University of Twente
MSc Applied Mathematics

A polyhedral study of a constrained flow problem on decision diagrams

Author:
T.E. Hugen

Supervisor:
Dr. M. Walter

Graduation committee:
Prof. d.r. M.J. Uetz
Dr. J.M. Meylahn
Dr. M. Walter

December 17, 2023

Discrete Mathematics and Mathematical Programming (DMMP),
Mathematics of Operations Research (MOR),
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS),
University of Twente

Abstract

The graph coloring problem is an NP-hard combinatorial optimization problem that has many applications. In a recent publication [W.-J. van Hoeve, "Graph coloring with decision diagrams," *Mathematical Programming*, vol. 192, no. 1–2, pp. 631–674, May 2021], an exact algorithm employing decision diagrams was introduced for the graph coloring problem. This algorithm generates a sequence of relaxed decision diagrams, and solves an NP-hard constrained network flow problem posed as an integer linear program (ILP) on each decision diagram.

In this thesis project, we examined inequalities that are facet-defining for the integer hull of this ILP. We discovered that the majority of these facet-defining inequalities represented or were implied by objective cuts. We subsequently derived a method for efficiently finding these objective cuts. Additionally, we investigated the potential of generating Chvátal-Gomory cuts (CG-cuts) in one iteration, and reusing parts of those CG-cuts in successive iterations. To generate such cuts, we set up a mixed-integer linear program (MILP) that finds a CG-cut for the constrained network flow ILP that has maximum violation with respect to some fractional solution. We created another model that takes part of a CG-cut from a past iteration as input and completes this CG-cut to fit the ILP in the current iteration. We furthermore designed a heuristic algorithm that can complete such CG-cuts efficiently. The performance of our CG-cut reusage scheme was tested on a selected sample of DIMACS instances. Our results on these instances indicate that the CG-cut reusage scheme generally yields positive violation CG-cuts. Furthermore, the best cuts produced with the heuristic algorithm in each iteration are close to the theoretically best cuts that can be derived.

Acknowledgements

This thesis marks the final endeavor of my master Applied Mathematics. Coming from a more practically oriented bachelor (Industrial Engineering & Management), I expected this rather theoretical project to be quite the challenge. And it indeed has been, though I had a lot of fun taking it on. Of course, I did not do it all on my own, and I would like to dedicate this section to the people that made this project possible.

First and foremost, I would like to extend my sincere gratitude to my thesis supervisor, Matthias Walter. Thank you for all the moments where you really inspired me. Thank you for your enthusiasm, your patience and, above all, your unconditional willingness to help. I think you are an absolute genius, and I feel fortunate that I had the opportunity to collaborate with you on this project. Secondly, I would like to express my deepest appreciation to my mom and dad for their unwavering support throughout my master and my life as a whole. You really helped me to get to the place where I am now, and it is a place I am very happy to be in. Thirdly, I would like thank my family and friends, and especially those friends in my informal study group. Abel, Fiona and Maarten, even through the hardest parts of my thesis, you helped me to stay motivated, and I would have lost my sanity without you all. Finally, I would like to sincerely thank my boyfriend, Victor. No matter what, you have always been there for me, and your steadfast support is one of the main reasons why I succeeded in finishing this thesis. I honestly don't know what I would be doing without you.

Contents

1 Introduction

The graph coloring problem is a well-known NP-hard combinatorial optimization problem. A coloring of a graph refers to an assignment of colors (or labels) to all vertices on that graph such that no two adjacent vertices have the same color. The objective of the graph coloring problem is to find such a coloring while minimizing the number of different colors used. Though the graph coloring problem is in essence a purely theoretical problem, a wide range of real-world problems can be posed as a graph coloring problem such as timetabling and scheduling problems [8], [25], [27] and frequency assignment problems [1], [24], [27]. Methods to solve the graph coloring problem include meta-heuristics [26], [32], integer linear programming [17], [29] and column generation [28], [31].

In two recent publications by van Hoeve [15], [16], an exact algorithm employing decision diagrams was introduced for solving the graph coloring problem. A decision diagram is a layered directed acyclic graph that can be used to compactly represent (a relaxation of) the solution set of an optimization problem. The graph coloring algorithm by van Hoeve iteratively refines a decision diagram that encodes (not necessarily feasible) color assignments for the graph to be colored. In each iteration, a constrained network flow that represents such a color assignment is computed on the decision diagram. This constrained network flow problem is posed as an integer linear program (ILP). If this color assignment is infeasible, a conflict (a pair of adjacent vertices with the same color) is identified and the diagram is adapted such that this conflict is separated. Unfortunately, van Hoeve proved in his paper that this constrained network flow problem is NP-hard, and it is noted to be the computational bottleneck of the algorithm [16, p. 647].

As far as we are aware, the integer hull of this constrained network flow ILP has not been studied. For some decision diagram with node set N, arc set A and some disjoint subsets $A_1, A_2, ..., A_n \subseteq A$ of the arc set, this integer hull is the following:

$$\begin{aligned} P_I^{\text{flow}} = \text{conv}\{y \in \mathbb{Z}_{\geq 0} : \sum_{a \in \delta^-(u)} y(a) - \sum_{a \in \delta^+(u)} y(a) = 0 \qquad & \forall\ u \in N \setminus \{r, t\}, \\ \sum_{a \in A_j} y(a) = 1 \qquad & \forall\ A_j, \\ y(a) \geq 0 \qquad & \forall\ a \in A.\} \end{aligned}$$

We theorize that the constrained network flow ILP can be strengthened through facet-defining inequalities, which are inequalities that are valid for the integer hull of this. To that end, we studied the integer hull of this constrained network flow problem to identify families of such facet-defining inequalities. In doing so, we attempted to answer the following research question:

> **(Q1)** *What does the integer hull of the constrained network flow integer linear program in van Hoeve's graph coloring algorithm look like?*

We used computational methods of the field—Investigating Polyhedra by Oracles (IPO) [36], [37], which uses the SCIP software suite [5]—to find facet-defining inequalities valid for P_I^{flow} on a variety of graph coloring instances. A large majority of the facet-defining inequalities we found represented or were implied by objective cuts, which are inequalities that force a bound on the objective function of the ILP. One major challenge for the implementation of such objective cuts is finding a good objective bound value. Fortunately, van Hoeve's graph coloring algorithm solves a sequence of similar ILPs, and we show that this similarity can be exploited to generate objective cuts that force strong objective bounds.

During our work on objective cuts, we speculated that this similarity can also be exploited to generate general purpose cutting planes, which are inequalities that cut off fractional points. In particular, we hypothesized that those cutting planes can be used for multiple iterations of van Hoeve's graph coloring algorithm due to this similarity. As such, we posed a second question:

> **(Q2)** *How can cutting planes for the constrained network flow integer linear program in some iteration of van Hoeve's graph coloring algorithm be reused for successive iterations?*

In our pursuit of an answer to this question, we developed a novel scheme in which cutting planes are reused for sequences of ILPs such as the ones in van Hoeve's algorithm. This scheme functions by storing those parts of the cutting planes that are related to all ILPs in the sequence, and completing each cut with parts specific to each ILP. We propose an efficient algorithm for cut completion, and show results on this algorithm and the scheme as a whole.

1.1 Related work

The decision diagram based graph coloring algorithm was first introduced by van Hoeve in 2020 [15]. In this publication, he also shows correctness of the algorithm and he experimentally shows that the algorithm is competitive with state-of-the-art algorithms for finding lower bounds to the chromatic number of a graph. In a follow-up publication in 2021, van Hoeve provides more theoretical results on the algorithm and a more detailed discussion of the experimental results [16]. Van Hoeve's graph coloring algorithm was furthermore used in a case study on variable orderings for decision diagrams by Karahalios & van Hoeve [21].

While the application of this particular network flow formulation to decision diagrams is novel, another network flow formulation on decision diagrams has been widely studied before. Behle [3] introduced this network flow formulation on decision diagrams representing 0/1 polytopes. He showed that this network flow formulation can be used for enumerating the facets of the 0/1 polytope and generating cutting planes for that polytope. This network flow formulation has been applied in a wide variety of other studies as well. For example, Tjandraatmadja and van Hoeve [35] solves this 0/1 network flow problem to generate target cuts for binary optimization problems, and Cire et al. [11] reformulated this network flow problem to model a feasible set of clinical rotation schedules. For a broad overview of the literature on network flow formulations on decision diagrams—and optimization with decision diagrams in general—we refer to the literature survey by Castro et al. [9].

Several types of flow problems have been studied that are similar to the constrained network flow problem we examine in this thesis. Those problems include the minimum cost flow problem with conflict constraints [34] or disjunctive constraints [33], the flow problem with disjoint bundles [13], [18] and the minimum cost noncrossing flow problem [2]. As far as we are aware, no studies on the polyhedra of these problems have been conducted so far.

1.2 Outline

This thesis report is structured as follows: in Chapter 2, we lay the theoretical foundation for this thesis and describe van Hoeve's algorithm in detail. We answer our first question—**(Q1)**—in Chapter 3, in which we show our theoretical results on the facet-defining inequalities we found. The second question—**(Q2)**—is answered in Chapter 4 as we explore methods for finding and reusing general purpose cutting planes for the constrained network flow problem. In the final chapter, Chapter 5, we conclude this thesis with some closing remarks.

2 Theory & notation

In this chapter, we explain the theoretical concepts that are relevant to the context of this thesis. In Section 2.1, we discuss our definition of the graph coloring problem and define related concepts. In Section 2.2, we explain in detail what a decision diagram is and how it relates to optimization. Having formalized both graph coloring and decision diagrams, we provide a detailed discussion of van Hoeve's graph coloring algorithm [15], [16] in Section 2.3. In this section, the constrained network flow integer program that we study in this thesis is also introduced and explained. We end this chapter with some remarks on polyhedra in Section 2.4.

Before we discuss these four topics, we shortly introduce common notation for several graph-related concepts, matrices and vectors. We use set notation, i.e. $\{u, v\}$ to denote an undirected edge between nodes u and v. Conversely, an *arc* or directed edge is denoted using brackets, i.e. for nodes u and v, (u, v) is an arc directed from u to v. For an undirected graph $G = (V, E)$, we define $N_v := \{u \in V : \{u, v\} \in E\}$ to be the *neighborhood* of a vertex $v \in V$, which are all the nodes adjacent to v. For a directed graph $G = (V, A)$ and some node $u \in V$, $\delta^+(u) := \{(u, v) : (u, v) \in A\}$ denotes the set of *outgoing arcs* of u, and we let $\delta^-(u) := \{(v, u) : (v, u) \in A\}$ denote the set of *incoming arcs* of u. Furthermore, we define a *path* on a directed graph $G = (V, A)$ to be a sequence of distinct arcs $(a_1, a_2, ..., a_k)$, all pointing in the same direction, that joins a sequence of distinct nodes. To indicate that an arc a is on some path p, we slightly abuse notation by writing that $a \in p$.

For some matrix B, we let $B_{i,j}$ denote the element in the i^{th} row and j^{th} column, and we indicate the entire i^{th} row of B by $B_{i,*}$ and the entire j^{th} column of B by $B_{*,j}$. Similarly, x_i denotes the i^{th} element of some vector x. Lastly, $\mathbf{1}$ denotes a vector of all ones, and $\mathbf{0}$ denotes the zero-vector.

2.1 Graph coloring

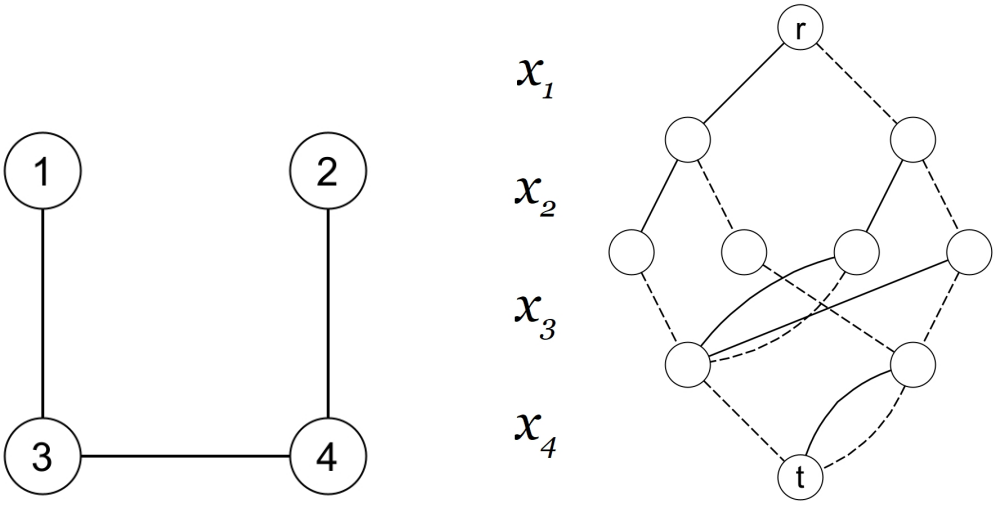
In the context of this thesis, with "graph coloring", we specifically refer to a coloring of vertices. Let $G = (V, E)$ be an undirected simple graph. We define a *coloring* of G to be an assignment of a color $c(v)$ to each vertex $v \in V$. We define such a coloring to be a *feasible coloring* if each vertex v is assigned a color such that no two adjacent vertices have the same color, i.e., for all $\{u, v\} \in E$, $c(u) \neq c(v)$. We then define a *minimum coloring* to be a feasible coloring that uses the fewest number of colors, and we define the *chromatic number* $\chi(G)$ of G to be the minimum number of colors required for a feasible coloring of G. We lastly define the *graph coloring problem* on G to be the problem of finding a minimum coloring of G. Deciding if an arbitrary graph admits a feasible coloring of k colors was shown to be NP-complete by Karp [22], so the problem of finding the chromatic number for an arbitrary graph is NP-hard. As a result, the graph coloring problem, is also NP-hard.

For any coloring of G, we define a *color class* for a particular color to be a subset of V containing all vertices that have been assigned that color. For a feasible coloring, each color class forms an *independent set*, which we define as a subset of V such that no two vertices in that subset are adjacent. Then each feasible coloring is a partition of the node set into mutually exclusive independent sets.

2.2 Decision diagrams

In the context of this thesis, a *decision diagram* is a layered directed acyclic graph $D = (N, A)$ with node set N and arc set A. The node set N is partitioned into $n + 1$ layers. The first layer consists of a root node r and the last layer consists of a sink node t. We denote the layer of a node $v \in N$ by $L(v)$, and we denote the set of layers by $\mathcal{L}$.

Every arc $(u, v) \in A$ is directed from a node $u \in N$ to a node $v \in N$ in a consecutive layer, i.e. $L(v) = L(u) + 1$ for all $(u, v) \in A$. Though the layers are only defined for the nodes, we say that an arc $(u, v) \in A$ is in layer j if $L(u) = j$. Furthermore, each arc $a \in A$ arc is assigned a label $\ell(a) \in \{0, 1\}$. Similarly to van Hoeve [16], we refer to an arc with label 1 as a *1-arc* and an arc with label 0 as a *0-arc*. Each node $v \in N \setminus \{t\}$ has a unique outgoing 0-arc, and may have a unique outgoing 1-arc, while the sink node t has no outgoing arcs. Each node $v \in N \setminus \{r\}$ also must have



(a) *Input graph* **(b)** *Exact decision diagram*

Figure 1: Input graph to be colored (1a) and the associated exact decision diagram (1b). Each layer x_i on the decision diagram is related to a node i of the input graph. Dashed arcs represent 0-arcs, while 1-arcs represent 1-arcs. This example is taken from [16].

an incoming arc, while the root node r does not have an incoming arc. By this construction, each arc and each node is part of some directed r-t path. See Figure 1b for an example.

Decision diagrams can be built to graphically represent (an over-approximation of) the solution space of optimization problems. Consider some optimization problem Π with an ordered set of binary variables $\{x_1, x_2, ..., x_n\}$ and feasible set Sol(Π). Furthermore, consider some decision diagram $D = (N, A)$ for Π. Then D has $n + 1$ layers, and the nodes in layer $j \in \mathcal{L}$ are associated with variable x_j. Moreover, any r-t path $(a_1, a_2, ..., a_n)$ on D corresponds to a variable assignment for Π by setting $x_j := \ell(a_j)$. We let $\mathcal{P}(D)$ denote the set of all r-t paths on D, and we let $\mathcal{X}(D)$ denote the collection of all variable assignments encoded by r-t paths on D, formally defined as:

$$\mathcal{X}(D) = \{\{x_1, x_2, ..., x_n\} : \exists (a_1, a_2, ..., a_n) \in \mathcal{P}(D) \text{ with } \ell(a_k) = x_k \in \{0, 1\} \text{ for } k = 1, ..., n\}.$$

As an example, consider Figure 1. Each r-t path on the decision diagram in Figure 1b encodes independent sets of the input graph in Figure 1a.

