10	5	5	5	11	timeout	5	9	timeout
le450_5d	450	9757	0.10	5	5	5	11	timeout	5	10	timeout
miles1000	128	3216	0.40	42	42	42	42	0.11	42	42	0.20
miles1500	128	5198	0.64	73	73	73	73	0.22	73	73	0.59
miles250	128	387	0.05	8	8	8	8	0.00	8	8	0.01
miles500	128	1170	0.14	20	20	20	20	0.02	20	20	0.03
miles750	128	2113	0.26	31	31	31	31	0.08	31	31	0.10
mug100_1	100	166	0.03	4	4	4	4	2.80	3	4	timeout
mug100_25	100	166	0.03	4	4	4	4	2.61	3	4	timeout
mug88_1	88	146	0.04	4	4	4	4	1.24	3	4	timeout
mug88_25	88	146	0.04	4	4	4	4	1.54	3	4	timeout
mulsol.i.1	197	3925	0.20	49	49	49	49	0.18	49	49	0.58
mulsol.i.2	188	3885	0.22	31	31	31	31	0.13	31	31	0.17
mulsol.i.3	184	3916	0.23	31	31	31	31	0.04	31	31	0.15
mulsol.i.4	185	3946	0.23	31	31	31	31	0.06	31	31	0.16
mulsol.i.5	186	3973	0.23	31	31	31	31	0.06	31	31	0.16
myciel3	11	20	0.36	4	4	4	4	0.00	4	4	0.03
myciel4	23	71	0.28	5	5	5	5	3.61	5	5	6.27
myciel5	47	236	0.22	6	6	4	6	timeout	5	6	timeout
myciel6	95	755	0.17	7	7	4	7	timeout	4	7	timeout
myciel7	191	2360	0.13	8	8	5	8	timeout	4	8	timeout
qg.order100	10000	990000	0.02	100	100	-	106	timeout	100	104	timeout
qg.order30	900	26100	0.06	30	30	30	32	timeout	30	30	1.39
qg.order40	1600	62400	0.05	40	40	-	42	timeout	40	42	timeout
qg.order60	3600	212400	0.03	60	60	-	63	timeout	60	64	timeout
queen10_10	100	1470	0.30	11	11	11	11	301.80	10	13	timeout
queen11_11	121	1980	0.27	11	11	11	12	timeout	11	14	timeout
queen12_12	144	2596	0.25	12	12	12	14	timeout	12	15	timeout
queen13_13	169	3328	0.23	13	13	13	14	timeout	13	17	timeout
queen14_14	196	4186	0.22	14	14	14	15	timeout	14	17	timeout
queen15_15	225	5180	0.21	15	15	15	17	timeout	15	18	timeout
queen16_16	256	6320	0.19	16	17	16	21	timeout	16	20	timeout
queen5_5	25	160	0.53	5	5	5	5	0.00	5	5	0.00
queen6_6	36	290	0.46	7	7	7	7	0.31	7	7	1.17
queen7_7	49	476	0.40	7	7	7	7	0.62	7	7	1.68
queen8_12	96	1368	0.30	12	12	12	12	10.85	12	13	timeout
queen8_8	64	728	0.36	9	9	9	9	7.53	9	9	681.08
queen9_9	81	1056	0.33	10	10	10	10	12.98	10	11	timeout
r1000.1	1000	14378	0.03	20	20	20	20	1.18	20	20	0.78
r1000.1c	1000	485090	0.97	96	98	96	107	timeout	85	104	timeout
r1000.5	1000	238267	0.48	234	234	234	248	timeout	234	239	timeout
r125.1	125	209	0.03	5	5	5	5	0.00	5	5	0.01
r125.1c	125	7501	0.97	46	46	46	46	0.02	46	46	0.14
r125.5	125	3838	0.50	36	36	36	36	21.82	36	36	13.30
r250.1	250	867	0.03	8	8	8	8	0.04	8	8	0.03
r250.1c	250	30227	0.97	64	64	64	64	50.92	64	64	31.94
r250.5	250	14849	0.48	65	65	65	65	348.26	64	65	timeout
school1	385	19095	0.26	14	14	14	14	1720.33	14	14	2.62
school1_nsh	352	14612	0.24	14	14	14	14	1271.60	14	14	0.06
wap01a	2368	110871	0.04	41	43	-	47	timeout	41	45	timeout
wap02a	2464	111742	0.04	40	42	-	46	timeout	40	44	timeout
wap03a	4730	286722	0.03	40	47	-	57	timeout	40	51	timeout
wap04a	5231	294902	0.02	40	42	-	46	timeout	40	46	timeout
wap05a	905	43081	0.11	50	50	50	50	7.20	50	50	4.10
wap06a	947	43571	0.10	40	40	40	44	timeout	40	45	timeout
wap07a	1809	103368	0.06	40	41	-	47	timeout	40	45	timeout
wap08a	1870	104176	0.06	40	42	-	44	timeout	40	45	timeout

Table 3.3: (continued)

Instance	n	m	d	$\underline{X}$	$\overline{X}$	Exactcolors [HCS12] LB	UB	Time	Decision Diagram LB	UB	Time
will199GPIA	701	6772	0.03	7	7	7	7	5.96	6	7	timeout
zeroin.i.1	211	4100	0.19	49	49	49	49	0.10	49	49	0.67
zeroin.i.2	211	3541	0.16	30	30	30	30	0.09	30	30	0.32
zeroin.i.3	206	3540	0.17	30	30	30	30	0.06	30	30	0.30

Table 3.3: (continued)

– Exactcolors finds 23 better upper bounds compared to the 23 better upper bounds found by DD. For 91 instances they report the same upper bound.