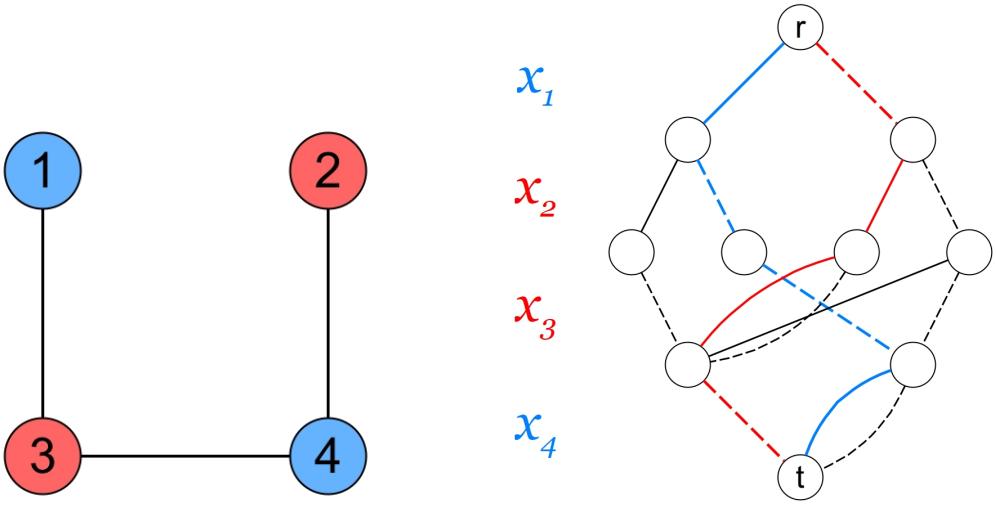
2.2.1 Relation to graph coloring

Consider a graph $G = (V, E)$ with node set $V = \{v_1, v_2, ..., v_n\}$ and a decision diagram $D = (N, A)$ for G. In the context of graph coloring, we let an r-t path $(a_1, a_2, ..., a_n) \in \mathcal{P}(D)$ encode a subset $V' \subseteq V$ by setting $V' = \{v_i \in V : \ell(a_i) = 1\}$. Now, consider a set of r-t paths $\{p_1, p_2, ..., p_f\} \subseteq \mathcal{P}(D)$. Those paths encode node sets $V_1, V_2, ..., V_f$ that partition the node set V if, for each layer, exactly one path in the path set $\{p_1, p_2, ..., p_f\}$ has a 1-arc in that layer. Such a partition can be used to define a coloring c of G by

1. $c(u) = c(v)$ for all $u, v \in V_i$, $i = 1, .., f$,
2. $c(u) \neq c(v)$ for all $u \in V_i$ and $v \in V \setminus V_i$ for $i = 1, .., f$.

Then we define Sol(D) to be a family of subsets of $\mathcal{X}(D)$ such that, for each $k = 1, ..., n$, exactly one element of each subset has $x_k = 1$. Then, since $X \in \mathcal{X}(D)$ encodes a subset of the nodes V, any $C \in$ Sol(D) encodes a partition of the node set V by construction. As a result, any element $C \in$ Sol(D) can be translated to a coloring of G by the above procedure. Formally, we define Sol(D) as:

$$\text{Sol}(D) = \left\{ \{X_1, ..., X_f\} \subseteq \mathcal{X}^{(i)} : \sum_{j=1}^{f} X_j(k) = 1 \text{ for } k = 1, ..., n \right\},$$



(a) *Input graph with feasible color assignment*

(b) *Exact decision diagram D with paths representing color classes*

Figure 2: The same example as Figure 1 (taken from [16]), but now with a color assignment. The blue r-t path in (2b) corresponds to the blue color class on (2a), and the red r-t path corresponds to the red color class. Each colored r-t path encodes an element of $\mathcal{X}(D)$ (an independent set). In each layer, exactly one colored path in (2b) traverses a 1-arc. The red and blue r-t paths together therefore encode an element of $\text{Sol}(D)$ (a coloring).

where $X(k)$ denotes the k^{th} element of $X \in \mathcal{X}^{(i)}$. Figure 2 illustrates each of the sets defined in this section.

Note that $\text{Sol}(D)$ is defined here to encode colorings of G, but that we did not restrict to *feasible* colorings of G. Here, we make a distinction between exact decision diagrams and relaxed decision diagrams. We define a decision diagram D for a graph coloring problem Π to be an *exact decision diagram* if $\text{Sol}(\Pi) = \text{Sol}(D)$, and a *relaxed decision diagram* if $\text{Sol}(\Pi) \subseteq \text{Sol}(D)$. For relaxed decision diagrams, not all colorings encoded by $\text{Sol}(D)$ are feasible.

2.2.2 Variable ordering & reduced diagrams

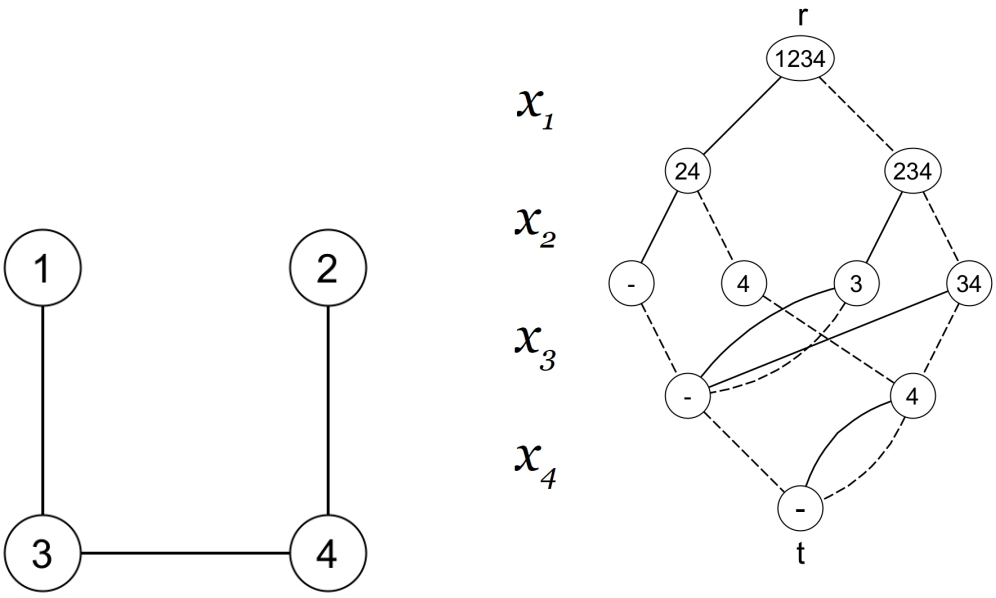
An advantage of decision diagrams is that they have to potential to compactly represent solution sets. Whether this potential is realized depends largely on two factors: the variable ordering and eliminating redundancy.

The *variable ordering* is the assignment rule of the variables to the layers. For example, for a set of binary variables $\{x_1, x_2, ..., x_n\}$, a possible variable ordering can be the lexicographical ordering, under which a node labelled j is assigned to layer j. The choice of variable ordering has a large impact on the size of *exact* decision diagrams, and experimental results indicate that *relaxed* decision diagrams compiled with good variable orderings give better lower bounds [4]. Unfortunately, the problem of finding an optimal variable ordering is NP-hard [6]. In the context of graph coloring, Karahalios and van Hoeve [21] tested six heuristic variable orderings on several graph instances, and found that the so-called Minimum Width heuristic performed the best, though none of the six tested heuristics strictly dominated another. The Minimum Width heuristic is described in Algorithm 1.

The second factor we alluded to was redundancy elimination. For a decision diagram $D = (N, A)$, we define the *subgraph rooted by* $v \in N$ to be the graph consisting of v and all of its descendants. Nodes $v \in N$ and $v' \in N$ are said to be *equivalent* nodes if the subgraphs rooted by v and v' are isomorphic. The two equivalent nodes v and v' then have the same set of completions, i.e. every (v, t) path encodes a partial variable assignment that is also encoded by some (v', t) path and vice versa. Either one of these nodes is then redundant, as a smaller decision diagram $D' = (N', A')$ can be created with $\text{Sol}(D') = \text{Sol}(D)$ by merging v and v' into one node.

Algorithm 1: Minimum Width Variable Ordering Algorithm [21]

Input: Graph $G = (V, E)$.
Output: Ordered list of Vertices L
Definition: $\deg(v, G)$ is the degree of v in G. $L \leftarrow \emptyset$
while V *not empty* **do**
  $N \leftarrow \arg\min_{v \in V}\{\deg(v, G)\}$
  $V \leftarrow V - N$
  $E \leftarrow E - \{(i, v) : (i, v) \in E, v \in N\}$
  $L \leftarrow N{:}L$ {add N to front of L}
  $G \leftarrow (V, E)$
end



(a) *Input graph*

(b) *Exact decision diagram with node labels representing state information*

Figure 3: The unique exact reduced decision diagram (3b) for input graph (3a) (taken from [16]). Each node in (3b) has a label representing the state information of that node.

We call a decision diagram *reduced* if no two nodes in a layer are equivalent. For a given set of variable assignments, a reduced diagram is the smallest possible decision diagram that exactly represents this set [10]. A key result by Bryant [7] is that there exists a *unique* reduced decision diagram for each fixed variable ordering. Furthermore, Bergman et al. [4] showed that, for decision diagrams for independent set problems, two nodes are equivalent if and only if they have the same *state information*. The state information $S(u) \subseteq V$ of a node $u \in N$ on D is a set consisting of all nodes $v_i \in V$ on G that can be encoded by some (u, t) path on D. Essentially, for a partial variable assignment $X = \{x_1, x_2, ..., x_{j-1}\}$ that encodes a subset V', $S(u)$ is the set of all other candidates $v_i \in V$ that can still be added to V' (though not necessarily all at the same time). We refer to Figure 3 for an example of state information in decision diagrams. This result of Bergman et al. [4] implies that, for decision diagrams encoding independent sets, the unique reduced decision diagram can be built by only tracking the state information $S(u)$ to check node equivalence.

Nevertheless, even for reduced exact decision diagrams with an optimal variable ordering, the number of nodes may scale exponentially (Theorem 7 of [16]). As such, relaxed decision diagrams may be desirable for large-scale problems because they can be compiled with limited size. For example, van Hoeve's shows that there are graph coloring instances where an optimal solution can be found with a relaxed decision diagram that is exponentially smaller than the reduced exact decision diagram for that problem (Theorem 7 of [16]).

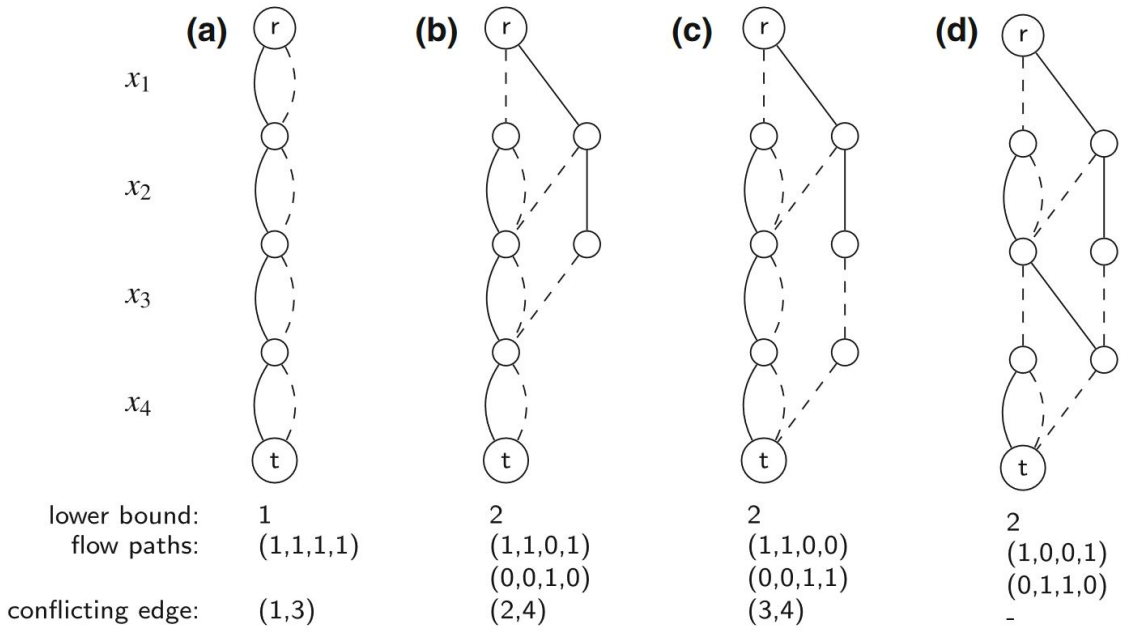


Figure 4: Applying constraint separation to the input graph in Figure 1a. The initial diagram for Figure 1a is represented in (4a). The graph is refined three times with Algorithm 2 until a feasible coloring has been found in (4d). Note that (4c) still contains a path that encodes (2,4) even though that conflict was resolved along another path in the previous iteration. This figure was directly copied from [16, p.637].

2.3 Van Hoeve's graph coloring algorithm

[This section is based on van Hoeve's paper "Graph Coloring with Decision Diagrams" [16]]. We refer the reader to this paper for more details, such as proofs of the lemmata and theorems shown in this section.

In this section, we provide all information on the decision diagram based graph coloring by van Hoeve [15], [16] that is necessary for understanding the remainder of this thesis. From a high-level perspective, the algorithm initializes a relaxed decision diagram and iteratively refines to tighten the relaxation. In each iteration, it solves a flow problem on the (refined) decision diagrams to compute colorings of the input graph until a feasible coloring for that graph is found. The decision diagram is refined using a method called *constraint separation*, which is the removal of a path on the decision diagram that encodes a part of an infeasible solution [10].

To describe the algorithm, we start by explaining how the algorithm initializes the decision diagram. We continue by showing how the decision diagram is refined in each iteration with constraint separation. We finish this section by showing how the algorithm computes colorings, which is through the constrained network flow problem that is the topic of this thesis.

2.3.1 Initialization

For some input graph $G = (V, E)$ with $|V| = n$, van Hoeve's algorithm initializes a decision diagram $D = (N, A)$ with $n + 1$ layers and one node per layer, where each layer j is associated to some node $v_j \in V$. The first layer consists of the root node r and the last layer being the sink node t. We define the *initial nodes* to be all the nodes present in this initial decision diagram. Each initial node $u \in N \setminus \{t\}$ in layer $L(u) = j$ has both an outgoing 1-arc and an outgoing 0-arc, both directed to the initial node in the next layer—the initial node $v \in N$ with $L(v) = j + 1$. An example of such an initial diagram is depicted in Figure 4.

As a result of this construction, any subset $V' \subseteq V$ is encoded by some r-t path on D, and therefore all independent sets of V are encoded by D. As a result, this initial decision diagram D is a valid relaxed decision diagram. Furthermore, each initial node $u \in N$ in layer $L(u) = j$ is assigned state information $S(u) = \{v_j, v_{j+1}, ..., v_n\}$, since all the nodes in this state information are encoded by

the (u, t)-path consisting of only 1-arcs. Since t is located in layer $n + 1$, it follows that $S(t) = \emptyset$.

2.3.2 Constraint separation

The initial diagram is then refined by a subroutine that van Hoeve calls the *separation algorithm*. The separation algorithm refines an input decision diagram by separating a *conflict* along a path r-t path. A conflict is a tuple (j, k) with $j < k$ such that $\{v_j, v_k\} \in E$ is an edge of G. An r-t path $(a_1, ..., a_j, ..., a_k, ..., a_n) \in \mathcal{P}(D)$ containing a conflict (j, k) has a 1-arc in layer j and k, and therefore represents an infeasible variable assignment. The separation algorithm has the following inputs: 1) a reduced decision diagram D, 2) an edge conflict (j, k) with $j < k$, and 3) and the path with this conflict $(a_j, a_{j+1}, ..., a_{k-1})$ that visits nodes $u_j, u_{j+1}, ..., u_{k-1}$. This path is not allowed to contain another edge conflict (j', k') such that $j \leq j' < k' < k$.

The separation algorithm resolves the conflict by adding a new node to layers $j + 1$ to k and redirecting the input path along these nodes. Each new node in layer $i + 1$ for $i = j, ..., k - 1$ is assigned state information based on the state of their parent node. If the arc a_i is a 1-arc, then the neighbors N_{v_i} of $v_i \in V$ are removed from the assigned state information. Recall that two nodes are equivalent if they have the same state information [4], so if the new node has the same state information as another node in layer i, then those nodes are merged. If the node is not merged, then the new node receives an outgoing 1-arc if v_{i+1} is its state information. It also receives the same outgoing 0-arc as node u_{i+1}. The separation algorithm is described in Algorithm 2. See Figure 4 for an example of three applications of Algorithm 2 to the input graph in Figure 1a.

The following lemma by van Hoeve shows that the separation algorithm successfully separates a conflict along a path:

Lemma 2.1 (Lemma 1 of van Hoeve [16])**.** *Let decision diagram D, edge conflict (j, k), and node-arc specified path $(u_j, a_j, u_{j+1}, a_{j+1}, ..., u_{k-1}, a_{k-1})$ be the input to Algorithm 2. The application of the algorithm results in a decision diagram in which label-specified path $(\ell(a_j), ..., \ell(a_{k-1}), 1$ starting at u_j no longer exists, but $(\ell(a_j), ..., \ell(a_{k-1}), 0)$ with $\ell(a_k) = 0$ does.*

Another important result is that the decision diagram refined with the separation algorithm does not encode new variable assignments that the original did not. This was shown in the following Lemma by van Hoeve:

Lemma 2.2 (Lemma 2 of van Hoeve [16])**.** *Algorithm 2 does not introduce new label-specified paths to the input decision diagram.*

Lastly, the following theorem by van Hoeve states that, if the separation algorithm is applied enough times, the resulting decision diagram only encodes feasible colorings:

Theorem 2.1 (Theorem 1 of van Hoeve [16])**.** *Given a reduced decision diagram D as input and an oracle that provides us with edge conflicts and associated paths in D, repeated application of Algorithm 2 results in the unique reduced exact decision diagram.*

The oracle alluded to in this theorem is the topic of the next section.

2.3.3 Conflict detection

Recall that, for a (relaxed) decision diagram $D = (N, A)$ for a graph $G = (V, E)$, we defined $\text{Sol}(D)$ to be the set of all partitions of V encoded by D. Van Hoeve's algorithm finds such partitions by solving a *minimum constrained network flow problem* on D using integer programming. For each arc $a \in A$, a variable y_a is introduced that represents the (integer) flow through a. Then the

Algorithm 2: Separating edge conflict (j, k) in decision diagram D (van Hoeve [15], [16])

Input: reduced decision diagram D (D$[i][j]$ represents the jth node in layer i), a path $(a_j, ..., a_{k-1})$, node indices $u_j, ..., u_{k-1}$ of the nodes visited on the path, a conflict (j, k) (it is assumed that the path contains no edge conflicts (j', k') such that $j \leq j' < k' < k$) and list of neighbors N_v for each $v \in V$.

Output: reduced decision diagram in which the conflict along the path has been eliminated.

for $i = j$ **to** $k - 1$ **do**
  create node w `// split the path towards node` w
  $w.S \leftarrow$ D$[i][u_i].S \setminus \{v_i\}$ `// copy the parent state and remove` i
  if $\ell(a_i) = 1$ **then** $w.S \leftarrow w.S \setminus N_{v_i}$; `// remove` N_{v_i} `in case of 1-arc`
  $t \leftarrow -1$ `//` t `is index of the new node in layer` $i+1$
  if $\exists k$ *such that* $D[i+1][k].S = w.S$ **then** $t \leftarrow k$; `// check for equivalent node`
  else
    if $v_{i+1} \in w.S$ **then** w.oneArc $\leftarrow$ D$[i+1][u_{i+1}]$.oneArc; `// copy 1-arc`
    else w.oneArc $\leftarrow -1$
    w.zeroArc $\leftarrow$ D$[i+1][u_{i+1}]$.zeroArc `// copy 0-arc`
    D$[i+1]$.add(w) `//` `append` w `as the new node to layer` $i+1$
    $t \leftarrow |$D$[i+1]|$ `//` `update` t `to last index of layer` $i+1$
  if $\ell(a_i) = 1$ **then** D$[i][u_i]$.oneArc $\leftarrow t$; `//` `re-direct path to` w `in case of 1-arc`
  else D$[i][u_i]$.zeroArc $\leftarrow t$; `//` `re-direct path to` w `in case of 0-arc`
  $u_{i+1} \leftarrow t$ `// update path index`

minimum constrained network flow ILP (F) is as follows:

$$
\begin{aligned}
(F) \;=\; \min \; & \sum_{a \in \delta^+(r)} y_a & & \\
\text{s.t.} \; & \sum_{a=(u,v) \mid L(u)=j, \ell(a)=1} y_a = 1 & \forall\, j \in \mathcal{L}, & \quad (2a) \\
& \sum_{a \in \delta^-(u)} y_a - \sum_{a \in \delta^+(u)} y_a = 0 & \forall\, u \in N \setminus \{r, t\}, & \quad (2b) \\
& y \in \mathbb{Z}_{\geq 0} & \forall\, a \in A. & \quad (2c)
\end{aligned}
$$

The objective function of (F) minimizes the total amount of flow on D, as any $r - t$ flow passes an arc outgoing of r. Constraints (2b) ensure flow conservation, which is why we refer to them as the *flow conservation constraints*. Constraints (2c), which we refer to as *the integrality constraints*, force all flow variables to have a nonnegative integer value. Note that we slightly deviate from van Hoeve's original formulation, as he defines $y_a \in \{0, 1, ..., n\}$, while we allow y_a to be any nonnegative integer value. Constraints (2a) ensure that in each layer, exactly one 1-arc a has $y_a = 1$ since all flows have to be integer by the integrality constraints. This constraint ensures that each node $v \in V$ on G is encoded exactly once. We call these constraints *the 1-arc constraints*.

This minimum constrained network flow ILP formulation (F), which we refer to as the *flow model* (F), is the central topic of this thesis. We use Sol(F) to denote the solution space of this model, and we use y exclusively to refer to any vector $y \in$ Sol(F). We use val(y) to refer to the objective value of any vector $y \in$ Sol(F), and $y^* \in$ Sol(F) indicates an optimal solution to (F).

Any solution of flow model (F) can be decomposed into a set of r-t paths, each with a flow value of 1, to obtain a set of paths on D that encode a partition of V. Van Hoeve proves that this holds true in the following Lemma:

Lemma 2.3 (Lemma 5 of van Hoeve [16])**.** *A solution to flow model (F) corresponds to a (not necessarily unique) partition of vertex set V.*

Note that, as the lemma states, it is possible that a flow can be decomposed in a number of different ways.

Algorithm 3: Iterative refinement by conflict detection and separation. (van Hoeve [15], [16])

Input: input graph $G = (V, E)$, and list of neighbors N_v for each $v \in V$
Output: chromatic number of G.

```
foundSol ← false
initialize decision diagram D
while foundsol = false do
    solve model (F) with decision diagram D
    lowerBound ← obj(F)
    decompose solution vector of (F) to determine conflict (j, k) with node/label path vectors
      P, L
    if no conflict is detected then foundSol ← true
    else separate conflict (j, k) along path P, L in D using Algorithm 2
return lowerBound
```

Furthermore, this partition is guaranteed to be a minimum coloring in case the decision diagram is exact:

Theorem 2.2 (Theorem 2 of van Hoeve [16])**.** *If the decision diagram is exact, flow model* (F) *finds an optimal solution to the graph coloring problem.*

An optimal solution to flow model (F) on some relaxed decision diagram may also encode a minimum coloring, though there is no such guarantee.

Van Hoeve provides an algorithm that decomposes any optimal solution $y^* \in \text{Sol}(F)$ into r-t paths and checks for edge conflicts. If it finds such a conflict, it terminates and returns the conflict and the associated path. If it does not find a conflict, that means that the solution corresponds to a feasible coloring. This algorithm itself is outside of the scope of this thesis.

The flow decomposition algorithm and flow model (F) together form the "oracle" for finding conflicts that is required for the separation algorithm (Algorithm 2). These three building blocks together form van Hoeve's graph coloring algorithm, which is shown in Algorithm 3. By Theorem 2.1, repeated application of the separation algorithm yields an exact decision diagram, and by Theorem 2.2, flow model (F) finds an optimal solution to the graph coloring problem if the decision diagram is exact, so Algorithm 3 finds an optimal solution. This is further formalized in the following theorem:

Theorem 2.3. *Given a graph G, Algorithm 3 computes the chromatic number of G.*

Note that Algorithm 3 only returns the chromatic number, but could additionally return a minimum coloring since the last path decomposition does not contain any conflicts. Furthermore, if the algorithm is terminated early, the algorithm finds a lower bound to the chromatic number of G.

We end this section with a closing remark on complexity. Both the separation algorithm and the flow decomposition run in time polynomial in the number of nodes. However, as mentioned in the introduction, van Hoeve proved that flow model (F) is NP-hard:

Theorem 2.4 (van Hoeve Theorem 3)**.** *Solving model (F) for an arbitrary decision diagram is NP-hard.*

The NP-hardness of flow model (F) motivated us to study this formulation.

2.4 A note on polyhedra

To close this chapter, we introduce some concepts related to polyhedra. Firstly, we define the *LP relaxation* of an IP model to be the same model without integrality requirements on the variables. For flow model (F), we derive such an LP relaxation in standard form as follows: let I_m denote the $m \times m$ identity matrix with $m = |A|$, and let $\text{In}(D)$ denote the *node-arc-incidence matrix* of D for all nodes except r and t. The node-arc-incidence matrix of $D = (N, A)$, for which the rows

correspond to nodes and the columns correspond to arcs, has the following entries:

$$\mathrm{In}(D)_{v,a} = \begin{cases} -1 & \text{if arc } a \in A \text{ leaves } v \in N \setminus \{r,t\}, \\ 1 & \text{if arc } a \in A \text{ enters } v \in N \setminus \{r,t\}, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, define $\Lambda \in \{0,1\}^{n \times A}$ as follows:

$$\Lambda_{j,a} = \begin{cases} 1 & \text{if arc } a \in A \text{ is a 1-arc in layer } j \text{ on } D = N, A \\ 0 & \text{otherwise.} \end{cases}$$

Lastly, relaxing the integrality requirement on the integrality constraints (2c) yields $y_a \geq 0$ for all $a \in A$. Then the standard form of the LP relaxation of flow model (F) is as follows:

$$\begin{aligned} (F^{rel}) \; = \; & \min \sum_{a \in \delta^+(r)} y_a \\ & \text{s.t.} \;\; \Lambda y \geq \mathbf{1} && \text{(3a)} \\ & \quad \mathrm{In}(D) y \geq \mathbf{0} && \text{(3b)} \\ & \quad -\mathrm{In}(D) y \geq \mathbf{0} && \text{(3c)} \\ & \quad I_m y \geq \mathbf{0} && \text{(3d)} \end{aligned}$$

We refer to this model as the LP relaxation (F^{rel}). Furthermore, we let:

$$B = \begin{pmatrix} \Lambda \\ \mathrm{In}(D) \\ -\mathrm{In}(D) \\ I_m \end{pmatrix}, \qquad\qquad e = \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}.$$

Then the feasible set of the LP relaxation (F^{rel}) is given by the polyhedron $P^{flow} := \{y \in \mathbb{R} : By \geq e\}$.

We say that an inequality $\alpha^\intercal y \geq \gamma$ is *valid* for any polyhedron $P = \{Ay \geq b\}$ if $\alpha^\intercal \hat{y} \geq \gamma$ for all $\hat{y} \in P$. We define a *face* of P to be a set $\mathcal{F} = P \cap \{y : \alpha^\intercal y = \gamma\} \neq \emptyset$ such that $\alpha^\intercal y \geq \gamma$ is valid for P. Informally, a face can be seen as a boundary of the polyhedron. We call a face a *vertex* if $\mathcal{F} = \{y\}$, and we call a face a *facet* if $\dim(\mathcal{F}) = \dim(P) - 1$. Note that P^{flow} does contain at least one vertex as a consequence of the non-negativity constraints on y (Constraints 3d).

We use $P_I^{flow} = \mathrm{conv}\{P^{flow} \cap \mathbb{Z}^A\}$ to denote *integer hull* of flow model (F). Since B is rational, it follows that P_I^{flow} is also a polyhedron [30]. Secondly, since all integer points of P^{flow} are contained in P_I^{flow}, it follows that $\mathrm{Sol}(F) \subseteq P_I^{flow}$. Lastly, the vertices of this integer hull are integer points by construction. As a result, any linear programming solution method that finds optimal solutions that are vertices—such as the simplex algorithm—can find integer optimal solutions to $\mathrm{Sol}(F)$ by optimizing over P_I^{flow}.

Unfortunately, we do not have any inequality description of P_I^{flow}, which is why we try to find *facet-defining inequalities*, which are inequalities that induce a facet of P_I^{flow}. Such inequalities are always valid for P_I^{flow} but may not be for P^{flow}. If that is the case, adding these facet-defining inequalities to the LP relaxation (F^{rel}) yields a tighter bound on the relaxation.

3 Objective cuts

Recall from the introduction that the first question we attempted to answer is the following:

> **(Q1)** *What does the integer hull of the constrained network flow integer linear program in van Hoeve's graph coloring algorithm look like?*

In the pursuit of an answer, we generated a number of instances of flow model (F) and found facet-defining inequalities for the integer hulls of all of these instances using IPO [36], [37]. In our examination of these inequalities, we noticed the following:

Observation 3.1. *Nearly all observed facet-defining inequalities represented or were implied by r-t cuts.*

To illustrate this, consider any r-t cut $(X, N \setminus X)$ for some decision diagram $D = (N, A)$ and some $X \subseteq N$ such that $r \in X$ and $t \in N \setminus X$. Let K denote the set of arcs that cross the cut, i.e. $K = \{(u, v) : u \in X, v \in N \setminus X\}$. Then such an inequality representing an r-t cut can be written in the following form:

$$\sum_{a \in K} y_a \geq f. \tag{4}$$

Such a cutting plane forces a flow of at least f through the network, and cuts off fractional solutions if f equals a larger integer amount. The largest value for f that still yields cutting planes valid for the integer hull is the objective value for (F). As such, inequalities of the form 4 actually bound the objective function. In particular, the outgoing arcs of r form an r-t cut, which are also precisely the arcs contributing to the objective function. We will use the term *objective cut* to refer to any cutting plane that explicitly bounds the objective function.

These objective cuts could be added to strengthen flow model (F), but unfortunately, finding the correct bounding value f is NP-hard as a consequence of Theorem 2.4; suppose this bounding value could be found in polynomial time for any decision diagram. This implies that certificate for a yes-instance could be found in polynomial time with some algorithm for any flow value. Repeating this algorithm for all possible flow values $(f = 1, ..., n)$ would then imply a polynomial time algorithm to solve model (F) and P = NP, but we assume the contrary in this thesis.

Fortunately, a decision diagram in some iteration i and its successor in iteration $i+1$ are relatively similar. Then, if the objective values of the optimal solutions in each iterations bear similarity as well, we can use the objective value of some iteration to generate objective cuts for the successive iteration. To support our discussion of objects in multiple iterations in this particular chapter, we use a superscript (i) to denote a certain object in iteration i.

In this chapter, we prove that there is indeed some similarity between the objective values of two successive iterations in the following theorem:

Theorem 3.2. *If Algorithm 3 detects a conflict for an optimal solution $y^{*(i)}$ to flow model $(F^{(i)})$ in some iteration i, then for iteration $i+1$, $val(y^{*(i+1)}) = val(y^{*(i)})$ or $val(y^{*(i+1)}) = val(y^{*(i)}) + 1$.*

As a consequence of this theorem, for some iteration $i + 1$, we can add the following objective cut to accelerate solving in case iterations i and $i + 1$ have the same objective value:

$$\sum_{a \in \delta^+(r)} y_a^{(i+1)} \geq \text{val}(y^{*(i)}).$$

To prove Theorem 3.2, we firstly derive some structural results of the decision diagrams generated with Algorithm 3 in Section 3.1. Using those structural results, we prove that the objective value is monotonically increasing in i in Section 3.2. Using the results from both of these sections, Theorem 3.2 is be proven in Section 3.3.

3.1 Structural results for decision diagrams

In this section, we establish results on the existence of some types of paths on each decision diagrams generated with Algorithm 3. In particular, we show results on zero-paths, which we define as follows:

Definition 3.1 (Zero-path)**.** A *zero-path* is an r-t path for which all arcs on the path have label 0.

A zero-path encodes the empty node set on G. Since the empty set is an independent set, this zero-path exists on each decision diagram formed with Algorithm 3. Notably, it can be shown that such a zero-path always exists along the initial nodes:

Lemma 3.1. *Each decision diagram formed with Algorithm 3 has a zero-path along the initial nodes.*

Proof. The zero-path along the initial nodes exists on the decision diagram initialized in Algorithm 3, as all initial nodes except t have an outgoing 0-arc to another initial node. the separation algorithm (Algorithm 2) does not remove existing arcs and can only redirect them. It therefore suffices to show that the separation algorithm cannot redirect 0-arcs from the initial nodes.

Consider a conflict (j, k) to be resolved along the path $(a_j, ..., a_k)$. Since (j, k) is a conflict, $\ell(a_j) = 1$, so in the first iteration of the separation algorithm, the redirected arc is a 1-arc. In any other iteration $h \in \{j + 1, ..., k - 1\}$ of the separation algorithm, the new node created in that iteration receives an incoming arc from the node that was created or merged in iteration $h - 1$. Therefore, only if a new node in layer h is merged with an initial node in the same layer, a 0-arc can be redirected from an initial node.

A new node is merged with an existing node only if they have the same state information. To show that a 0-arc cannot be redirected from an initial node, we show that no new node can have the same state information as an initial node in the same layer.

All new nodes created with the separation algorithm inherit the state information of the previous nodes along the path. For any iteration except the first, this means that a new node inherits the state information from the node created or merged in the previous iteration.

The new node in the first iteration of the separation algorithm has an incoming 1-arc. It therefore inherits the state information of the previous node and subtracts the neighbours of node $v_j \in V$. Since (j, k) is a conflict, v_k is one of the neighbours of v_j, so v_k is removed from the state information of the node created in the first iteration (line 4 of Algorithm 2). The state information of all nodes created or merged in the next iterations of the separation algorithm therefore also cannot contain v_k.

For every layer h, the state information of the initial node in that layer is $\{h, ..., n\}$. Therefore, all initial nodes in layer 1 to k have state information containing v_k. Furthermore, the state information of an existing node does not change during any iteration of Algorithm 3. No new node can therefore have the same state information as an initial node in the same layer. Therefore, 0-arcs cannot be redirected from the initial nodes, and each decision diagram formed in every iteration of has a zero-path. □

We will denote the zero-path along the initial nodes by p_Z. It follows from Lemma 3.1 that p_Z is actually the only zero-path on each decision diagram:

Corollary 3.1.1. *p_Z is the unique zero-path on each decision diagram formed with Algorithm 3.*

Proof. p_Z exists by Lemma 3.1. Every non-sink node on a decision diagram has exactly one outgoing 0-arc, and all outgoing 0-arcs of initial nodes are directed towards other initial nodes. Since r and t are also initial nodes, there is only one r-t path of only 0-arcs possible. □

Other trivial independent sets on graphs are sets containing only one node. Those sets are encoded by paths with only one 1-arc on a corresponding decision diagram. Using the existence of p_Z, we can show that paths with only one 1-arc exist as well on each decision diagram:

Corollary 3.1.2. *Consider any decision diagram $D = (N, A)$ formed with any number of applications of the separation algorithm. Then the path $p_{k=1} = (a_1, ..., a_j, ..., a_n)$ with $\ell(a_k) = 1$ and $\ell(a_j) = 0$ for $j = 1, ..., k - 1, k + 1, ...n$ exists on $D = (N, A)$ for $k = 1, ..., n$.*

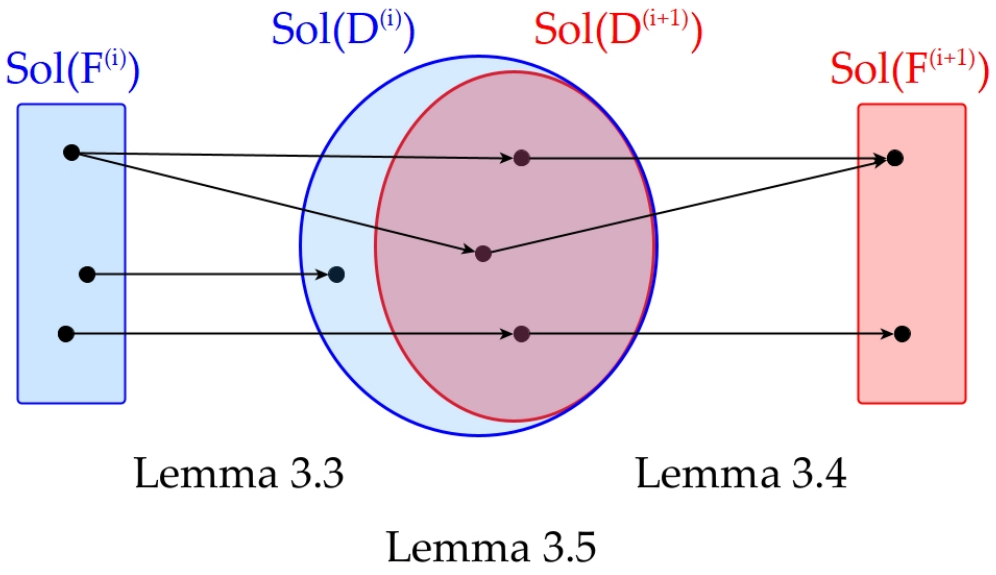


Figure 5: Schematic overview of the monotonicity proof. We relate elements of $\text{Sol}(F^{(i)}$ to elements of $\text{Sol}(D^{(i)}$ in Lemma 3.3. We show in Lemma 3.5 that $\text{Sol}(D^{(i+1)} \subset \text{Sol}(D^{(i)}$. We furthermore relate the elements of $\text{Sol}(D^{(i+1)}$ to elements of $\text{Sol}(F^{(i+1)}$ in Lemma 3.4.