Comparing this benchmark against the one done by van Hoeve, we find that the number of instances solved to optimality by our implementation is comparable. For finding good lower bounds however, our implementation performs worse. While van Hoeve reported more best known lower bounds found than Exactcolors, in our tests Exactcolors found several more best known lower bounds, while our implementation of DD did not scale up the same way with the faster machine we used, finding less lower bounds than Exactcolors. Van Hoeve also found 51 better lower bounds than Exactcolors, compared to our 25 better lower bounds. Part of the reason for that is the benefit Exactcolors had from the faster machine: in van Hoeve's test, Exactcolors did not report a lower bound for 50 instances, leading to DD finding 51 better lower bounds, while in our tests Exactcolors did not report a lower bound for only 24 instances, leading to our implementation of DD only finding 25 better lower bounds. This is not all of the reason however and we conclude that, for finding lower bounds, our implementation performs slightly worse with the fixed setting. Lastly, we do report better relative upper bounds compared to Exactcolors, van Hoeve in his tests had Exactcolors find 37 better upper bounds and DD find 13 better upper bounds. In our tests, this changes to Exactcolors finding 23 and DD also finding 23 better upper bounds. This is because Exactcolors also uses a coloring heuristic, Dsatur, to get an initial upper bound. Using the best of three coloring heuristics, including Dsatur, and employing the recolor technique, we get better bounds when only the heuristic was used to obtain an upper bound, likely because of the time limit.

3.2.5 Results On Open Instances

Finally we want to report the performance and quality of the bounds obtained by the algorithm on the set of open DIMACS instances in more detail, and explore further experiments, going beyond the time or node limit. First, we report in Table 3.4 the best lower respectively upper bounds obtained and give the fastest setting as well as report their time, all within the time limit of 1 hour and at most 1 million nodes, with the exception of instance `C2000.9`. Comparing with van Hoeve's evaluation, we report slightly worse obtained lower bounds by our method, though in most cases they differ by at most 1. We report better upper bounds than those van Hoeve did on all but the instance `DSJC1000.1`, owing to our improved coloring heuristics 'H' and 'N', though there are instances where our iterative method or exact decision diagram found better upper bounds as well.

Two interesting cases we want to point out are instance `DSJC500.9` and `r1000.1c`. On the first, van Hoeve computes an upper bound of 132 after running exact compilation with the Dsatur variable ordering for 10.7 hours. Also with exact compilation, we obtain an upper bound of 131 after just 15 minutes. We reason that this is because of the used clique in the variable ordering, but more so because of our changed integer programming formulation of (F) that is used in our implementation. This confirms our observation in Section 3.1.2 about CPLEX performing better when given more freedom, i.e. omitting the

Instance	n	m	d	$\underline{X}$	$\overline{X}$	LB	TTLB	Setting	UB	TTUB	Setting
1-Insertions_6	607	6337	0.03	4	7	3	0.07	B	7	0.00	H
3-Insertions_5	1406	9695	0.01	4	6	3	0.25	A	6	0.01	H
C2000.5	2000	999836	0.50	99	145	19	2902.43	L	203	23.63	H
C2000.9	2000	1799532	0.90	98	400	93	2.7 days	L	529	265.45	N
C4000.5	4000	4000268	0.50	107	259	19	1050.01	L	373	254.17	H
DSJC1000.1	1000	49629	0.10	10	20	6	0.53	L	27	0.06	H
DSJC1000.5	1000	249826	0.50	73	82	18	2887.43	L	115	2.62	N
DSJC1000.9	1000	449449	0.90	216	222	68	2407.90	K	288	2898.80	A
DSJC250.1	250	3218	0.10	6	8	5	0.05	L	10	0.00	H
DSJC250.5	250	15668	0.50	26	28	15	599.79	I	36	211.97	K
DSJC500.1	500	12458	0.10	9	12	6	3163.13	L	16	0.00	H
DSJC500.5	500	62624	0.50	43	47	17	2067.89	C	63	2373.56	C
DSJC500.9	500	112437	0.90	123	126	123	10.06	M	131	900.10	M
flat1000_76_0	1000	246708	0.49	72	81	17	787.09	L	114	2.29	H
latin_square_10	900	307350	0.76	90	97	90	17.43	I	130	2.02	H
queen16_16	256	6320	0.19	16	17	16	0.04	A	20	0.00	H
r1000.1c	1000	485090	0.97	96	98	96	62.00	F	99	585.84	F
wap01a	2368	110871	0.04	41	43	41	11.93	L	45	0.12	H
wap02a	2464	111742	0.04	40	42	40	11.82	L	44	3.40	L
wap03a	4730	286722	0.03	40	47	40	23.29	C	50	20.17	N
wap04a	5231	294902	0.02	40	42	40	41.94	K	46	0.41	H
wap07a	1809	103368	0.06	40	41	40	7.89	L	44	1.56	N
wap08a	1870	104176	0.06	40	42	40	8.39	L	43	0.10	H

Table 3.4: Best performing setting to each of the open DIMACS instances, we list the number of node (n) and edges (M), the edge density (d) and the best known lower ($\underline{X}$) respectively upper bounds ($\overline{X}$). We further give the lower bound (LB), time to lower bound (TTB), upper bound (UB), time to upper bound (TTUB), and the setting with the best time.

variable bounds. For a similar reason, we observe the great performance of setting 'F' on instance `r1000.1c`, reporting the best known lower bound after 62 seconds and finding an upper bound just 1 worse of the best known in less than 10 minutes.

Based on this great performance, we let setting 'F' run on `r1000.1c` without a time limit and tell CPLEX to emphasize improving the best lower bound. After 3.8 days, CPLEX reports a new lower bound of 98, concluding that the chromatic number of instance `r1000.1c` is 98. It still takes a while longer till CPLEX itself also finds the upper bound of 98 and finishes. This time can be lowered significantly if the pre-processing procedure mentioned in Section 3.1.2 is employed, reducing the graph with 1000 vertices and 485090 edges to one with 686 vertices and 227525 edges. Using the Dsatur variable ordering with a clique and telling CPLEX to emphasize improving the lower bound, the exact compilation on this reduced graph terminates with the chromatic number after only 5 hours and 48 minutes.

On instance `DSJC500.9`, the lower bound of 123 is found very quickly. This graph can not be reduced by the pre-processing procedure and with the standard settings of CPLEX, neither an improved lower or upper bound is found after 1.6 days (though it did find an upper bound of 129), using exact compilation with the Dsatur ordering and a clique. But if we again tell CPLEX to emphasize improving the lower bound, we are able to report a new lower bound of 124 after 5 hours and 45 minutes.

We were unable to reproduce the improved lower bound for instance `C2000.9` reported by van Hoeve with our implementation of the iterative refinement procedure.

Chapter 4

Conclusion

We introduced the foundations of graph coloring and gave an overview of previous effort on heuristic and exact methods to solve this problem, and we presented the fractional chromatic number as well as their integrality gap to the chromatic number. Based on van Hoeve's paper [Hoe21], we presented decision diagrams and how they could be used to compute the chromatic number. Specifically the exact compilation, building one large decision diagram representing all independent sets in a graph and computing $\chi(G)$ by solving a modified network flow problem on it. Furthermore, we saw the iterative approach beginning with a relaxed decision diagram containing possible conflicts and separating those conflicts until we obtained a sufficiently good enough approximation to the solution set to determine the chromatic number.

Adding to van Hoeve's effort, we showed that, while the integer program (F) computes the chromatic number, the linear relaxation (F') computes the fractional chromatic number of the graph. This also showed that the integrality gap between the two models is $\mathcal{O}(\log n)$. Second, we adapted a method from Neumaier [NS04] to obtain safe bounds from the linear program by post-processing the dual solution, avoiding wrong computation arising from floating point errors.

As part of this thesis we implemented our own version of van Hoeve's [Hoe21] new approach to graph coloring. We repeated his experimental results, first showing the benefit of the iterative refinement compared to the exact compilation and confirming the theoretical observation that in some cases where an exponential sized exact decision diagram is needed, a polynomial sized relaxed diagram is enough. We evaluated his proposed settings of the algorithm and similarly found that 54 instances were solved to optimality, with 49 solved in under a minute and 36 within a second.

We proposed further ideas and settings potentially leading to a performance improvement, and evaluated the best of these against the column generation implementation from Held et al. [HCS12]. We reported similar results for both programs, with our implementation on average being outperformed by their Exactcolors algorithm.

Turning to look at the set of open instances, we observed good performance of the exact compilation on two instances with a large edge density. We further investigated these and changed the settings of CPLEX respectively reduced the graph in a pre-processing procedure. In these experiments, we improved the lower bound of the instance `DSJC500.9` and computed the chromatic number of the previously open instance `r1000.1c`.

4.1 Discussion

In Section 2.5 we considered the linear program and its possible floating point errors. We presented a method to still obtain safe bounds via a solution to the dual linear program and post-processing it. As it uses the dual, this method does not transfer to integer programs, yet the literature we mentioned also displays floating point errors occurring on instances of that type. While Neumaier's method required little effort in terms of changing

our code to be used, we do not know of such a nice method for obtaining safe bounds to integer programs. To ensure that one computes the correct solution, one method is to embed the building of a certificate into the solving process, which can then be used to certify the correctness of a solution [CGS17]. Alternatively, the integer program is solved in exact arithmetic to avoid any rounding errors. One implementation of such a mixed integer programming solver is (an extension of) SCIP [EG21].

We did not consider either of these two methods in our implementation, therefore most of our results and those of van Hoeve depend on the assumption that no floating point error occurred in CPLEX when solving an integer program. Since there are no large differences in coefficients and all of them can be stored exactly, we do not expect any floating point errors, but this should be kept in mind.

Observing the performance of our implementation, especially on the set of open instances but also the explicit comparison of the two methods in Section 3.2.1, indicates the following: exact compilation performs better and has an advantage over the iterative refinement if the edge density is large. We reason that this is because the exact decision diagram then has a manageable size, as more edges mean less independent sets and CPLEX is in these cases able to reduce the problem to a much smaller size. Instance `r1000.1c` with exact compilation and setting 'F' produces a decision diagram with 915059 nodes and 940685 arcs, which is reduced in CPLEX to a problem with only 11475 rows, 36131 columns and 100093 nonzeros. On the other hand, we observe the performance benefit in Section 3.2.1 from the iterative refinement to be most significant on instances with a smaller edge density, as more independent sets also means less potential conflicts to separate. Of the 15 instances that were solved by the relaxed decision diagrams but not the exact ones, all had an edge density of at most 33%, 12 of them had one of at most 11%.

4.2 Outlook

In further literature discussing decision diagrams [Tja18; Ber+14], so called *long arcs* are utilized to reduce the number of nodes. These are arcs that skip a layer, possibly avoiding having to create a node in said layer. In general, long arcs may represent any set of assignments to the decision variables between layers of the beginning and end of the arc, indicated by a label of that arc. Usually they are employed to remove nodes from the diagram with only an outgoing 0-arc. These long arcs could theoretically be utilized also for decision diagrams in this context, and potentially reduce the size of the diagram needed, both for exact and relaxed ones. This would require a certain level of modifications to almost all considered algorithms, and is therefore only brought up as a possible extension to this approach.

Further, the decision diagrams appearing in the iterative refinement in each iteration are not enormously different from that in the previous iteration. Possibly some information of a solution on the previous diagram could be used to speed up the solving process of the linear or integer program on the next decision diagram. This is not immediately possible, since both new nodes and new arcs are introduced when separating a conflict and thus both the primal and dual solutions of the old program become infeasible in the next linear program. It is not known to us whether any of the information of a previous solution could be used in this context but this could be a further direction to explore.