Proof. By Lemma 3.1, there exists a path $(a_1, ..., a_{k-1})$ with $\ell(a_j) = 0$ for $j = 1, ..., k-1$ along the initial nodes, ending in the initial node u_k. In the initial decision diagram, all initial nodes have an outgoing 1-arc and the separation algorithm does not remove arcs. Therefore, there exists an arc $(u_k, u_{k+1}) \in A$ with label 1 $\ell((u_k, u_{k+1})) = 1$ to some node $u_{k+1} \in N$. Since all initial nodes have an outgoing 0-arc and all new nodes are assigned an outgoing 0-arc, there must also exist a path $(a_{k+1}, ..., a_n)$ from u_{k+1} to t with $\ell(a_j) = 0$ for $j = k+1, ..., n$. □

3.2 Monotonicity of the objective function

We continue by proving monotonicity of the objective function. Recall that $\mathcal{X}(D^{(i)})$ denotes the collection of all variable assignments defined by r-t paths on $D^{(i)}$. Furthermore, recall that $\text{Sol}(D^{(i)})$ denotes the set of all partitions of V encoded by paths of $D^{(i)}$, and for each $C \in \text{Sol}(D^{(i)})$, we have that $C \subseteq \mathcal{X}(D^{(i)})$. We establish monotonicity by showing that $\text{Sol}(D^{(i+1)}) \subset \text{Sol}(D^{(i)})$. A consequence of this result is that a smallest cardinality element of $\text{Sol}(D^{(i)})$ is smaller than or equal to a smallest cardinality element of $\text{Sol}(D^{(i+1)})$. If we then relate objective values of solutions $y^{(i)} \in \text{Sol}(F^{(i)})$ to the cardinality of elements of $\text{Sol}(D^{(i)})$, monotonicity follows. For a schematic overview of the monotonicity proof, we refer to Figure 5.

To facilitate the proofs, we introduce a function $\phi : \text{Sol}(F^{(i)}) \times \mathbb{P}(A^{(i)}) \times \mathbb{Z} \longmapsto \mathbb{Z}^{A^{(i)}}$, where $\mathbb{P}(A^{(i)})$ denotes the power set of $A^{(i)}$. This function is, for some $y^{(i)} \in \text{Sol}(F^{(i)})$, some $A' \subseteq A^{(i)}$ and some $f \in \mathbb{Z}$ defined as follows:

$$\phi(y^{(i)}, A', f) := \begin{cases} y_a^{(i)} + f & \text{if } a \in A', \\ y_a^{(i)} & \text{otherwise.} \end{cases} \tag{5}$$

This function ϕ satisfies the following property, which is also a well-known operation for establishing the Flow Decomposition Theorem [23, p. 177]:

Lemma 3.2. *Consider a vector $y^{(i)}$ that satisfies the flow conservation constraints* (2b) *and the integrality constraints* (2c) *for some decision diagram $D^{(i)}$. Consider an r-t path $p \in \mathcal{P}(D^{(i)})$ with $f_{\min} := \{y_a^{(i)} | a \in p\}$. Then the vector $y'^{(i)} := \phi(y^{(i)}, p, f)$ for any $f \in \mathbb{Z}, f \geq -f_{\min}$ also satisfies the flow conservation constraints* (2b) *and the integrality constraints* (2c).

Proof. Consider the vectors $y^{(i)}$ and $y'^{(i)}$ defined in this lemma. As $y^{(i)}$ satisfies the flow conservation constraints (2b) and the integrality constraints (2c), all constraints containing only arc variables $y_a^{(i)}$ for $a \in A \setminus p$ are satisfied for $y_a'^{(i)}$ as well. To complete the proof, we show that $y'^{(i)}$ is feasible for all flow conservation constraints and integrality constraints containing at least one variable related to an arc on P.

Consider any node $u \in N^{(i)} \setminus \{r, t\}$ with one incoming arc a^- that lies on p. Since p is a path, there must also be exactly one arc $a^+ \in p$ outgoing of u. Then the flow conservation constraint for u is as follows:

$$\begin{aligned}
\sum_{a \in \delta^-(u)} y_a'^{(i)} - \sum_{a \in \delta^+(u)} y_a'^{(i)} &= \sum_{a \in \delta^-(u) \setminus a^-} y_a'^{(i)} + y_{a^-}'^{(i)} - \sum_{a \in \delta^+(u) \setminus a^+} y_a'^{(i)} - y_{a^+}'^{(i)}, \\
&= \sum_{a \in \delta^-(u) \setminus a^-} y_a^{(i)} + \left(y_{a^-}^{(i)} + f\right) - \sum_{a \in \delta^+(u) \setminus a^+} y_a^{(i)} - \left(y_{a^+}^{(i)} + f\right), \\
&= \sum_{a \in \delta^-(u)} y_a^{(i)} - \sum_{a \in \delta^+(u)} y_a^{(i)} = 0.
\end{aligned}$$

the flow conservation constraints (2b) therefore also hold for $y'^{(i)}$.

Furthermore, $y_a'^{(i)} := y_a^{(i)} + f \geq y_a^{(i)} - f_{\min} \geq 0$ for all $a \in p_Z$ by definition of $f_{\min}$. Additionally, $y_a^{(i)}$ is integer by the integrality constraints (2c) and f is integer by definition, so $y_a^{(i)} - f =: y_a'^{(i)}$ is integer as well. As a result, $y_a'^{(i)} \in \mathbb{Z}_{\geq 0}$ for all $a \in p_Z$, and $y_a'^{(i)}$ satisfies the integrality constraints (2c) □

We firstly relate the objective value of solutions $y^{(i)} \in \text{Sol}(F^{(i)})$ to the cardinality of corresponding elements of $\text{Sol}(D^{(i)})$. We define an element $C^{(i)} \in \text{Sol}(D^{(i)})$ to be *corresponding* to $y^{(i)}$ if there exists a flow decomposition of $y^{(i)}$ into r-t paths that encode $C^{(i)}$ such that the paths have a positive integral flow.

Van Hoeve [16] shows that $\text{val}(y^{(i)}) = |C^{(i)}|$ for some $C^{(i)} \in \text{Sol}(D^{(i)})$ corresponding to $y^{(i)}$ for a specific flow decomposition algorithm. In general, flow decompositions are not unique, and as such, we wish to prove that this notion holds for any flow decomposition, regardless of the algorithm chosen.

This notion seems trivial for optimal solutions, but for non-optimal solutions, this is not immediately obvious, especially if those solutions have positive flow along all arcs of the zero-path p_Z; for those solutions, decompositions could consist of different numbers of r-t paths, or the decomposition may contain r-t paths that are not unique.

To facilitate the discussion, we introduce the following definition:

Definition 3.2 (Reduced solution)**.** A *reduced solution* is a vector y that is feasible for flow model (F) with $\min\{y_a : a \in p_Z\} = 0$.

For any reduced solution, any flow decomposition into r-t paths with positive integral flow cannot contain a zero-path as part of the decomposition, because p_Z is the unique zero-path according to Corollary 3.1.1. It also follows that optimal solutions are reduced solutions, since flow along p_Z can be subtracted from any non-reduced solution to obtain a feasible solution with a lower objective value.

The following holds for reduced solutions:

Lemma 3.3. *Consider a reduced solution $y^{(i)} \in Sol(F^{(i)})$ to flow model $(F^{(i)})$ for some decision diagram $D^{(i)}$. Then $y^{(i)}$ can be decomposed into val$(y^{(i)})$ many unique r-t paths with flow value 1. Furthermore, any flow decomposition of $y^{(i)}$ into r-t paths with flow value 1 consist of val$(y^{(i)})$ many unique r-t paths.*

Proof. We show that this result follows by induction on a sequence of vectors $y^{(i),j}$ generated by removing flow along paths.

Consider a reduced solution $y^{(i),0} \in \text{Sol}(F^{(i)})$, and let p_1 be an r-t path with $y_a^{(i),0} \geq 1$ for all $a \in p_1$. This path necessarily exists because $D^{(i)}$ is acyclic and because the 1-arc constraints (2a) and the flow conservation constraints (2b) hold for $y^{(i),0}$. Since $y^{(i),0}$ is reduced, p_1 must contain at least one 1-arc a_1. Moreover, a_1 has a flow of exactly 1 because $y^{(i),0}$ satisfies the 1-arc constraints (2a).

Let $y^{(i),1} = \phi(y^{(i)}, p_1, -1)$, i.e., $y^{(i),1}$ is the vector obtained by subtracting flow from $y^{(i),0}$. Then $\text{val}(y^{(i),1}) = \sum_{a \in \delta^+(r)} y_a^{(i),1} = \text{val}(y^{(i),0}) - 1$ and p_1 traverses exactly one arc in the first layer. By Lemma 3.2, $y_a^{(i),1}$ also satisfies the flow conservation constraints (2b) and the integrality constraints (2c).

Assume that, for any reduced solution $y^{(i),0}$, there exist distinct r-t paths $p_1, ..., p_j$, $j \leq \text{val}(y^{(i),0})-1$ such that the vector $y^{(i),j}$, obtained through the recursion $y^{(i),k} := \phi(y^{(i),k-1}, p_k, -1)$, $1 \leq k \leq j$, satisfies the flow conservation constraints (2b) and the integrality constraints (2c), and has objective value $\text{val}(y^{(i),j}) = \text{val}(y^{(i)}) - j$.

Because 1) $y^{(i),j}$ satisfies the flow conservation constraints (2b), 2) $D^{(i)}$ is acyclic, and 3) $\text{val}(y^{(i),j}) \geq 1$, there must exist an r-t path p_{j+1} with positive flow. Since $y^{(i)}$ is reduced, $p_{j+1} \neq p_Z$ and there must therefore be a 1-arc $a_{j+1} \in p_{j+1}$. From the integrality constraints (2c) and the 1-arc constraints (2a), it follows that $y_{a_{j+1}}^{(i),j} = 1$. As a result p_{j+1} has to be distinct from paths $p_1, ..., p_j$.

Let $y^{(i),j+1} := \phi(y^{(i),j}, p_{j+1}, -1)$. For the same reasons as for p_1 in the base case, p_{j+1} traverses only one arc in the first layer, and therefore $\text{val}(y^{(i),j+1}) = \text{val}(y^{(i),j}) - 1 = (y^{(i)}) - (j+1)$. Furthermore, since $y^{(i),j}$ satisfies the flow conservation constraints (2b) and the integrality constraints (2c), then so must $y^{(i),j+1}$ by Lemma 3.2. This concludes the induction step.

For $k = \text{val}(y^{(i),0})$, it follows that $\text{val}(y^{(i),k}) = 0$. Then, by the flow conservation constraints (2b) and the integrality constraints (2c), this implies that $y^{(i),k} = \mathbf{0}$, and that we can only decompose a reduced solution $y^{(i),0}$ into exactly $\text{val}(y^{(i),0})$ many unique r-t paths with flow value 1. Furthermore, the presented arguments hold for an arbitrary choice of r-t paths, so any flow decomposition of a reduced solution $y^{(i),0}$ into r-t paths with flow value 1 consist of $\text{val}(y^{(i),0})$ many unique r-t paths. □

Since the labels in each r-t path p_j encode a variable assignment X_j, a consequence of this lemma is that any reduced solution $y^{(i)}$ corresponds to elements of $\text{Sol}(D^{(i)})$ that all have a cardinality equal to the objective value of $y^{(i)}$. Furthermore, using this result, we can show the following for non-reduced solutions:

Corollary 3.3.1. *An integral flow decomposition of any non-reduced solution $y^{(i)}$ consists of at least val$(y^{*(i)}) + 1$ many r-t paths.*

Proof. Let $f_{\min} = \min\{y_a^{(i)} : a \in p_Z\}$, with $f_{\min} > 0$ as $y^{(i)}$ is non-reduced. Note that $f_{\min}$ is integer by the integrality constraints (2c). Then, let $y'^{(i)} := \phi(y^{(i)}, p_Z, -f_{\min})$. Since $y^{(i)}$ satisfies the flow conservation constraints (2b) and the integrality constraints (2c), by Lemma 3.2, so does $y'^{(i)}$. Furthermore, $y^{(i)}$ satisfies the 1-arc constraints (2a), and since $y_a'^{(i)}$ and $y_a^{(i)}$ have the same value on any $a \in A$ with $\ell(a) = 1$, $y_a'^{(i)}$ must satisfy the 1-arc constraints (2a). We conclude that $y'^{(i)}$ is feasible for flow model $(F^{(i)})$.

$y'^{(i)}$ has objective value $\text{val}(y'^{(i)}) = \text{val}(y^{(i)}) - f_{\min}$. Furthermore, by construction, this vector has an arc $a \in p_Z$ with $y_a'^{(i)} = 0$. It is therefore reduced, and any flow decomposition of $y'^{(i)}$ into r-t paths consists of $\text{val}(y'^{(i)}) = \text{val}(y^{(i)}) - f_{\min}$ many r-t paths by Lemma 3.3. As a result, $y^{(i)}$ can be decomposed into $\text{val}(y^{(i)}) - f_{\min}$ many unique r-t paths with flow value 1 plus p_Z with a flow value of $f_{\min}$. This decomposition yields a total of $\text{val}(y^{(i)}) - f_{\min} + 1 = \text{val}(y'^{(i)}) + 1 \geq \text{val}(y^{*(i)}) + 1$ many r-t paths.

Consider a decomposition of $y^{(i)}$ with a flow $f' \in \mathbb{Z}_{\geq 0}$ such that $f' < f_{\min}$ along p_Z. Then any other r-t path in the decomposition traverses a 1-arc, so every other r-t path in the decomposition has a flow value of at most 1. The decomposition then consists of $\text{val}(y^{(i)}) - f' + 1 > \text{val}(y^{(i)}) - f_{\min} + 1 \geq \text{val}(y^{*(i)}) + 1$ many r-t paths. □

The implication of this result is that all elements of $\text{Sol}(D^{(i)})$ that correspond to a non-reduced solution cannot be a smallest cardinality elements of $\text{Sol}(D^{(i)})$.

Conversely, we can relate the cardinality of any $C \in \text{Sol}(D^{(i)})$ to the objective value of some corresponding solution $y^{(i)} \in \text{Sol}(F^{(i)})$:

Lemma 3.4. *Any $C \in Sol(D^{(i)})$ corresponds to a vector $y^{(i)}$ that is feasible for flow model $(F^{(i)})$ with $|C| = val(y^{(i)})$.*

Proof. Consider any $C \in \text{Sol}(D^{(i)})$, and denote its elements by $X_1, X_2, ..., X_c$. By definition of $\text{Sol}(D^{(i)})$, every X_j, $j = 1, 2, ..., c$ is an element of $\mathcal{X}(D^{(i)})$ and is therefore encoded by an r-t path on $D^{(i)}$. Let p_j denote the r-t path on $D^{(i)}$ corresponding to X_j. The following procedure creates a vector $y^{(i)}$ corresponding to C:

1. Initialize $y_a^{(i)} := 0$ for all $a \in A^{(i)}$.
2. For $j = 1, 2, ..., c$, set $y^{(i)} := \phi(y^{(i)}, p_j, 1)$.

We show that $|C| = \text{val}(y^{(i)})$ holds and that the vector $y^{(i)}$ is feasible for flow model $(F^{(i)})$.

The initial vector with $y_a^{(i)} := 0$ for all $a \in A^{(i)}$ satisfies the flow conservation constraints (2b) and the integrality constraints (2c). Then, by Lemma 3.2, after setting $y^{(i)}(p_1) := \phi(y^{(i)}, p_1, 1)$, it follows that $y^{(i)}$ still satisfies the flow conservation constraints (2b) and the integrality constraints (2c) since in this instance, $f = -1 < 0 \leq f_{\min}$. Repeating this operation for paths $p_2, p_3, ..., p_c$ yields a vector $y^{(i)}$ that still satisfies the flow conservation constraints (2b) and the integrality constraints (2c).

The procedure transforms C into $|C|$ r-t paths p_j on $D^{(i)}$, and the flow through all arcs traversed on any p_j is increased by 1 for all p_j. Because $D^{(i)}$ is layered and acyclic, each r-t path has exactly one arc in the first layer. Therefore, the sum of flows through all arcs in the first layer is equal to the number of r-t paths, i.e.:

$$\text{val}(y^{(i)}) = \sum_{a \in \delta^+(r)} y_a^{(i)} = |C|$$

Consider any layer on $D^{(i)}$ corresponding to a node $v \in V$. By definition of $\text{Sol}(D^{(i)})$, there is only one variable assignment $X_j \in C$ that has $x_v = 1$ for $x_v \in X_j$. As a result, for each layer, there is exactly one path p_j with a 1-arc in that layer, and the flow through that arc is 1 by construction of $y^{(i)}$. Therefore, the 1-arc constraints (2a) are satisfied, and $y^{(i)}$ is therefore feasible for flow model $(F^{(i)})$. □

As a last step for establishing monotonicity of the objective function, we show that $\text{Sol}(D^{(i+1)}) \subset \text{Sol}(D^{(i)})$ in the following lemma:

Lemma 3.5. *Consider any non-terminal iteration i. Then $Sol(D^{(i+1)}) \subset Sol(D^{(i)})$*

Proof. From Lemmata 2.1 and 2.2 we conclude that the separation algorithm (Algorithm 2) removes at least one label-specified path and does not introduce new label-specified paths. Since each r-t path on $D^{(i)}$ corresponds to a variable assignment, it follows that $\mathcal{X}^{(i+1)} \subset \mathcal{X}^{(i)}$.