Appendix

A.1 Explicit Dual

The linear relaxation of the integer program (F) was

$$
\begin{aligned}
(F') = \min \quad & \sum_{a \in \delta^+(r)} y_a && && \text{(A.1)}\\
\text{s.t.} \quad & \sum_{a=(u,v) \mid L(u)=j, \ell(a)=1} y_a \geq 1 && \forall j \in V && \text{(A.2)}\\
& \sum_{a \in \delta^-(u)} y_a - \sum_{a \in \delta^+(u)} y_a = 0 && \forall u \in N \setminus \{r,t\} && \text{(A.3)}\\
& 0 \leq y_a \leq n && \forall a \in A && \text{(A.4)}
\end{aligned}
$$

We simplified the formulation by observing that we could omit the variable bounds (A.4): an optimal solution to the modified model also satisfied the bound constraints since otherwise we could reduce the flow along a path with superfluous flow and obtain a solution with smaller objective value, contradicting the optimality of the solution. Thus we obtain the following linear relaxation which is what we use internally in our program and to compute safe bounds with its dual.

$$
\begin{aligned}
\min \quad & \sum_{a \in \delta^+(r)} y_a && && \text{(A.5)}\\
\text{s.t.} \quad & \sum_{a=(u,v) \mid L(u)=j, \ell(a)=1} y_a \geq 1 && \forall j \in V && \text{(A.6)}\\
& \sum_{a \in \delta^-(u)} y_a - \sum_{a \in \delta^+(u)} y_a = 0 && \forall u \in N \setminus \{r,t\} && \text{(A.7)}\\
& 0 \leq y_a && \forall a \in A && \text{(A.8)}
\end{aligned}
$$

As the stated linear program is very close to a normal flow problem, so is its corresponding dual linear program. It has variables z_j for each vertex $j \in V$ coming from constraint (A.6) and u_v for each node $u \in N \setminus \{r,t\}$ coming from constraints (A.7). To avoid a number of case distinctions, we added u_r and u_t as dummy variables.

$$
\begin{aligned}
\max. \quad & \sum_{j \in V} z_j && \\
\text{s.t.} \quad & l(a) \cdot z_j - u_v + u_w \leq 0 && \forall a = (v,w) \in A, L(a) = j\\
& u_r = -1 && \\
& u_t = 0 && \\
& z_j \leq 0 && \forall a \in A
\end{aligned}
$$

A.2 DFMAX Benchmarks

To allow for comparability for the results obtained with different machines, a benchmark program (dfmax) and benchmark instances are available at http://mat.gsia.cmu.edu/COLOR04/ [Tri02]. The idea is for computing times obtained on different machines to be scaled according to their relative performance on this benchmark program and the given instances.

All benchmarks of our algorithm and that of Held et al. [HCS12] were done on a machine with an Intel Xeon X5690@3.47GHZ CPU running CentOS Linux 7. For instances `r300.5.b`, `r400.5.b` and `r500.5.b` we report a runtime of 0.23s, 1.47s and 5.58s.

A.3 Further Experimental Data

Comparing The Lexicographic And The Max-Connected-Degree Orderings

Table A.1 gives a comparison of how the iterative refinement procedure performs in terms of lower and upper bounds found, time to solve an instance to optimality, and size of the required decision diagram, depending on which variable ordering is used. We see a clear advantage of the Max-Connected-Degree ordering, outperforming the lexicographic ordering on all but the instance `queen9_9` and often computing the chromatic number with an enormously smaller decision diagram. For example, `jean` can be solved with a diagram using 292 nodes for the Max-Connected-Degree ordering, while the same instance with lexicographic ordering exceed the time limit after having built a decision diagram with 42608 nodes.

Detailed Performance Of The Original Settings

In Table A.2 we state the best lower and upper bound found by any of the original settings formulated in Section 3.2.2 and list which settings were able to obtain that bound. If the upper bound was found by the coloring heuristic, we only list 'H', as the iterative refinement uses it to get an initial bound and does not produce a better bound through the primal heuristic.

Detailed Comparison Of The Improved Setting

Table A.3 gives a comparison of the algorithm with the two settings 'C' and 'L' in terms of lower and upper bounds found, time to solve an instance to optimality, and size of the required decision diagram. We mentioned before the speed benefit We see a clear advantage of the Max-Connected-Degree ordering, outperforming the lexicographic ordering on all but the instance `queen9_9` and often computing the chromatic number with an enormously smaller decision diagram. For example, `jean` can be solved with a diagram using 292 nodes for the Max-Connected-Degree ordering, while the same instance with lexicographic ordering exceed the time limit after having built a decision diagram with 42608 nodes. For each instance we list the number of vertices (n) the number (m) of edges, and report for each setting the obtained lower bound (LB), obtained upper bound (UB), the solving time (in seconds), and the size of the decision diagram needed. In the last column (R/E) we list the ration of the sizes of the decision diagram sizes. The time limit was 3600s and at most 1 million nodes were allowed.

				Max-Connected-Degree ordering				Lexicographic ordering				
Instance	n	m	d	LB	UB	Size	Time	LB	UB	Size	Time	R/E
1-FullIns_3	30	100	0.23	4	4	245	0.02	4	4	607	0.18	0.40
2-FullIns_3	52	201	0.15	5	5	1257	0.18	5	5	6919	24.39	0.18
DSJC125.9	125	6961	0.90	44	44	9417	42.55	44	44	9990	50.06	0.94
miles1500	128	5198	0.64	73	73	2662	0.89	73	73	13631	272.49	0.19
mulsol.i.1	197	3925	0.20	49	49	1109	0.60	49	49	14870	648.26	0.07
mulsol.i.2	188	3885	0.22	31	31	888	0.25	31	31	25200	897.50	0.03
mulsol.i.3	184	3916	0.23	31	31	884	0.24	31	31	23189	696.52	0.03
mulsol.i.4	185	3946	0.23	31	31	885	0.25	31	31	23601	743.50	0.03
mulsol.i.5	186	3973	0.23	31	31	886	0.24	31	31	33048	3194.05	0.02
myciel3	11	20	0.36	4	4	59	0.02	4	4	66	0.02	0.89
myciel4	23	71	0.28	5	5	453	5.74	5	5	648	3.70	0.69
qg.order30	900	26100	0.06	30	30	1337	1.09	30	30	1337	1.15	1.00
queen5_5	25	160	0.53	5	5	194	0.00	5	5	228	0.00	0.85
queen6_6	36	290	0.46	7	7	1757	0.99	7	7	1646	0.46	1.06
queen7_7	49	476	0.40	7	7	3381	2.01	7	7	3102	0.90	1.08
queen8_8	64	728	0.36	9	9	27917	334.29	9	9	34157	494.14	0.81
queen9_9	81	1056	0.33	10	10	65978	2825.37	10	10	50432	664.93	1.30
r125.1c	125	7501	0.97	46	46	3547	1.42	46	46	4026	3.86	0.88
r250.1c	250	30227	0.97	64	64	17562	97.97	64	64	22740	211.33	0.77
zeroin.i.1	211	4100	0.19	49	49	1018	0.51	49	49	5925	89.63	0.17
zeroin.i.2	211	3541	0.16	30	30	888	0.23	30	30	18350	1433.83	0.04
zeroin.i.3	206	3540	0.17	30	30	883	0.23	30	30	17412	983.78	0.05
3-FullIns_3	80	346	0.11	6	6	9143	14.98	5	6	44855	timeout	0.20
4-FullIns_3	114	541	0.08	7	7	20556	140.63	6	7	59187	timeout	0.34
5-FullIns_3	154	792	0.07	8	8	55704	2187.85	7	8	76599	timeout	0.72
DSJR500.1	500	3555	0.03	12	12	624	0.05	2	12	45642	timeout	0.01
anna	138	493	0.05	11	11	350	0.01	3	11	39316	timeout	0.00
david	87	406	0.11	11	11	243	0.00	4	11	63106	timeout	0.00
fpsol2.i.1	496	11654	0.09	65	65	4407	13.15	28	65	54282	timeout	0.08
fpsol2.i.2	451	8691	0.09	30	30	1021	0.58	21	30	51392	timeout	0.01
fpsol2.i.3	425	8688	0.10	30	30	995	0.55	24	30	37980	timeout	0.02
games120	120	638	0.09	9	9	22379	14.77	4	9	86288	timeout	0.25
huck	74	301	0.11	11	11	811	0.14	5	11	58201	timeout	0.01
inithx.i.1	864	18707	0.05	54	54	5146	17.63	23	54	44150	timeout	0.11
inithx.i.2	645	13979	0.07	31	31	15509	95.48	24	31	64854	timeout	0.23
inithx.i.3	621	13969	0.07	31	31	15848	79.12	23	31	51961	timeout	0.30
jean	80	254	0.08	10	10	292	0.00	5	10	42608	timeout	0.00
le450_25a	450	8260	0.08	25	25	1209	0.69	3	25	53969	timeout	0.02
le450_25b	450	8263	0.08	25	25	1056	0.55	3	25	65411	timeout	0.01
miles1000	128	3216	0.40	42	42	5503	2.38	15	42	50939	timeout	0.10
miles250	128	387	0.05	8	8	293	0.01	3	8	53520	timeout	0.00
miles500	128	1170	0.14	20	20	345	0.04	5	20	64272	timeout	0.00
miles750	128	2113	0.26	31	31	788	0.15	8	31	56594	timeout	0.01
r1000.1	1000	14378	0.03	20	20	1234	0.62	3	20	54706	timeout	0.02
r125.1	125	209	0.03	5	5	339	0.01	2	5	59365	timeout	0.00
r125.5	125	3838	0.50	36	36	16907	267.88	24	36	40237	timeout	0.42
r250.1	250	867	0.03	8	8	3312	1.64	2	8	48033	timeout	0.06
school1	385	19095	0.26	14	14	2172	1.70	12	18	152441	timeout	0.01
school1_nsh	352	14612	0.24	14	14	6954	17.57	12	15	156941	timeout	0.04
wap05a	905	43081	0.11	50	50	17088	44.11	12	50	67302	timeout	0.25

Table A.1: Comparison of the performance of the iterative refinement procedure using the Max-Connected-Degree ordering and the lexicographic ordering on the DIMACS instances that were solved to optimality by either method. For each instance we list the number of vertices (n) the number (m) of edges, and report for each setting the obtained lower bound (LB), obtained upper bound (UB), the solving time (in seconds), and the size of the decision diagram needed. In the last column (R/E) we list the ration of the sizes of the decision diagram sizes. The time limit was 3600s and at most 1 million nodes were allowed.

						Decision Diagram				
Instance	n	m	d	$\underline{X}$	$\overline{X}$	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	ABCD	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	ABCDG	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	CDE	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	ABCDEG	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	7	ABCD	9	H	
abb313GPIA	1557	53356	0.04	9	9	8	AB	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	3	ABCD	5	H	
ash958GPIA	1916	12506	0.01	4	4	4	B	5	H	
C2000.5	2000	999836	0.50	99	145	18	C	203	H	
C2000.9	2000	1799532	0.90	98	400	66	BC	530	H	
C4000.5	4000	4000268	0.50	107	259	19	C	373	H	
david	87	406	0.11	11	11	11	BCDFG	11	H	*
DSJC1000.1	1000	49629	0.10	10	20	6	BCD	27	H	
DSJC1000.5	1000	249826	0.50	73	82	17	C	116	H	
DSJC1000.9	1000	449449	0.90	216	222	66	C	288	A	
DSJC125.1	125	736	0.09	5	5	5	BCD	6	H	
DSJC125.5	125	3891	0.50	17	17	16	EFG	18	EF	
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	15	CD	36	ABCD	
DSJC250.9	250	27897	0.90	72	72	72	EFG	72	EFG	*
DSJC500.1	500	12458	0.10	9	12	5	BCD	16	H	
DSJC500.5	500	62624	0.50	43	47	17	C	63	CD	
DSJC500.9	500	112437	0.90	123	126	123	EFG	131	EFG	
DSJR500.1	500	3555	0.03	12	12	12	BCD	12	H	*
DSJR500.1c	500	121275	0.97	85	85	85	EFG	85	EFG	*
DSJR500.5	500	58862	0.47	122	122	111	B	126	D	
flat1000_50_0	1000	245000	0.49	50	50	17	C	113	H	
flat1000_60_0	1000	245830	0.49	60	60	16	C	114	H	
flat1000_76_0	1000	246708	0.49	72	81	17	C	114	H	
flat300_20_0	300	21375	0.48	20	20	15	C	38	D	
flat300_26_0	300	21633	0.48	26	26	15	C	40	BCD	
flat300_28_0	300	21695	0.48	28	28	15	C	39	D	
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	

Table A.2: Best lower and upper bounds obtained by any of the settings on all DIMACS instances. Next to the data of the graphs, we list the best known lower ($\underline{X}$) and upper bounds ($\overline{X}$). Fully solved instances are marked with an asterisk.