Consider any $C \in \text{Sol}(D^{(i+1)})$, and recall that $C \subseteq \mathcal{X}^{(i+1)}$. By the previous result, it follows that $C \subset \mathcal{X}^{(i)}$. This means that $C \in \text{Sol}(D^{(i)})$, and it follows that all elements of $\text{Sol}(D^{(i+1)})$ are also in $\text{Sol}(D^{(i)})$.

Let $X \in \mathcal{X}^{(i)} \setminus \mathcal{X}^{(i+1)}$ denote the variable assignment encoded by the removed r-t path. Since this conflict was detected, X was part of an infeasible color assignment, and there exists some $C' \in \text{Sol}(D^{(i)})$ such that $X \in C'$. But since $X \notin \mathcal{X}^{(i+1)}$, it follows that $C' \notin \text{Sol}(D^{(i+1)})$. □

Using the results established in this section, we can now show monotonicity:

Theorem 3.1. *val*$(y^{*(i)})$ *is monotonically increasing in* i.

Proof. Consider some iteration i of Algorithm 3. Let $y^{*(i)} \in \text{Sol}(F^{(i)})$ denote an optimal solution and let $C'^{(i)}$ denote any element of $\text{Sol}(D^{(i)})$ corresponding to $y^{*(i)}$ (which exists by Lemma 3.3). Furthermore, in the successive iteration, let $C^{*(i+1)}$ denote a smallest cardinality element of $\text{Sol}(D^{(i+1)})$, and let $y'^{(i+1)}$ denote the corresponding solution vector (which exists by Lemma 3.4).

By definition, $\text{val}(y^{*(i)}) \leq \text{val}(y^{(i)})\ \forall y^{(i)} \in \text{Sol}(F^{(i)})$. Consider any reduced solution $y^{(i)} \in \text{Sol}(F^{(i)})$. By Lemma 3.3, for any $C^{(i)}$ corresponding to $y^{(i)}$, $\text{val}(y^{(i)}) = |C^{(i)}|$, and it follows that $|C'^{(i)}| \leq |C^{(i)}|$ for all $C^{(i)}$ corresponding to reduced solutions. For any non-reduced solution $y^{(i)} \in \text{Sol}(F^{(i)})$, we can conclude using Corollary 3.3.1 that $|C'^{(i)}| < |C^{(i)}|$ for all $C^{(i)}$ corresponding to non-reduced solutions. It follows that $C'^{(i)}$ is a smallest cardinality element of $\text{Sol}(D^{(i)})$.

From Lemma 3.5, it follows that $C^{*(i+1)} \in \text{Sol}(D^{(i)})$. Then $|C'^{(i)}| \leq |C^{*(i+1)}|$ because $C'^{(i)}$ is a smallest cardinality element of $\text{Sol}(D^{(i)})$.

For any smallest cardinality $C^{*(i+1)} \in \text{Sol}(D^{(i+1)})$, it holds by definition that $|C^{*(i+1)}| \leq |C^{(i+1)}|$ for all $C^{(i+1)} \in \text{Sol}(D^{(i+1)})$. From Lemma 3.4, it follows that $|C^{*(i+1)}| = \text{val}(y'^{(i+1)})$. We can then conclude that $\text{val}(y'^{(i+1)}) \leq \text{val}(y^{(i+1)})\ \forall y^{(i+1)} \in \text{Sol}(F^{(i+1)})$, which means that $y'^{(i+1)}$ is an optimal solution to flow model $(F^{(i+1)})$.

Combining these results, we find that $\text{val}(y^{*(i)}) = |C'^{(i)}| \leq |C^{*(i+1)}| = \text{val}(y'^{(i+1)}) = \text{val}(y^{*(i+1)})$. We conclude that $\text{val}(y^{*(i)})$ is monotonically increasing in i. □

3.3 Proof of Theorem 3.2

The optimal objective value can only grow between two successive iterations as a result of Theorem 3.1. We show that this growth is bounded in the following theorem, which is the main result of this chapter:

Theorem 3.2. *If Algorithm 3 detects a conflict for an optimal solution* $y^{*(i)}$ *to flow model* $(F^{(i)})$ *in some iteration* i*, then for iteration* $i+1$*,* val$(y^{*(i+1)})$ = val$(y^{*(i)})$ *or* val$(y^{*(i+1)})$ = val$(y^{*(i)})+1$.

Proof. Consider an optimal solution $y^{*(i)}$ to flow model $(F^{(i)})$ with objective value f, and suppose $y^{*(i)}$ was decomposed into r-t paths $p_1, ..., p_f$ where a conflict was detected along some path $p_c = (a_1, a_2, ..., a_n) \in \mathcal{P}(D^{(i)})$. Such a decomposition necessarily exists as a consequence of Lemma 3.3. Consider the path $(a_j, ..., a_{k-1}, a_k)$ with $\ell(a_k) = 1$ along which the conflict was detected. Lemma 2.1 states that, after application of the separation algorithm (Algorithm 2), the label-specified path $(\ell(a_j), ..., \ell(a_{k-1}), \ell(a_k))$ with $\ell(a_k) = 0$ still exists in iteration $i+1$, and all other label-specified r-t paths in the flow decomposition of $y^{*(i)}$ remain unchanged; only the 1-arc a_k is removed, which is part of only 1 r-t path in the decomposition by the 1-arc constraints (2a).

Let p'_c denote the r-t path on $D^{(i+1)}$ defined by arcs $(a'_1, a'_2, ..., a'_n)$ such that $\ell(a'_h) = \ell(a_h)$ for $h = 1, ..., k-1, k+1, ..., n$ and $\ell(a'_k) = 0$. Consider a new vector $y'^{(i+1)}$ obtained as follows:

1. Initialize $y'^{(i+1)}_a := 0$ for all $a \in A^{(i+1)}$.
2. For $j = 1, ..., c-1, c+1, ..., f$, set $y'^{(i+1)} := \phi(y'^{(i+1)}, p_j, 1)$.
3. Set $y'^{(i+1)} := \phi(y'^{(i+1)}, p'_c, 1)$.

Then $\text{val}(y'^{(i+1)}) = \text{val}(y^{*(i)})$. Note that $y'^{(i+1)}$ satisfies the integrality constraints (2c) and the flow conservation constraints (2b) because of Lemma 3.2. Furthermore, since $y^{*(i)}$ satisfies all 1-arc constraints (2a), $y'^{(i+1)}$ is feasible for all 1-arc constraints except for the 1-arc constraint in layer k.

Consider the r-t path $p_{k=1} = (a_1, ..., a_k, ..., a_n)$ with $\ell(a_k) = 1$ and $\ell(a_l) = 0$ for $l = 1, ..., k-1, k+1, ..., n$. This path necessarily exists as a consequence of Corollary 3.1.2. Then, let $y^{(i+1)} := \phi(y'^{(i+1)}, p_{k=1}, 1)$. This new vector satisfies the 1-arc constraints (2a) in layer k, and the above operation does not influence 1-arc constraints in other layers since only flow through a 0-arc in those layers is changed. Furthermore, the flow conservation constraints (2b) and the integrality

constraints (2c) are satisfied for $y'^{(i+1)}$, and therefore also for $y^{(i+1)}$ by Lemma 3.2. As a result, there exists a solution $y^{(i+1)}$ to flow model $(F^{(i+1)})$ with $\text{val}(y^{(i+1)}) = \text{val}(y^{*(i)}) + 1$.

By monotonicity of the objective function (as established in Theorem 3.1), $\text{val}(y^{*(i+1)}) \geq \text{val}(y^{*(i)})$ and since the flow is integral, an optimal solution $y^{*(i+1)}$ can only have value $\text{val}(y^{*(i+1)}) = \text{val}(y^{*(i)})$ or $\text{val}(y^{*(i+1)}) = \text{val}(y^{*(i)}) + 1$. □

4 Chvátal-Gomory cut reusage

As mentioned in Observation 3.1, nearly all observed facet-defining inequalites for flow model (F) were r-t cuts. Furthermore, we showed that those r-t cuts bear a strong relation with objective cuts on decision diagrams. Because we did not find any patterns in the remaining facet-defining inequalities, we approached the strengthening of flow model (F) from another angle: general purpose cutting planes. We theorized that the same trick of using information of previous iterations could to these cutting planes, and therefore formulated the following question:

> **(Q2)** *How can cutting planes for the constrained network flow integer linear program in some iteration of van Hoeve's graph coloring algorithm be reused for successive iterations?*

To answer this question, we considered the usage of Chvátal-Gomory cutting planes, which are defined as follows:

Definition 4.1. Let $P = \{y \in \mathbb{R} : By \geq e\}$ be a polyhedron, and let $\alpha^\intercal y \geq \beta$ be valid for P, where $\alpha \in \mathbb{Z}^n$. Then a *Chvátal-Gomory cutting plane* (CG-cut) is an inequality $\alpha^\intercal y \geq \lceil \beta \rceil$ that is valid for $P \cap \mathbb{Z}^n$.

These types of cutting planes can strengthen formulations by cutting off non-integer elements of the polyhedron P if $\beta < \lceil \beta \rceil$ and $P \cap \{y : \alpha^\intercal y = \beta\} \neq \emptyset$, but they remain valid for the integer hull.

One method for finding CG-cuts is through the use of *multiplier vectors* $u \in [0, 1)^m$, as evidenced by the following lemma:

Lemma 4.1 (Adaptation of Lemma 5.13 by [12])**.** *Let $P = \{y \in \mathbb{R} : By \geq e\}$ be a polyhedron with $B \in \mathbb{Z}^{m \times n}$ and $e \in \mathbb{R}^m$. Then every CG cut for P can be expressed as $u^\intercal By \geq \lceil u^\intercal e \rceil$ for some $\mathbf{0} \leq u^\intercal < \mathbf{1}$ such that $u^\intercal B \in \mathbb{Z}^n$, or is implied by such an inequality and $By \geq e$*

As a consequence of this lemma, we can find CG-cuts by finding multiplier vectors u such that $uB \in \mathbb{Z}^n$. Preferably, we wish to find a multiplier vector such that the corresponding CG-cut removes as much of the polyhedron P as possible, but finding such cuts is generally difficult. Instead, we compute a fractional solution $\hat{y}$, and attempt to find a CG-cut that cuts off this solution such that the distance between the cutting plane and the point $\hat{y}$ is maximized. For such a CG-cut $\alpha^\intercal y \geq \lceil \beta \rceil$, we measure this distance by the *violation*, which is equal to $\lceil \beta \rceil - \alpha^\intercal \hat{y}$. Finding such cuts can be expensive, but if a CG-cut in some iteration can be reused in successive iterations, we can pay the initial computational price of generating such a CG-cut to win on time in future iterations. Such an approach would require an efficient algorithm for cheaply generating reused CG-cuts.

In this chapter, we derive a scheme for reusing CG cuts in successive iterations. In Section 4.1, we show an MILP-formulation that finds a CG-cut $\alpha^\intercal y \geq \lceil \beta \rceil$ that cuts off some fractional point $\hat{y}$ with maximum violation. In Section 4.2, we show a method for reusing CG-cuts and some structural insight about the method. In Section 4.3, we develop an efficient algorithm for reusing CG-cuts. In Section 4.4, we show our experimental results on the efficacy of the cut reusage scheme, and we discuss those results in Section 4.5.

To aid our discussions, we introduce some additional notation. We let $\text{fr}(x) = x - \lfloor x \rfloor$ be the fractional part of some scalar x. Additionally, for some decision diagram $D = (N, A)$, we let $A_1 = \{a \in A : \ell(a) = 1\}$ be the set of 1-arcs $A_0 = \{a \in A : \ell(a) = 0\}$ be the set of 0-arcs.

4.1 Maximum violation CG-cuts for P^{flow}

In this section, we show a method for finding maximum violation CG-cuts. Consider any binary decision diagram $D = (N, A)$ with set of layers $\mathcal{L}$. Let $0 < \varepsilon < 1$ and let $\hat{y}$ be some fractional optimal solution to the LP relaxation (F^{rel}). We introduce the following mixed-integer linear

program (MILP) for finding maximum violation CG-cuts given this fractional optimal solution:

$$\begin{aligned}
&\min \sum_{a\in A} \hat{y}_a\mu_a + \sum_{a\in A_1} \hat{y}_a\pi_{L(a)} - \gamma \\
\text{s.t. } &\alpha_a = \pi_j + \lambda_u - \lambda_v + \mu_a && \forall a=(u,v)\in A_1, L(a)=j, && \text{(6a)}\\
&\alpha_a = \lambda_u - \lambda_v + \mu_a && \forall a=(u,v)\in A_0, && \text{(6b)}\\
&\gamma - \sum_{j\in L}\pi_j \le 1-\varepsilon, && && \text{(6c)}\\
&\lambda_r = \lambda_t = 0, && && \text{(6d)}\\
&\alpha_a \in \mathbb{Z} && \forall a\in A, && \text{(6e)}\\
&\gamma \in \mathbb{Z}, && && \text{(6f)}\\
&\pi_j \in [0,1) && \forall j\in\mathcal{L}, && \text{(6g)}\\
&\lambda_u \in [0,1) && \forall u\in N, && \text{(6h)}\\
&\mu_a \in [0,1) && \forall a\in A. && \text{(6i)}
\end{aligned}$$

The correctness of this formulation is shown in the following theorem:

Theorem 4.1. *Consider a decision diagram $D=(N,A)$ with set of layers $\mathcal{L}$. Let $A=A_1\cup A_0$, with A_1 being the set of 1-arcs and A_0 being the set of 0-arcs. Furthermore, let $0<\varepsilon<1$ and let $\hat{y}$ be a solution to the LP relaxation (F^{rel}) on D. Then the MILP-formulation (6) yields a CG-cut $\alpha^\intercal y\ge\gamma$ with maximum violation of at most $1-\varepsilon$ with respect to $\hat{y}$.*

Proof. Consider Formulation (3) (the LP relaxation (F^{rel}). From Lemma 4.1, it follows that, for any vector u such that $\mathbf{0}\le u<\mathbf{1}$ and $u^\intercal B\in\mathbb{Z}^A$, the inequality $u^\intercal B\ge\lceil u^\intercal e\rceil$ is a CG-cut.

We firstly construct such a vector u by splitting it into four parts, depending on which rows of B those parts are multiplied with: 1) $\pi\in[0,1)^{\mathcal{L}\times 1}$ for Constraints (3a), 2) $\lambda^+\in[0,1)^{(N\setminus\{r,t\})\times 1}$ for Constraints (3b) (with the dimension being $(N\setminus\{r,t\})\times 1$ because there are no flow conservation Constraints for the root and sink node), 3) $\lambda^-\in[0,1)^{(N\setminus\{r,t\})\times 1}$ for Constraints (3c) and 4) $\mu\in[0,1)^{A\times 1}$ for Constraint (3d).

We then let $u'^\intercal=(\pi^\intercal,(\lambda^+)^\intercal,(\lambda^-)^\intercal,\mu^\intercal)$, which by construction meets the requirement that $\mathbf{0}\le u'<\mathbf{1}$. For $u'^\intercal B\ge\lceil u'^\intercal e\rceil$ to be valid for P^{Ch}, it is required that $u'^\intercal B\in\mathbb{Z}^A$. To that end, we introduce a vector $\alpha\in\mathbb{Z}^A$ such that $\alpha^\intercal=u'^\intercal B$, i.e.:

$$\alpha^\intercal=\pi^\intercal\Lambda+(\lambda^+)^\intercal\mathrm{In}(D)-(\lambda^-)^\intercal\mathrm{In}(D)+\mu^\intercal I_m.$$

We then define $\lambda':=(\lambda^+-\lambda^-)$ with $\lambda'\in(-1,1)^{(N\setminus\{r,t\})\times 1}$ and simplify this equation to:

$$\alpha^\intercal=\pi^\intercal\Lambda+\lambda'^\intercal\mathrm{In}(D)+\mu^\intercal I_m. \tag{7}$$

We can further simplify this equation by firstly noting that $\Lambda_{j,a}=1$ if and only if $a\in A_1$ and $L(a)=j$. Then $\pi^\intercal\Lambda_{*,a_1}=\pi_j$ for any arc $a_1\in A_1$ that is a 1-arc in layer j, and $\pi^\intercal\Lambda_{*,a_0}=0$ for any 0-arc $a_0\in A_0$.

Secondly, because $\mathrm{In}(D)$ is the node-arc incidence matrix for all nodes except r and t, $\mathrm{In}(D)_{u,a}=1$ if and only if $u\in N\setminus\{r,t\}$ is the head of arc $a\in A$, and $\mathrm{In}(D)_{v,a}=-1$ if and only if $v\in N\setminus\{r,t\}$ is the tail of arc a. Therefore, for all arcs $a=(u,v)\in A$ with $u,v\in N\setminus\{r,t\}$, it follows that $\lambda'^\intercal\mathrm{In}(D)_{*,a}=\lambda'_u-\lambda'_v$. Conversely, for arcs $a_r=(r,v)\in A$, we find that $\lambda'^\intercal\mathrm{In}(D)_{*,a_r}=-\lambda'_v$ and for arcs $a_t=(u,t)\in A$, we have $\lambda'^\intercal\mathrm{In}(D)_{*,a_t}=\lambda'_u$. For compact notation, we introduce λ_r and λ_t with $\lambda_r=\lambda_t=0$, and we let $\lambda^\intercal=(\lambda_s,\lambda'^\intercal,\lambda_t)\in(-1,1)^{N\times 1}$.

Lastly, $\mu^\intercal I_m = \mu^\intercal$ and we conclude that equations (7) read:

$$\alpha_a = \pi_j + \lambda_u - \lambda_v + \mu_a \qquad \forall a = (u,v) \in A_1 \text{ s.t. } L(a) = j, \tag{8a}$$
$$\alpha_a = \lambda_u - \lambda_v + \mu_a \qquad \forall a = (u,v) \in A_0, \tag{8b}$$
$$\lambda_r = \lambda_t = 0, \tag{8c}$$
$$\alpha \in \mathbb{Z}^A, \tag{8d}$$
$$\pi \in [0,1)^{\mathcal{L}},\ \lambda \in (-1,1)^N,\ \mu \in [0,1)^A. \tag{8e}$$

Then for any $\alpha^\intercal = u'^\intercal B$ satisfying this system it follows that $\alpha^\intercal y \geq \lceil u'^\intercal e \rceil$ is a CG-cut. Since the system (8) matches Constraints (6a), (6b), (6d), (6e), (6g), (6h) and (6i), we conclude that Formulation (6) finds an α such that $\alpha^\intercal y \geq \lceil u'^\intercal e \rceil$ is a CG-cut.

Let $\lceil u'^\intercal e \rceil = \gamma \in \mathbb{Z}$. Then, for a CG-cut with a violation of at most $1 - \varepsilon$, it should hold that:

$$\begin{aligned} 1 - \varepsilon &\geq \gamma - u'^\intercal e \\ &= \gamma - \pi^\intercal \mathbf{1} - \lambda^+ \mathbf{0} - \lambda^- \mathbf{0} - \mu \mathbf{0} \\ &= \gamma - \sum_{j \in \mathcal{L}} \pi_j, \end{aligned}$$

which matches Constraint (6c). Then Formulation (6) finds an α such that $\alpha^\intercal y \geq \lceil u'^\intercal e \rceil$ is a CG-cut with a violation of at most $1 - \varepsilon$.

A CG-cut $a^\intercal y \geq \gamma$ with the largest violation can be found by minimizing $\alpha^\intercal y - \gamma$. Substituting equations (8a) and (8b) in this expression, we obtain:

$$\begin{aligned} \alpha^\intercal \hat{y} - \gamma &= \sum_{a \in A_0} \alpha_a \hat{y}_a + \sum_{a \in A_1} \alpha_a \hat{y}_a - \gamma, \\ &= \sum_{a=(u,v) \in A_0} (\lambda_u - \lambda_v + \mu_a)\, \hat{y}_a \ + \sum_{a=(u,v) \in A_1} \left(\pi_{L(a)} + \lambda_u - \lambda_v + \mu_a\right) \hat{y}_a - \gamma, \\ &= \sum_{a \in A} \mu_a \hat{y}_a + \sum_{a \in A_1} \pi_{L(a)} \hat{y}_a \ + \sum_{a=(u,v) \in A} \lambda_u \hat{y}_a \ - \sum_{a=(u,v) \in A} \lambda_v \hat{y}_a - \gamma, \\ &= \sum_{a \in A} \mu_a \hat{y}_a + \sum_{a \in A_1} \pi_{L(a)} \hat{y}_a + \sum_{u \in V} \sum_{a \in \delta^+(u)} \lambda_u \hat{y}_a - \sum_{v \in V} \sum_{a \in \delta^-(v)} \lambda_v \hat{y}_a - \gamma. \end{aligned} \tag{9}$$

Note that $\hat{y} \in P$, and therefore $\hat{y}$ satisfies the flow conservation constraints (2b). As a result, for any $u \in N \setminus \{r, t\}$:

$$\sum_{a \in \delta^+(u)} \lambda_u \hat{y}_a - \sum_{a \in \delta^-(u)} \lambda_u \hat{y}_a = \lambda_u \left(\sum_{a \in \delta^+(u)} \hat{y}_a - \sum_{a \in \delta^-(u)} \hat{y}_a \right) = 0.$$

Furthermore, recall that $\lambda_t = \lambda_r = 0$. Combining these two facts, Equation (9) reduces to:

$$\alpha^\intercal \hat{y} - \gamma = \sum_{a \in A} \mu_a \hat{y}_a + \sum_{a \in A_1} \pi_{L(a)} \hat{y}(a) - \gamma. \tag{10}$$

Then a largest violation CG-cut can be found by minimizing $\sum_{a \in A} \mu_a \hat{y}_a + \sum_{a \in A_1} \pi_{L(a)} \hat{y}(a) - \gamma$, which matches the objective function of Formulation (6).

This result implies that the choice of λ does not affect the objective function directly, and only affects integrality of the Constraints (6a). For any λ_u, $u \in N$ valid for Constraints (6a) such that $1 < \lambda_u < 0$, it follows that $\lambda'_u := \lambda_u + 1$ is also valid for those constraints. Then any λ_u, $u \in N$ can be restricted to the domain $[0, 1)$, which completes Constraints (6h). □

4.2 Theoretical results on CG-cut reusage

As outlined in the introduction of this chapter, we attempt to devise a method that finds CG-cuts in some iteration and reuses those cuts efficiently in successive iterations. Recall that, through

Formulation (6), a CG-cut can be found by finding appropriate values of the vectors $\lambda \in [0,1)^N$, $\mu \in [0,1)^A$ and $\pi \in [0,1)^{\mathcal{L}}$ such that Constraints (6a), (6b) and (6d) are satisfied. However, finding a maximum violation CG-cut with this method is computationally expensive.

In each iteration of Algorithm 3, the number of arcs and nodes change, and so do the dimensions of λ and μ. However, the number of layers remains constant throughout the iterations. As such, we theorize that the π-values derived in some iteration i can be reused to generate cutting planes in iterations $j > i$. Suppose we then find an efficient algorithm to calculate corresponding μ- and λ-values to yield a maximum violation CG-cut. Then with an initial investment of solving Formulation (6), we can efficiently generate CG-cuts in successive iterations. In this section, we present theoretical insights that substantiate the design of such an algorithm.

Suppose, in some iteration, we solved the LP relaxation (F^{rel}) on a decision diagram $D = (N, A)$ and obtained a fractional optimal solution $\hat{y}$. Furthermore, assume that we obtained some vector π by solving Formulation (6) in a previous iteration, with $\pi_j > 0$ for some $j \in L$. To simplify notation, we introduce the vector $\hat{\pi}$, which we define as follows:

$$\hat{\pi}_a = \begin{cases} \pi_j & \text{if } a \text{ is a 1-arc with } L(a) = j, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

This vector allows us to write Constraints (8b) and (8a) in a single sentence. Note that, since $\hat{\pi}$ is now fixed, then γ is known as well by Constraints (6c) and (6f). As a result, we can rewrite Formulation (6) to:

$$\begin{aligned} & \min \sum_{a \in A} \hat{y}_a \mu_a & & \\ \text{s.t. } & \hat{\pi}_{(u,v)} + \lambda_u - \lambda_v + \mu_{(u,v)} \in \mathbb{Z} & \forall (u,v) \in A, & \quad (12a) \\ & \lambda_r = \lambda_t = 0, & & \quad (12b) \\ & \lambda_u \in [0,1) & \forall u \in N, & \quad (12c) \\ & \mu_a \in [0,1) & \forall a \in A. & \quad (12d) \end{aligned}$$

With this new formulation, the problem of finding a CG-cut $\alpha^\intercal \hat{y} \geq \gamma$ given $\hat{\pi}$ transforms to a labelling problem where each node $u \in N$ is assigned a label $\lambda_u \in [0,1)$ and each arc $a \in A$ is assigned a label $\mu_a \in [0,1)$ such that Constraints (12a) are satisfied. See 6 for an illustration of this analogy to a labelling problem.

Because only the μ-values incur costs in this formulation, λ should be chosen such that integrality in Constraints (12a) is maintained while simultaneously avoiding $\mu > 0$.

A first important insight about Formulation (12) can be obtained by reconsidering its objective function. Recall from the proof of Theorem 4.1 that the objective function of Formulation (6) is a simplification of $\min \alpha^\intercal y - \gamma$. But then the objective function of Formulation (12) is essentially given by $\min \alpha^\intercal y$, as the term γ is fully dependent on $\hat{\pi}$, which is now a constant. What we learn from this realization is that our only objective for Formulation (12) is to keep α as small as possible—though of course with respect to the "costs" $\hat{y}$.

Since $\hat{\pi}$, μ and λ are all restricted to the domain $[0,1)$, it follows that $\alpha_{(u,v)} = \hat{\pi}_{(u,v)} + \lambda_u - \lambda_v + \mu_{(u,v)} \in \{0,1,2\}$. Furthermore, any arc (u,v) with $\hat{\pi}_{(u,v)} + \lambda_u > \lambda_v$ means that $\mu_{(u,v)}$ should be chosen to be larger than 0 to maintain integrality. But this immediately results in an $\alpha_{(u,v)} \geq 1$, which increases the objective function and is therefore undesirable. We refer to the situation that $\hat{\pi}_{(u,v)} + \lambda_u > \lambda_v$ as a *jump*. This leads us to our first insight about Formulation (12):

Observation 4.1. *To optimize Formulation* (12)*, λ should be chosen such that jumps—situations with $\hat{\pi}_{(u,v)} + \lambda_u > \lambda_v$—are avoided as much as possible.*

Note that a jump is sometimes unavoidable; in the case of $\hat{\pi}_{(u,v)} + \lambda_u > 1$, it follows that $\alpha_{(u,v)} > 0$ regardless of the choice of λ_v.

A second insight about Formulation (12) follows from formulating Constraints (12a) as $\mu_a = \text{fr}(\lambda_v - \lambda_u - \hat{\pi}_{(u,v)})$ for all $(u,v) \in A$. By substituting these reformulated constraints into the

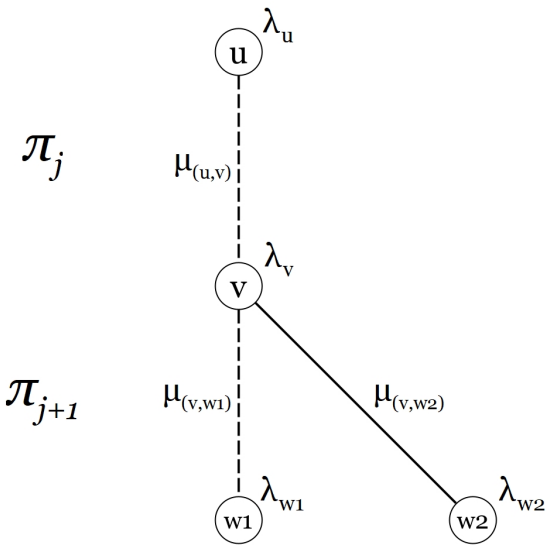


Figure 6: Subgraph of some decision diagram D to illustrate the physical interpretation of λ, μ and π. Note that $\hat{\pi}$ is related to the arcs, but π, the vector that is used to create $\hat{\pi}$, is related to the layers as depicted in this example.

objective function, we obtain:

$$g(\lambda) = \sum_{(u,v)\in A} \hat{y}_{(u,v)} \text{fr}\left(\lambda_v - \lambda_u - \hat{\pi}_{(u,v)}\right). \tag{13}$$

This substitution can be used to prove the following property of Formulation (12):

Theorem 4.3. *Consider a decision diagram $D = (N, A)$. Let $\hat{y}$ be a fractional optimal solution to the LP relaxation (F^{rel}), and let $\hat{\pi} \in [0, 1)^{A\times 1}$ be some vector with $\hat{\pi}_a = 0$ for all $a \in A_0$. Then there exists a solution (λ^*, μ^*) that minimizes Formulation (12) such that, for all $v \in N \setminus \{r, t\}$, $\lambda^*_v = fr\left(\lambda^*_{u_*} + \hat{\pi}_{(u_*,v)}\right)$ for some direct predecessor u_* of v.*

The relevance of Theorem 4.3 may not be immediately clear. As such, we will first provide an interpretation of this theorem, and we prove the theorem afterwards.

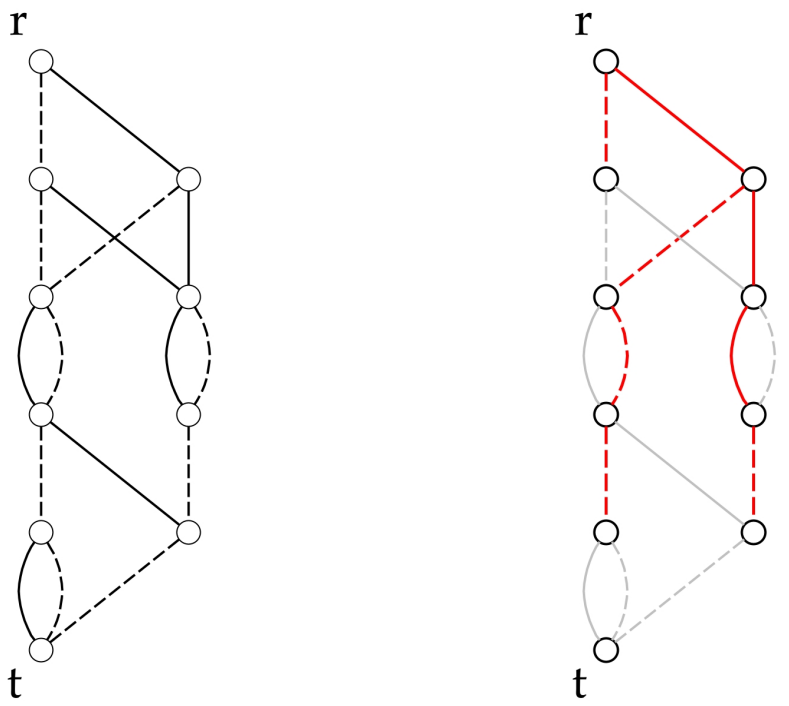
4.2.1 Implications of Theorem 4.3

Consider some decision diagram $D = (N, A)$, some $\hat{\pi} \in [0, 1)^A$ with $\hat{\pi}_a = 0$ for all $a \in A_0$ and some fractional optimal solution to the LP relaxation (F^{rel}) $\hat{y}$. We define a *predecessor labelling* to be any solution (λ, μ) to Formulation (12) such that, for all $v \in N \setminus \{r, t\}$, $\lambda_v = \text{fr}\left(\lambda_{u_*} + \hat{\pi}_{(u_*,v)}\right)$ for some direct predecessor u_* of v. Essentially, v inherits the label from this predecessor u_*. Theorem 4.3 states that there exists an optimal solution to Formulation (12) that is a predecessor label labelling. Therefore, an optimal solution to Formulation (12) can be found by only considering all predecessor labellings. For a decision diagram $D = (N, A)$ with a finite number of arcs, this means that the solution space for Formulation (12) can be truncated to a finite solution space.

We can even obtain results on the size of this solution space by establishing the link between predecessor labellings and r-arborescences. For any decision diagram $D = (N, A)$, an *r-arborescence* is an acyclic subgraph (N, T) of D such that each $v \in N \setminus \{r\}$ has exactly one incoming arc. An r-arborescence can be seen as the directed analogue of a rooted spanning tree. See Figure 7 for an example of such an r-arborescence.

For any predecessor labelling (λ', μ') that is a solution to Formulation (12), each node $v \in N \setminus \{r, t\}$ inherits a node label from exactly one direct predecessor. As a result, we can construct an r-arborescence $(N \setminus \{t\}, T)$ by including only the arcs over which each node $v \in N \setminus \{r, t\}$ inherits a node label, i.e.:

$$T = \left\{(u, v) \in A : \lambda'_v = \text{fr}\left(\lambda'_u + \hat{\pi}_{(u,v)}\right),\ |\delta^-(v)| = 1\ \forall v \in N \setminus \{r, t\}.\right\}$$



(a) *Example decision diagram*

(b) *R-arborescence on the example decision diagram*

Figure 7: An example decision diagram $D = (N, A)$ (7a) and an example of an r-arborescence $(N \setminus \{t\}, T)$, where all arcs $a \in T$ are colored red. Each node $N \setminus \{r, t\}$ has an incoming arc. These arcs represent from which ancestor each node "inherits" their node label.

The sink-node t is excluded from this r-arborescence because it has label $\lambda_t = 0$ by default.

Equivalently, for any r-arborescence $(N \setminus \{t\}, T)$ on D, we can set $\lambda_v := \text{fr}\left(\lambda'_u + \hat{\pi}_{(u,v)}\right)$ (starting from the root node r and updating labels layer by layer) to obtain a predecessor label labelling that is a solution to Formulation (12). If we then find an r-arborescence with a corresponding λ that minimizes $g(\lambda)$—the reformulated objective function of Formulation (12)—we find an optimal solution to Formulation (12). Consequently, an optimal solution to the following problem is also an optimal solution to Formulation (12):

Problem 4.1 (Arborescence labelling problem)**.**

Input: a decision diagram $D = (N, A)$, a vector $\hat{\pi} \in [0, 1)^A$ with $\hat{\pi}_a = 0$ for all $a \in A_0$, and a fractional optimal solution $\hat{y}$ to the LP relaxation (F^{rel}).

Feasible solutions: acyclic subgraphs $(N \setminus \{t\}, T)$ of $D = (N, A)$ such that $T \subseteq A$ and every $v \in N \setminus \{r, t\}$ has exactly one incoming arc.

Goal: find such an acyclic subgraph $(N \setminus \{t\}, T)$ that minimizes $g(\lambda)$ with $\lambda_v := \text{fr}\left(\lambda'_u + \hat{\pi}_{(u,v)}\right)$ for all $(u, v) \in T$.

The solution set of Problem 4.1 is equal to the number of distinct r-arborescences on D. Lemma 4.2 gives an upper bound to the number of r-arborescences on D. As a result, it also provides an upper bound on the size of the "truncated" solution space of Formulation (12):

Lemma 4.2. *The number of distinct r-arborescences $(N \setminus \{t\}, T)$ on some decision diagram $D = (N, A)$ is less than $2^{|N|-2}$.*

Proof. Under any r-arborescence $(N \setminus \{t\}, T)$, each node $v \in N \setminus \{r, t\}$ has exactly one incoming arc. In each node $v \in N \setminus \{r, t\}$, there are $|\delta^-(v)|$ decisions that can be made for arcs to include in the r-arborescence, so the total number of distinct r-arborescences on D is $\prod_{v \in N \setminus \{r,t\}} |\delta^-(v)|$.