						Decision Diagram				
Instance	n	m	d	$\underline{X}$	$\overline{X}$	LB	Setting	UB	Setting	
le450_15a	450	8168	0.08	15	15	15	BCD	17	H	
le450_15b	450	8169	0.08	15	15	15	BCD	16	H	
le450_15c	450	16680	0.17	15	15	15	BCD	24	H	
le450_15d	450	16750	0.17	15	15	15	BCD	24	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	7	H	
le450_5c	450	9803	0.10	5	5	5	BCD	8	D	
le450_5d	450	9757	0.10	5	5	5	BCD	9	C	
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	ABCDG	7	H	
myciel7	191	2360	0.13	8	8	4	ABCD	8	H	
qg.order100	10000	990000	0.02	100	100	100	ABCD	102	H	
qg.order30	900	26100	0.06	30	30	30	ABCD	30	H	*
qg.order40	1600	62400	0.05	40	40	40	ABCD	41	H	
qg.order60	3600	212400	0.03	60	60	60	ABCD	62	H	
queen10_10	100	1470	0.30	11	11	10	ABC	12	D	
queen11_11	121	1980	0.27	11	11	11	ABC	14	H	
queen12_12	144	2596	0.25	12	12	12	AB	15	H	
queen13_13	169	3328	0.23	13	13	13	AB	16	H	
queen14_14	196	4186	0.22	14	14	14	AB	17	C	
queen15_15	225	5180	0.21	15	15	15	AB	19	H	
queen16_16	256	6320	0.19	16	17	16	AB	20	H	
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	13	H	
queen8_8	64	728	0.36	9	9	9	ABCDEFG	9	ACDEFG	*
queen9_9	81	1056	0.33	10	10	10	AC	10	AC	*
r1000.1	1000	14378	0.03	20	20	20	BCD	20	H	*
r1000.1c	1000	485090	0.97	96	98	96	F	99	F	
r1000.5	1000	238267	0.48	234	234	211	C	245	D	
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	H	*
r250.1	250	867	0.03	8	8	8	BCD	8	H	*
r250.1c	250	30227	0.97	64	64	64	ABCDEFG	64	H	*
r250.5	250	14849	0.48	65	65	65	BCDEFG	65	BFG	*
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	BC	45	H	
wap02a	2464	111742	0.04	40	42	40	BC	45	H	
wap03a	4730	286722	0.03	40	47	40	BCD	51	H	
wap04a	5231	294902	0.02	40	42	40	BCD	46	H	
wap05a	905	43081	0.11	50	50	50	BCD	50	H	*
wap06a	947	43571	0.10	40	40	40	BCD	42	H	
wap07a	1809	103368	0.06	40	41	38	B	45	H	
wap08a	1870	104176	0.06	40	42	38	B	43	H	

Table A.2: (continued)

						Decision Diagram				
Instance	n	m	d	$\underline{X}$	$\overline{X}$	LB	Setting	UB	Setting	
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 A.2: (continued)

				Fixing vertices of a clique				Standard setting				
Instance	n	m	d	LB	UB	Size	Time	LB	UB	Size	Time	R/E
1-FullIns_3	30	100	0.23	4	4	245	0.01	4	4	245	0.02	1.00
2-FullIns_3	52	201	0.15	5	5	1257	0.19	5	5	1257	0.18	1.00
3-FullIns_3	80	346	0.11	6	6	9143	15.61	6	6	9143	14.98	1.00
4-FullIns_3	114	541	0.08	7	7	20556	142.50	7	7	20556	140.63	1.00
5-FullIns_3	154	792	0.07	8	8	55704	1936.40	8	8	55704	2187.85	1.00
DSJC125.9	125	6961	0.90	44	44	9071	31.50	44	44	9417	42.55	0.96
DSJR500.1	500	3555	0.03	12	12	624	0.11	12	12	624	0.05	1.00
anna	138	493	0.05	11	11	300	0.01	11	11	350	0.01	0.85
david	87	406	0.11	11	11	243	0.00	11	11	243	0.00	1.00
fpsol2.i.1	496	11654	0.09	65	65	1638	2.61	65	65	4407	13.15	0.37
fpsol2.i.2	451	8691	0.09	30	30	768	0.30	30	30	1021	0.58	0.75
fpsol2.i.3	425	8688	0.10	30	30	742	0.27	30	30	995	0.55	0.74
games120	120	638	0.09	9	9	336	0.01	9	9	22379	14.77	0.01
huck	74	301	0.11	11	11	163	0.00	11	11	811	0.14	0.20
inithx.i.1	864	18707	0.05	54	54	1985	3.88	54	54	5146	17.63	0.38
inithx.i.2	645	13979	0.07	31	31	1086	0.84	31	31	15509	95.48	0.07
inithx.i.3	621	13969	0.07	31	31	1062	0.71	31	31	15848	79.12	0.06
jean	80	254	0.08	10	10	197	0.00	10	10	292	0.00	0.67
le450_25a	450	8260	0.08	25	25	754	0.36	25	25	1209	0.69	0.62
le450_25b	450	8263	0.08	25	25	755	0.36	25	25	1056	0.55	0.71
miles1000	128	3216	0.40	42	42	970	0.20	42	42	5503	2.38	0.17
miles1500	128	5198	0.64	73	73	2079	0.59	73	73	2662	0.89	0.78
miles250	128	387	0.05	8	8	293	0.01	8	8	293	0.01	1.00
miles500	128	1170	0.14	20	20	328	0.03	20	20	345	0.04	0.95
miles750	128	2113	0.26	31	31	600	0.10	31	31	788	0.15	0.76
mulsol.i.1	197	3925	0.20	49	49	1071	0.58	49	49	1109	0.60	0.96
mulsol.i.2	188	3885	0.22	31	31	601	0.17	31	31	888	0.25	0.67
mulsol.i.3	184	3916	0.23	31	31	544	0.15	31	31	884	0.24	0.61
mulsol.i.4	185	3946	0.23	31	31	598	0.16	31	31	885	0.25	0.67
mulsol.i.5	186	3973	0.23	31	31	599	0.16	31	31	886	0.24	0.67
myciel3	11	20	0.36	4	4	59	0.03	4	4	59	0.02	1.00
myciel4	23	71	0.28	5	5	453	6.27	5	5	453	5.74	1.00
qg.order30	900	26100	0.06	30	30	1337	1.39	30	30	1337	1.09	1.00
queen5_5	25	160	0.53	5	5	206	0.00	5	5	194	0.00	1.06
queen6_6	36	290	0.46	7	7	1912	1.17	7	7	1757	0.99	1.08
queen7_7	49	476	0.40	7	7	3865	1.68	7	7	3381	2.01	1.14
queen8_8	64	728	0.36	9	9	34081	681.08	9	9	27917	334.29	1.22
r1000.1	1000	14378	0.03	20	20	1200	0.78	20	20	1234	0.62	0.97
r125.1	125	209	0.03	5	5	289	0.01	5	5	339	0.01	0.85
r125.1c	125	7501	0.97	46	46	1005	0.14	46	46	3547	1.42	0.28
r125.5	125	3838	0.50	36	36	12959	13.30	36	36	16907	267.88	0.76
r250.1	250	867	0.03	8	8	459	0.03	8	8	3312	1.64	0.13
r250.1c	250	30227	0.97	64	64	15349	31.94	64	64	17562	97.97	0.87
school1	385	19095	0.26	14	14	2707	2.62	14	14	2172	1.70	1.24
school1_nsh	352	14612	0.24	14	14	461	0.06	14	14	6954	17.57	0.06
wap05a	905	43081	0.11	50	50	1959	4.10	50	50	17088	44.11	0.11
zeroin.i.1	211	4100	0.19	49	49	1190	0.67	49	49	1018	0.51	1.16
zeroin.i.2	211	3541	0.16	30	30	847	0.32	30	30	888	0.23	0.95
zeroin.i.3	206	3540	0.17	30	30	836	0.30	30	30	883	0.23	0.94
homer	561	1629	0.01	13	13	664	0.16	10	13	47284	timeout	$\leq$0.01
queen9_9	81	1056	0.33	10	11	65632	timeout	10	10	65978	2825.37	$\geq$0.99

Table A.3: Comparison of the performance of the iterative refinement procedure using the Max-Connected-Degree orderng with and without a clique to fix the first vertices on the DIMACS instances that were solved to optimality by either method.

Bibliography

[Ahm12] Shamim Ahmed. "Applications of graph coloring in modern computer science". In: *International Journal of Computer and Information Technology* 3.2 (2012), pp. 1–7 (cit. on p. 2).

[BB04] Nicolas Barnier and Pascal Brisset. "Graph coloring for air traffic flow management". In: *Annals of operations research* 130.1 (2004), pp. 163–178 (cit. on p. 2).

[Ber+12a] David Bergman et al. "Variable Ordering for the Application of BDDs to the Maximum Independent Set Problem". In: *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*. Ed. by Nicolas Beldiceanu, Narendra Jussien, and Éric Pinson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 34–49. ISBN: 978-3-642-29828-8 (cit. on p. 12).

[Ber+12b] David Bergman et al. "Variable ordering for the application of BDDs to the maximum independent set problem". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2012, pp. 34–49 (cit. on pp. 29, 32).

[Ber+14] David Bergman et al. "Optimization bounds from binary decision diagrams". In: *INFORMS Journal on Computing* 26.2 (2014), pp. 253–268 (cit. on pp. 29, 48).

[Bré79] Daniel Brélaz. "New Methods to Color the Vertices of a Graph". In: *Commun. ACM* 22.4 (April 1979), pp. 251–256. ISSN: 0001-0782. DOI: 10.1145/359094.359101. URL: https://doi.org/10.1145/359094.359101 (cit. on pp. 4, 5).

[Bry86] Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8 (1986), pp. 677–691. DOI: 10.1109/TC.1986.1676819 (cit. on p. 12).

[CGS17] Kevin KH Cheung, Ambros Gleixner, and Daniel E Steffy. "Verifying integer programming results". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2017, pp. 148–160 (cit. on p. 48).

[Coo+13] William Cook et al. "A hybrid branch-and-bound approach for exact rational mixed-integer programming". In: *Mathematical Programming Computation* 5.3 (2013), pp. 305–344 (cit. on p. 26).

[CS79] Claude A Christen and Stanley M Selkow. "Some perfect coloring properties of graphs". In: *Journal of Combinatorial Theory, Series B* 27.1 (1979), pp. 49–59. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(79)90067-4. URL: https://www.sciencedirect.com/science/article/pii/0095895679900674 (cit. on p. 4).

[Dvo18] Zdenek Dvorťak. *Fractional coloring*. March 24, 2018. URL: https://iuuk.mff.cuni.cz/~rakdver/barevnost/fractional.pdf (visited on July 7, 2021) (cit. on p. 8).

[EG21] Leon Eifler and Ambros Gleixner. “A Computational Status Update for Exact Rational Mixed Integer Programming”. In: *arXiv preprint arXiv:2101.09141* (2021) (cit. on p. 48).

[FGT17] Fabio Furini, Virginie Gabrel, and Ian-Christopher Ternier. “An Improved DSATUR-Based Branch-and-Bound Algorithm for the Vertex Coloring Problem”. In: *Networks* 69.1 (2017), pp. 124–141 (cit. on p. 32).

[HCS12] Stephan Held, William Cook, and Edward C Sewell. “Maximum-weight stable sets and safe lower bounds for graph coloring”. In: *Mathematical Programming Computation* 4.4 (2012), pp. 363–381 (cit. on pp. 6, 8, 32, 34, 42–45, 47, 50).

[Hig02] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002 (cit. on p. 26).

[HK19] Emmanuel Hebrard and George Katsirelos. “A hybrid approach for exact coloring of massive graphs”. In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2019, pp. 374–390 (cit. on p. 33).