Recall that t is an initial node, which means that t has an incoming 0-arc by Lemma 3.1. Likewise, the 1-arc from the initial node that is a predecessor of t cannot be redirected to any other node because t is the sole node in its layer. It follows that $|\delta^-(t)| \geq 2$. Furthermore, since each node on D can have at most two outgoing arcs, it follows that:

$$\begin{aligned} 2|N| - 2 &\geq \sum_{v \in N \setminus \{t\}} |\delta^+(v)|, \\ &= \sum_{v \in N \setminus \{r\}} |\delta^-(v)|, \\ &\geq \sum_{v \in N \setminus \{r,t\}} |\delta^-(v)| + 2, \end{aligned}$$

or $2|N| - 4 \geq \sum_{v \in N \setminus \{r,t\}} |\delta^-(v)|$.

Consider the AM-GM inequality, which states that, for a list of non-negative numbers, the arithmetic mean is larger or equal than the geometric mean. Using this inequality and our previous result, we obtain:

$$\sqrt[|N|-2]{\prod_{v \in N \setminus \{r,t\}} |\delta^-(v)|} \leq \frac{1}{|N| - 2} \sum_{v \in N \setminus \{r,t\}} |\delta^-(v)| \leq \frac{1}{|N| - 2} (2|N| - 4) = 2.$$

It then follows that

$$\prod_{v \in N \setminus \{r,t\}} |\delta^-(v)| \leq 2^{|N|-2},$$

from which we conclude that the number of distinct r-arborescences $(N \setminus \{t\}, T)$ on some decision diagram $D = (N, A)$ is bounded by $2^{|N|-2}$. □

Theorem 4.2. *An optimal solution to Formulation* (12) *can be found in time* $\mathcal{O}(|N|2^{|N|})$.

Proof. For any r-arborescence $(N \setminus \{t\}, T)$ on some decision diagram $D = (N, A)$, calculating the costs per arc can be done $\mathcal{O}(1)$, and since there are at most $2|N| - 2$ arcs on a decision diagram, calculating $g(\lambda)$ for that r-arborescence can be done in $\mathcal{O}(|N|)$ time. By enumerating over all possible r-arborescences on D—which is done in time $\mathcal{O}(2^{|N|})$ by Lemma 4.2—calculating the costs per r-arborescence (time $\mathcal{O}(|N|)$) and storing the r-arborescence with the minimum cost (time $\mathcal{O}(|1|)$), we find an optimal solution to Problem 4.1 in time $\mathcal{O}(|N|2^{|N|})$. This r-arborescence can then be translated to a solution to Formulation (12). □

In conclusion, the important implications of Theorem 4.3 are that the Formulation (12) can be solved by finding a minimum cost r-arborescence with respect to $g(\lambda)$ (Problem 4.1). This truncates the solution space of Formulation (12) to a finite solution space. Furthermore, there exists an $\mathcal{O}(|N|2^{|N|})$ time algorithm for solving Formulation (12).

4.2.2 Proof of Theorem 4.3

Now that the relevance of Theorem 4.3 has been established, we show its proof. In order to prove this theorem, we define a function $\zeta_v(\lambda_v) : [0, 1) \longmapsto \mathbb{R}_{\geq 0}$ as follows:

$$\zeta_v(\lambda_v) = \sum_{(u,v) \in \delta^-(v)} \hat{y}_{(u,v)} \mathrm{fr}\left(\lambda_v - \lambda_u - \hat{\pi}_{(u,v)}\right) + \sum_{(v,w) \in \delta^+(v)} \hat{y}_{(v,w)} \mathrm{fr}\left(\lambda_w - \lambda_v - \hat{\pi}_{(v,w)}\right). \tag{14}$$

This function expresses all arc-costs in $g(\lambda)$ that are influenced by a change in λ_v. Essentially, it's the local cost-function of λ_v. This function $\zeta_v(\lambda_v)$ has the following form, arising from the fact that $\hat{y}$ satisfies the flow conservation constraints (2b):

Proposition 4.1. *Consider a node $v \in N$. Let $U_v \subseteq N$ be the set of direct predecessors of v with fixed labels $\lambda_u \in [0,1)$ for $u \in U_v$, and let $W_v \subseteq N$ be the set of direct successors of v with fixed labels $\lambda_w \in [0,1)$ for $w \in W_v$. Furthermore, let $\hat{\pi} \in [0,1)^{A\times 1}$ be some vector with $\hat{\pi}_a = 0$ for all $a \in A_0$, and let $\hat{y}$ denote a fractional solution to the LP relaxation (F^{rel}). Then $\zeta_v(\lambda_v)$ is a piece-wise constant function with breakpoints $(\lambda_u + \hat{\pi}_{u,v})$ for all $u \in U_v$ and $(\lambda_w - \hat{\pi}_{v,w})$ for all $w \in W_v$.*

Proof. Let $\rho_u = \text{fr}(\lambda_u + \hat{\pi}_{u,v})$ for $u \in U_v$, and let $\rho_w = \text{fr}(\lambda_w - \hat{\pi}_{w,v})$ for $w \in W_v$. Consider two nodes $v_1, v_2 \in U_v \cup W_v$ such that there is no other node v' such that $\rho_{v'}$ lies between ρ_{v_1}, i.e. ρ_{v_2} $\rho_{v_1} < \rho_{v_2}$ and $\rho_{v'} \notin (\rho_{v_1}, \rho_{v_2})$ for all $v' \in U_v \cup W_v$. Furthermore, consider any $\lambda'_v \in (\rho_{v_1}, \rho_{v_2})$ and any ε such that $0 < \varepsilon < \rho_{v_2} - \lambda'_v$.

By definition of ε and λ'_v, it holds that $\lambda'_v + \varepsilon < \rho_{v'}$ for any $\rho_{v'} > \lambda'_v$. As a result, $\text{fr}\,(\lambda'_v + \varepsilon - \rho_{v'}) = \text{fr}\,(\lambda'_v - \rho_{v'}) + \varepsilon$, and $\text{fr}\,(\rho_{v'} - \lambda'_v - \varepsilon) = \text{fr}\,(\rho_{v'} - \lambda'_v) - \varepsilon$. This and the fact that $\hat{y}$ satisfies the flow conservation constraints (2b), imply that:

$$\begin{aligned}
\zeta_v(\lambda'_v + \varepsilon) &= \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)} \text{fr}\,(\lambda'_v + \varepsilon - \lambda_u - \hat{\pi}_{u,v}) + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)} \text{fr}\,(\lambda_w - \lambda'_v - \varepsilon - \hat{\pi}_{v,w}),\\
&= \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)} \text{fr}\,(\lambda'_v - \rho_u) + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)} \text{fr}\,(\rho_w - \lambda'_v) + \varepsilon \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)} - \varepsilon \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)},\\
&= \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)} \text{fr}\,(\lambda'_v - \rho_u) + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)} \text{fr}\,(\rho_w - \lambda'_v) = \zeta_v(\lambda'_v),
\end{aligned}$$

from which we conclude that $\zeta_v(\lambda_v)$ is constant for $\lambda_v \in (\rho_{v_1}, \rho_{v_2})$.

Let $\rho_{\min} := \min\{\rho_{v'} : v' \in U_v \cup W_v\}$ and $\rho_{\max} := \max\{\rho_{v'} : v' \in U_v \cup W_v\}$. Then there is no $\rho_{v'}$, $v' \in U_v \cup W_v$ in the interval $(\rho_{\max}, 1 + \rho_{\min})$. It follows by the previous derivation that $\zeta_v(\lambda_v)$ is constant for $\lambda_v \in (\rho_{\max}, 1 + \rho_{\min})$. Furthermore, $\text{fr}\,(\lambda_v - (1 + \rho_{\min})) = \text{fr}\,(\lambda_v - \rho_{\min})$ and $\text{fr}\,(1 + \rho_{\min} - \lambda_v) = \text{fr}\,(\rho_{\min} - \lambda_v)$, and consequently, $\zeta_v(\lambda_v)$ is also constant for $\lambda_v \in (\rho_{\max}, 1) \cup [0, \rho_{\min})$. Then $\zeta_v(\lambda_v)$ is a piecewise constant function with breakpoints $\rho_u = (\lambda_u + \hat{\pi}_{u,v})$ for $u \in U_v$ and $\rho_w = (\lambda_w + \hat{\pi}_{w,v})$ for $w \in W_v$. □

The fact that $\zeta_v(\lambda_v)$ is a piecewise constant function can be used to prove the following property of $\zeta_v(\lambda_v)$:

Lemma 4.3. *Consider a node $v \in N$ and let $U_v \subseteq N$ be the set of direct predecessors of v with fixed labels $\lambda_u \in [0,1)$ for $u \in U_v$. Furthermore, let $\hat{\pi} \in [0,1)^{A\times 1}$ be some vector with $\hat{\pi}_a = 0$ for all $a \in A_0$, and let $\hat{y}$ denote a fractional solution to the LP relaxation (F^{rel}). Then $\lambda^*_v := fr\big(\lambda_{u_*} + \hat{\pi}_{(u_*,v)}\big)$ for some $u_* \in U_v$ minimizes $\zeta_v(\lambda_v)$.*

Proof. Let $W_v \subseteq N$ be the set of direct successors of v with fixed labels $\lambda_w \in [0,1)$ for $w \in W_v$. Furthermore, let $\rho_u = \text{fr}(\lambda_u + \hat{\pi}_{u,v})$ for $u \in U_v$ and let $\rho_w = \text{fr}(\lambda_w + \hat{\pi}_{w,v})$ for $w \in W_v$. As shown in Proposition 4.1, these values ρ_u and ρ_w are the breakpoints of the local cost function ζ_v. We prove this lemma by considering three consecutive values ρ_{v_1}, ρ_u, ρ_{v_2} and examining if the local cost function increases or decreases when going from ρ_{v_1} to ρ_{v_2}. We show that the local cost function decreases when going from ρ_{u_1} to ρ_{u_2} for predecessor nodes $u_1, u_2 \in U_v$ such that $\rho_{u_1} < \rho_{u_2}$. Subsequently, we show that the local cost function increases when going from ρ_{w_1} to ρ_{w_2} for successors $w_1, w_2 \in W_v$ such that $\rho_{w_1} < \rho_{w_2}$. We do so by considering what happens on the breakpoint in between those two points. Because breakpoints may overlap, we instead consider a set of breakpoints with the same value.

Assume there exist a set of nodes $U'_v = \{u'_1, ..., u'_k\} \subseteq U_v$, and nodes $v_1, v_2 \in U_v \cup W_v$ such that $\rho_{v_1} < \rho_{u'_1} = ... = \rho_{u'_k} < \rho_{v_2}$, and there is no other $v' \in (U_v \cup W_v) \setminus U'_v$ with $\rho_{v'} \in (\rho_{v_1}, \rho_{v_2})$. Let $u' \in U'_v$, and consider some $\varepsilon > 0$ such that $\varepsilon < \rho_{v_2} - \rho_{u'}$ and $\varepsilon < \rho_{u'} - \rho_{v_1}$. Then $\text{fr}\,(\rho_{v'} - \rho_{u'} + \varepsilon) = \text{fr}\,(\rho_{v'} - \rho_{u'}) + \varepsilon$, and $\text{fr}\,(\rho_{u'} - \varepsilon - \rho_{v'}) = \text{fr}\,(\rho_{u'} - \rho_{v'}) - \varepsilon$ for all $v' \in (U_v \cup W_v) \setminus U'_v$. Using this fact, the fact that $\hat{y}$ satisfies the flow conservation constraints (2b) and because $\lambda_r = \lambda_t = 0$, we

can show that the following holds:

$$\begin{aligned}
\zeta_v(\rho_{u'} - \varepsilon) &= \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)} \mathrm{fr}\left(\rho_{u'} - \varepsilon - \lambda_u - \hat{\pi}_{(u,v)}\right) + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)} \mathrm{fr}\left(\lambda_w - \rho_{u'} + \varepsilon - \hat{\pi}_{(v,w)}\right), \\
&= \sum_{\substack{(u,v)\in\delta^-(v) \\ u\notin U'_v}} \hat{y}_{(u,v)} \mathrm{fr}\left(\rho_{u'} - \rho_u\right) - \sum_{\substack{(u,v)\in\delta^-(v) \\ u\notin U'_v}} \hat{y}_{(u,v)}\varepsilon + \sum_{\substack{(u,v)\in\delta^-(v) \\ u\in U'_v}} \hat{y}_{(u,v)} \mathrm{fr}\left(\rho_u - \varepsilon - \rho_u\right) \\
&\quad + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)} \mathrm{fr}\left(\rho_w - \rho_{u'}\right) + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)}\varepsilon, \\
&= \zeta_v\rho_{u'}) - \sum_{\substack{(u,v)\in\delta^-(v) \\ u\notin U'_v}} \hat{y}_{(u,v)}\varepsilon + \sum_{\substack{(u,v)\in\delta^-(v) \\ u\in U_v}} \hat{y}_{(u,v)}\left(1-\varepsilon\right) + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)}\varepsilon, \\
&= \zeta_v\rho_{u'}) - \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)}\varepsilon + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)}\varepsilon + \sum_{\substack{(u,v)\in\delta^-(v) \\ u\in U'_v}} \hat{y}_{(u,v)} = \zeta_v\rho_{u'}) + \sum_{\substack{(u,v)\in\delta^-(v) \\ u\in U'_v}} \hat{y}_{(u,v)}.
\end{aligned}$$

Since $\rho_{u'} + \varepsilon - \rho_{u'} = \varepsilon$, we can show using the same derivation and arguments that $\zeta_v(\rho_{u'} + \varepsilon) = \zeta_v(\rho_{u'})$. We conclude that, for $u_1 \in U_v$ and $u_2 \in U_v$, if $\rho_{v'} \notin (\rho_{u_1}, \rho_{u_2})$ for all $v' \in U_v \cup W_v$, then:

$$\zeta_v(\rho_{u_1}) \geq \zeta_v(\rho_{u_2}) \tag{15}$$

Assume there exist a set of nodes $W'_v = \{w'_1, ..., w'_m\} \subseteq W_v$ and nodes $v_1, v_2 \in U_v \cup W_v$ such that $\rho_{v_1} < \rho_{w'_1} = ... = \rho_{w'_m} < \rho_{v_2}$, and there is no other $v' \in (U_v \cup W_v) \setminus W'_v$ with $\rho_{v'} \in (\rho_{v_1}, \rho_{v_2})$. Let $w' \in W'_v$ and consider some $\varepsilon > 0$ such that $\varepsilon < \rho_{v_2} - \rho_{w'}$ and $\varepsilon < \rho_{w'} - \rho_{v_1}$. Then it follows that $\mathrm{fr}\left(\rho_{v'} - \rho_{w'} - \varepsilon\right) = \mathrm{fr}\left(\rho_{v'} - \rho_{w'}\right) - \varepsilon$ and $\mathrm{fr}\left(\rho_{w'} + \varepsilon - \rho_{v'}\right) = \mathrm{fr}\left(\rho_{w'} - \rho_{v'}\right) + \varepsilon$ for all $v' \in (U_v \cup W_v) \setminus W'_v$. This fact and the fact that $\hat{y}$ satisfies the flow conservation constraints (2b) imply the following:

$$\begin{aligned}
\zeta_v(\rho_{w'} + \varepsilon) &= \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)} \mathrm{fr}\left(\rho_{w'} + \varepsilon - \lambda_u - \hat{\pi}_{u,v}\right) + \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)} \mathrm{fr}\left(\lambda_w - \rho_{w'} - \varepsilon - \hat{\pi}_{v,w}\right), \\
&= \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)} \mathrm{fr}\left(\rho_{w'} - \rho_u\right) + \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)}\varepsilon + \sum_{\substack{(v,w)\in\delta^-(v) \\ w\notin W'_v}} \hat{y}_{(v,w)} \mathrm{fr}\left(\rho_w - \rho_{w'}\right) \\
&\quad - \sum_{\substack{(v,w)\in\delta^-(v) \\ w\notin W'_v}} \hat{y}_{(v,w)}\varepsilon + \sum_{\substack{(v,w)\in\delta^-(v) \\ w\in W'_v}} \hat{y}_{(v,w)} \mathrm{fr}\left(\rho_w - \rho_{w'} - \varepsilon\right) \\
&= \zeta_v(\rho_{w'}) + \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)}\varepsilon - \sum_{\substack{(v,w)\in\delta^-(v) \\ w\notin W'_v}} \hat{y}_{(v,w)}\varepsilon + \sum_{\substack{(v,w)\in\delta^-(v) \\ w\in W'_v}} \hat{y}_{(v,w)}\left(1-\varepsilon\right), \\
&= \zeta_v(\rho_{w'}) + \sum_{(u,v)\in\delta^-(v)} \hat{y}_{(u,v)}\varepsilon - \sum_{(v,w)\in\delta^+(v)} \hat{y}_{(v,w)}\varepsilon + \sum_{\substack{(v,w)\in\delta^-(v) \\ w\in W'_v}} \hat{y}_{(v,w)} = \zeta_v(\rho_{w'}) + \sum_{\substack{(v,w)\in\delta^-(v) \\ w\in W'_v}} \hat{y}_{(v,w)},
\end{aligned}$$

and $\zeta_v(\lambda_{u'} - \varepsilon) = \zeta_v(\lambda_{u'})$ follows from the same arguments and derivation. Then, for $w_1 \in W_v$ and $w_2 \in W_v$, if $\rho_{v'} \notin (\rho_{w_1}, \rho_{w_2})$ for all $v' \in U_v \cup W_v$, then:

$$\zeta_v(\rho_{w_1}) \leq \zeta_v(\rho_{w_2}) \tag{16}$$

As a consequence of Inequality (15), Inequality (16) and the fact that $\zeta_v(\lambda_v)$ is a piecewise constant function of $\lambda_v \in [0, 1)$ by Proposition 4.1, we conclude that all $\rho_v \in [\rho_{u_*}, \rho_{w_*}]$ minimize ζ for some $u_* \in U_v$ and some $w_* \in W_v$ such that no nodes lie between u_* and w_*, i.e. for all $v' \in U_v \cup W_v$, $\rho_{v'} \notin (\rho_{u_*}, \rho_{w_*})$. Then $\zeta_v(\rho_{u_*}) = \zeta_v(\lambda_{u_*} + \pi_{(u_*,v)})$ is a minimum of c. □

Then the following corollary also follows from the last paragraph of the proof:

Corollary 4.3.1. *Consider a node $v \in N$ and let $W_v \subseteq N$ be the set of direct successors of v with fixed labels $\lambda_w \in [0,1)$ for $w \in W_v$. Then $\lambda_v^* := fr\left(\lambda_{w_*} - \hat{\pi}_{(v,w_*)}\right)$ for some $w_* \in W_v$ minimizes $\zeta_v(\lambda_v)$.*

Finally, Theorem 4.3 can be proven using the results from the previous lemma.