[Hoe20] Willem-Jan van Hoeve. “Graph Coloring Lower Bounds from Decision Diagrams”. In: *Integer Programming and Combinatorial Optimization*. Ed. by Daniel Bienstock and Giacomo Zambelli. Cham: Springer International Publishing, 2020, pp. 405–418. ISBN: 978-3-030-45771-6 (cit. on pp. 7, 11).

[Hoe21] Willem-Jan van Hoeve. “Graph coloring with decision diagrams”. In: *Mathematical Programming* (2021), pp. 1–44 (cit. on pp. iii, iv, 7, 11–13, 15, 18, 19, 21, 29, 31, 34, 35, 47).

[Jan+01] R. Janczewski et al. “The smallest hard-to-color graph for algorithm DSATUR”. In: *Discrete Mathematics* 236.1 (2001). Graph Theory, pp. 151–165. ISSN: 0012-365X. DOI: https://doi.org/10.1016/S0012-365X(00)00439-8. URL: https://www.sciencedirect.com/science/article/pii/S0012365X00004398 (cit. on p. 5).

[Jan04] Christian Jansson. “Rigorous lower and upper bounds in linear programming”. In: *SIAM Journal on Optimization* 14.3 (2004), pp. 914–935 (cit. on p. 26).

[JT96] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*. Vol. 26. American Mathematical Soc., 1996 (cit. on pp. 34, 38).

[Kar72] Richard M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9. URL: https://doi.org/10.1007/978-1-4684-2001-2_9 (cit. on pp. 1, 2).

[Klo14] Ed Klotz. *Identification, Assessment and Correction of Ill-Conditioning and Numerical Instability in Linear and Integer Programs*. 2014. URL: http://www.optimizationdirect.com/data/180805_ODIllCond.pdf (visited on July 15, 2021) (cit. on p. 26).

[LMM04] Corinne Lucet, Florence Mendes, and Aziz Moukrim. “Pre-processing and linear-decomposition algorithm to solve the k-colorability problem”. In: *International Workshop on Experimental and Efficient Algorithms*. Springer. 2004, pp. 315–325 (cit. on p. 34).

[Lov75] László Lovász. "On the ratio of optimal integral and fractional covers". In: *Discrete mathematics* 13.4 (1975), pp. 383–390 (cit. on p. 9).

[LPU95] Michael Larsen, James Propp, and Daniel Ullman. "The fractional chromatic number of Mycielski's graphs". In: *Journal of Graph Theory* 19.3 (1995), pp. 411–416 (cit. on p. 9).

[Ly19] Kevin Ly. *Graph coloring*. September 20, 2019. URL: http://algorithm-interest-group.me/assets/slides/dsatur.pdf (visited on June 28, 2021) (cit. on p. 2).

[MT96] Anuj Mehrotra and Michael A Trick. "A column generation approach for graph coloring". In: *informs Journal on Computing* 8.4 (1996), pp. 344–354 (cit. on pp. 6, 7, 42).

[Myc55] Jan Mycielski. "Sur le coloriage des graphes". In: *Colloq. Math.* Vol. 3. 161-162. 1955, p. 9 (cit. on p. 8).

[NS04] Arnold Neumaier and Oleg Shcherbina. "Safe bounds in linear and mixed-integer linear programming". In: *Mathematical Programming* 99.2 (2004), pp. 283–296 (cit. on pp. 26, 47).

[Owl] Owlapps. *Graph coloring*. URL: http://www.owlapps.net/owlapps_apps/articles?id=426743&lang=en (visited on June 28, 2021) (cit. on p. 2).

[PM04] Panos M Pardalos and Athanasios Migdalas. "A note on the complexity of longest path problems related to graph coloring". In: *Applied mathematics letters* 17.1 (2004), pp. 13–15 (cit. on p. 31).

[RA14] Ryan A Rossi and Nesreen K Ahmed. "Coloring large complex networks". In: *Social Network Analysis and Mining* 4.1 (2014), p. 228 (cit. on p. 33).

[San12] Pablo San Segundo. "A new DSATUR-based algorithm for exact vertex coloring". In: *Computers & Operations Research* 39.7 (2012), pp. 1724–1733 (cit. on p. 32).

[Sew96] EC Sewell. "An improved algorithm for exact graph coloring". In: *DIMACS series in discrete mathematics and theoretical computer science* 26 (1996), pp. 359–373 (cit. on p. 32).

[SU11] Edward R Scheinerman and Daniel H Ullman. *Fractional graph theory: a rational approach to the theory of graphs*. Courier Corporation, 2011 (cit. on pp. 7, 9).

[Tan20] Ole Tange. *GNU Parallel 20201222 ('Vaccine')*. GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them. December 2020. DOI: 10.5281/zenodo.4381888. URL: https://doi.org/10.5281/zenodo.4381888.

[Tja18] Christian Tjandraatmadja. "Decision Diagram Relaxations for Integer Programming". In: (2018) (cit. on pp. 11, 48).

[Tri02] Michael A Trick. *Computational symposium: graph coloring and its generalization*. 2002. URL: https://mat.tepper.cmu.edu/COLOR04/ (visited on July 23, 2021) (cit. on p. 50).

[Tur88] Jonathan S Turner. "Almost all k-colorable graphs are easy to color". In: *Journal of algorithms* 9.1 (1988), pp. 63–82 (cit. on p. 4).

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Vol. 4. SIAM, 2000 (cit. on p. 29).

[Zak06] Manouchehr Zaker. "Results on the Grundy chromatic number of graphs". In: *Discrete Mathematics* 306.23 (2006). International Workshop on Combinatorics, Linear Algebra, and Graph Coloring, pp. 3166–3173. ISSN: 0012-365X. DOI: https://doi.org/10.1016/j.disc.2005.06.044. URL: https://www.sciencedirect.com/science/article/pii/S0012365X06004122 (cit. on pp. 4, 8).

[Zuc06] David Zuckerman. "Linear degree extractors and the inapproximability of max clique and chromatic number". In: *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. 2006, pp. 681–690 (cit. on p. 1).