Theorem 4.3. *Consider a decision diagram $D = (N, A)$. Let $\hat{y}$ be a fractional optimal solution to the LP relaxation (F^{rel}), and let $\hat{\pi} \in [0,1)^{A\times 1}$ be some vector with $\hat{\pi}_a = 0$ for all $a \in A_0$. Then there exists a solution (λ^*, μ^*) that minimizes Formulation (12) such that, for all $v \in N \setminus \{r, t\}$, $\lambda_v^* = fr\left(\lambda_{u_*}^* + \hat{\pi}_{(u_*,v)}\right)$ for some direct predecessor u_* of v.*

Proof. Let U_v denote the set of direct predecessors of some node v, and consider any optimal solution (λ', μ') to Formulation (12).

Consider any node $v' \in N$ that does not have a predecessor label. By Lemma 4.3, there exists some $u_* \in U_{v'}$ such that $\zeta_{v'}(\lambda_{v'})$ is minimized for $\lambda_{v'} = \text{fr}\left(\lambda'_{u_*} + \hat{\pi}_{(u_*,v')}\right)$. We construct a new solution (λ^*, μ^*) to Formulation (12) by setting $\lambda_{v'}^* = \text{fr}\left(\lambda'_{u_*} + \hat{\pi}_{(u_*,v')}\right)$, $\lambda_u^* = \lambda'_u$ for all other nodes $u \in N \setminus \{v'\}$ and setting $\mu^*_{(u,v)} := \text{fr}\left(\lambda_v^* - \lambda_u^* - \hat{\pi}_{(u,v)}\right)$ for all $(u,v) \in A$.

By construction, (λ^*, μ^*) satisfies all constraints of Formulation (12) and is therefore feasible. Furthermore, by Lemma 4.3, $\zeta_{v'}(\lambda^*_{v'}) \leq \zeta_{v'}(\lambda'_{v'})$. Then, by considering our choice for $\mu^*_{(u,v)}$, it follows that the objective function of Formulation (12) is equal to the function $g(\lambda^*)$ defined in Equation (13). We can show the following about $g(\lambda^*)$:

$$
\begin{aligned}
g(\lambda^*) &= \sum_{(u,v)\in A} \hat{y}_{(u,v)} \text{fr}\left(\lambda_v^* - \lambda_u^* - \hat{\pi}_{(u,v)}\right) \\
&= \sum_{\substack{(u,v)\in A \\ u,v\neq v'}} \text{fr}\left(\lambda_v^* - \lambda_u^* - \hat{\pi}_{(u,v)}\right) + \sum_{(u,v')\in A} \hat{y}_{(u,v')} \text{fr}\left(\lambda^*_{v'} - \lambda_u^* - \hat{\pi}_{(u,v')}\right) \\
&\qquad + \sum_{(v',w)\in A} \hat{y}_{(v',w)} \text{fr}\left(\lambda_w^* - \lambda^*_{v'} - \hat{\pi}_{(v',w)}\right) \\
&= \sum_{\substack{(u,v)\in A \\ u,v\neq v'}} \text{fr}\left(\lambda_v^* - \lambda_u^* - \hat{\pi}_{(u,v)}\right) + \zeta_{v'}\left(\lambda^*_{v'}\right) \\
&\leq \sum_{\substack{(u,v)\in A \\ u,v\neq v'}} \text{fr}\left(\lambda'_v - \lambda'_u - \hat{\pi}_{(u,v)}\right) + \zeta_{v'}\left(\lambda'_{v'}\right) = g(\lambda')
\end{aligned}
$$

Since (λ', μ') was defined to be optimal, then $g(\lambda^*) = g(\lambda')$ must follow. This means that (λ^*, μ^*) is an optimal solution to Formulation (12) as well.

Let $\mathcal{N}_j := \{v \in N : L(v) = j\}$ be all nodes of N that are located in layer j, with $\mathcal{N}_1 = \{s\}$ and $\mathcal{N}_{L+1} = \{t\}$. Let $(\lambda^{(1)}, \mu^{(1)})$ be an optimal solution to Formulation (12). We construct a sequence of solutions $(\lambda^{(j)}, \mu^{(j)}), j \in \{2, ..., L\}$ by setting

1. $\lambda_v^{(j)} := \lambda_v^{(j-1)}$ for all $v \in N \setminus \mathcal{N}_j$,
2. $\lambda_v^{(j)} := \text{fr}\left(\lambda'_{u_*} + \hat{\pi}_{(u_*,v)}\right)$ for all $v \in \mathcal{N}_j$ and $u_* \in U_v$ such that $\zeta_v(\lambda_v)$ is minimized for $\lambda_v = \lambda'_{u_*} + \hat{\pi}_{(u_*,v)}$ (which exists by Lemma 4.3), and
3. $\mu^{(j)}_{(u,v)} := \text{fr}\left(\lambda_v^{(j)} - \lambda_u^{(j)} - \hat{\pi}_{(u,v}\right)$ for all $(u,v) \in A$.

By our previous observations, if $(\lambda^{(j-1)}, \mu^{(j-1)})$ is feasible and optimal for Formulation (12), then so is $(\lambda^{(j)}, \mu^{(j)})$. Furthermore, for the solution $(\lambda^{(j)}, \mu^{(j)})$, all $v \in \bigcup_{k=1}^{j} \mathcal{N}_j$ satisfy that $\lambda_v = \text{fr}(\lambda_u + \hat{\pi}_{(u,v)})$ for some $u \in U_v$. It follows that $(\lambda^{(|\mathcal{L}|)}, \mu^{(|\mathcal{L}|)})$ is an optimal solution to Formulation (12) with $\lambda_v = \text{fr}(\lambda_u + \hat{pi}_{(u,v)})$ for some $u \in U_v$ for all $v \in N \setminus \{r, t\}$ □

Then, from Corollary 4.3.1 follows a similar result for direct successors:

Corollary 4.3.1. *Consider a decision diagram $D = (N, A)$. Let $\hat{y}$ be a fractional optimal solution to the LP relaxation (F^{rel}), and let $\hat{\pi} \in [0,1)^{A \times 1}$ be some vector with $\hat{\pi}_a = 0$ for all $a \in A_0$. Then there exists a solution (λ^*, μ^*) that minimizes Formulation (12) such that, for all $v \in N \setminus \{r, t\}$, $\lambda^*_v = fr\left(\lambda^*_{w_*} - \hat{\pi}_{(v,w_*)}\right)$ for some direct successor w_* of v.*

Proof. The proof is analogous to the proof of Theorem 4.3 by starting with $(\lambda^{(|\mathcal{L}|)}, \mu^{(|\mathcal{L}|)})$, working towards $(\lambda^{(1)}, \mu^{(1)})$ and using Corollary 4.3.1. □

4.3 An efficient algorithm for reusing CG-cuts

So far, we showed a method for finding maximum violation CG-cuts in Formulation (6). Subsequently, we set up a method for reusing parts of a CG-cut in future iterations by storing the π-values and completing the CG-cut with Formulation (12). Afterwards, we discussed some results about solving this formulation. The goal of this approach was to find a method to efficiently and inexpensively reuse CG-cuts. As shown in Section 4.2.1, a consequence of Theorem 4.3 is that solving Formulation (12) reduces to finding an r-arborescence on the decision diagram that has a minimum cost with respect to $g(\lambda)$. We showed that such an r-arborescence—and thereby a solution to Formulation (12)—can be found in time $\mathcal{O}(|N|2^{|N|})$. However, such an exponential time algorithm makes computing reused CG-cuts computationally expensive as well.

For arborescences with fixed costs on each arc, a minimum cost arborescence can be found in polynomial time through the matroid intersection of both the partition matroid and the graphical matroid [20]. However, for Problem 4.1, the costs of including some arc in the r-arborescence depends on which other arcs are included in that r-arborescence and are therefore variable. Because of this variability, the matroid intersection algorithm cannot be applied to Problem 4.1, and it might not be polynomial time solveable as a result. At the same time, we have yet to find a reduction from some NP-hard problem, so the complexity of Problem 4.1—and thereby Formulation (12)—is still open.

Without any clear results on complexity, we instead developed a greedy labelling algorithm, which is depicted in Algorithm 4. Given some decision diagram $D = (N, A)$, a fractional solution $\hat{y}$ to the LP relaxation (F^{rel}) and some $\hat{\pi} \in [0,1)^A$ with $\hat{\pi}_a = 0$ for all $a \in A_0$, Algorithm 4 assigns labels λ_v to each node layer by layer from r to t. What label values are assigned to each node depends on some algorithm called `decision_rule(v, λ, π̂, ŷ)` which, for all $v \in N \setminus \{r, t\}$ returns a value in $[0, 1)$ given a current vector $\lambda \in [0,1)^N$, given $\hat{\pi}$ and given $\hat{y}$. By Theorem 4.3, the `decision_rule` algorithm should assign a value λ_v based on the value of some predecessor u_* of v to create a predecessor labelling.

After having completed the node labels λ_v, the algorithm finds the corresponding μ-values (line 8) and α-values (line 9). To find the corresponding γ, the algorithm firstly reconstructs the original π-values (for which we refer to Equation (11)) by setting $\pi_{L(u)} \leftarrow \hat{\pi}_{(u,v)}$ for all arcs $(u, v) \in A$ that are 1-arcs. These π-values are then used to find a γ that completes the CG-cut.

If `decision_rule(v, λ, π̂, ŷ)` is efficient, then Algorithm 4 is certainly efficient as well, as evidenced by the following theorem:

Theorem 4.4. *Suppose finding the values $\lambda_v \in [0,1)$ for all $v \in N \setminus \{r, t\}$ using* `decision_rule(v, λ, π̂, ŷ)` *can be done in time $\mathcal{O}(f(|N|))$. Then Algorithm 4 finds a CG-cut $\alpha^\intercal y \geq \gamma$ in time $\mathcal{O}(|N| + f(|N|))$.*

Proof. We firstly show that an inequality $\alpha^\intercal y \geq \gamma$ returned by Algorithm 4 is actually a CG-cut by showing that the vectors λ, μ and α after termination of Algorithm 4 are feasible for Formulation (6).

Because of how `decision_rule(v, λ, π̂, ŷ)` is defined, $\lambda_v \in [0,1)$ for all $v \in N \setminus \{r, t\}$. Additionally, since λ_r and λ_t are not updated, $\lambda_r = \lambda_t = 0$. Then Constraints (6d) and (6h) are satisfied. Furthermore, $\mu_a \in [0,1)$ for all $a \in A$ because of the fr() operator, satisfying Constraints (6i). As for $\alpha \leftarrow \hat{\pi}_{(u,v)} + \lambda_u - \lambda_v + \mu_{(u,v)}$, Constraints (6a) are satisfied because $\hat{\pi}_{(a)} = 0$ for all $a \in A_0$ and Constraints (6a) are satisfied as well since $\hat{\pi}_{(u,v)} = \pi_j$ for $(u, v) \in A_1$ such that $L(u) = j$.

Algorithm 4: Greedy labelling

Input: a decision diagram $D = (N, A)$ with $n+1$ layers, a vector $\hat{\pi} \in [0,1)^A$ with $\hat{\pi}_a = 0$ for all $a \in A_0$, and a fractional optimal solution $\hat{y}$ to the LP relaxation (F^{rel}).

Output: a CG-cut $\alpha^\intercal y \geq \gamma$

1 **Initialization:** set $\lambda_v \leftarrow 0$ for all $v \in N$, $\mu_a \leftarrow 0$ for all $a \in A$, $\pi_j \leftarrow 0$ for $j = 1, ..., n$, $\alpha_a \leftarrow 0$ for all $a \in A$ and $\delta \leftarrow 0$;
2 **for** $j = 2$ **to** n **do**
3 **foreach** $v \in N_j$ **do** `// assign λ-value to each v ∈ V using decision rule`
4 $\lambda_v \leftarrow$ `decision_rule(v, λ, π̂, ŷ)`
5 **end**
6 **end**
7 **foreach** $(u, v) \in A$ **do**
8 $\mu_{(u,v)} \leftarrow \text{fr}(\lambda_v - \lambda_u - \hat{\pi}_{(u,v)})$ `// find corresponding μ-values per arc`
9 $\alpha_{(u,v)} \leftarrow \hat{\pi}_{(u,v)} + \lambda_u - \lambda_v + \mu_{(u,v)}$ `// complete left hand side of the CG-cut`
10 **if** $\ell(u, v) = 1$ **then**
11 **if** $\pi_{L(u)} = 0$ **then** `// reconstruct original π-vector`
12 $\pi_{L(u)} \leftarrow \hat{\pi}_{(u,v)}$
13 **end**
14 **end**
15 **end**
16 **for** $j = 1$ **to** n **do**
17 $\gamma \leftarrow \gamma + \pi_j$ `// reconstruct original γ`
18 **end**
19 $\gamma \leftarrow \lceil \gamma \rceil$

Moreover:

$$\begin{aligned}
\alpha_{(u,v)} &= \hat{\pi}_{(u,v)} + \lambda_u - \lambda_v + \mu_{(u,v)}, \\
&= \hat{\pi}_{(u,v)} + \lambda_u - \lambda_v + \text{fr}(\lambda_v - \lambda_u - \hat{\pi}_{(u,v)}), \\
&= \hat{\pi}_{(u,v)} + \lambda_u - \lambda_v + (\lambda_v - \lambda_u - \hat{\pi}_{(u,v)}) - \lfloor \lambda_v - \lambda_u - \hat{\pi}_{(u,v)} \rfloor, \\
&= -\lfloor \lambda_v - \lambda_u - \hat{\pi}_{(u,v)} \rfloor \in \mathbb{Z},
\end{aligned}$$

so α satisfies Constraints (6e) as well.

Setting $\pi_{L(u)} \leftarrow \hat{\pi}_{(u,v)}$ for all $(u, v) \in A_1$ yields a vector π with $\pi \in [0,1)^{\mathcal{L}}$, so Constraints (6g) are satisfied too. Furthermore, by lines 16 to 19, $\gamma = \lceil \sum_{j \in \mathcal{L}} \pi_j \rceil$, so $\sum_{j \in \mathcal{L}} \pi_j \leq \gamma < \sum_{j \in \mathcal{L}} \pi_j + 1$, which satisfies Constraint (6c) by using the same value of ε that was used to generate π in the first place. Lastly, because γ is rounded, $\gamma \in \mathbb{Z}$, which means that Constraint (6f) is satisfied as well. This means that the α, λ, μ, π and γ generated by Algorithm 4 are feasible for Formulation 6, and therefore, $\alpha^\intercal y \geq \gamma$ is a CG-cut.

Because the number of arcs on D is bounded by $2|N| - 2$ and because the number of layers is equal to the number of nodes in the worst case, the initialization step of Algorithm 4 takes time $\mathcal{O}(|N|)$. Finding the values $\lambda_v \in N$ for all nodes $v \in N \setminus \{r, t\}$ takes time $\mathcal{O}(f(|N|))$ by our assumption (line 2 to 6). Updating μ, α and π for each arc takes $\mathcal{O}(|N|)$ because the number of arcs is $\mathcal{O}(|N|)$ and the operation is linear time (lines 7 to 15). Summing all π-values takes $\mathcal{O}(|N|)$ time in the worst case (lines 16 to 18), and rounding is an $\mathcal{O}(1)$ time operation (line 19). Then the complexity of Algorithm 4 is $\mathcal{O}(|N| + f(|N|))$ □

While Theorem 4.4 shows that Algorithm 4 has a clear upper bound on complexity, the CG-cut resulting from Algorithm 4 has no quality guarantee. The performance of the algorithm is highly dependent on how the labels λ are chosen, i.e. based on `decision_rule(v, λ, π̂, ŷ)`. Exploiting the results gathered in this chapter so far, we have developed two decision rules: `min_jumps` and `min_local_cost`. In the next two sections, we discuss these decision rules in detail.

4.3.1 The min_jumps decision rule

We concluded in Observation 4.1 that "jumps"—situations with $\hat{\pi}_{(u,v)} + \lambda_u > \lambda_v$—should be avoided because those jumps incur costs, though a jump is sometimes unavoidable if $\hat{\pi}_{(u,v)} + \lambda_u > 1$. Such jumps can be avoided by setting λ_v to be the highest value of $\text{fr}(\hat{\pi}_{(u,v)} + \lambda_u)$ among all u with $(u, v) \in A$ for some $v \in V$. This decision rule can be improved by realizing that, if $\hat{y}_a = 0$ for some arc $a \in A$, no costs are incurred regardless of the choice of λ_v. Arcs with $\hat{y}_{(u,v)} = 0$ can therefore be ignored for the decision rule. If none of the incoming arcs of v then have a positive cost $\hat{y}$, then a good choice for λ_v is $\lambda_v := 0$ to limit the chance of having $\hat{\pi}_{(u,w)} + \lambda_u$ exceeding 1 in successive layers. This is the logic behind our first decision rule, the `min_jumps` decision rule, which is formally given by:

$$\texttt{min\_jumps(v, } \lambda\texttt{, } \hat{\pi}\texttt{, } \hat{y}\texttt{)} := \max\left\{\{\text{fr}(\lambda_u + \hat{\pi}_{(u,v)}) : (u, v) \in A,\ \hat{y}_{(u,v)} > 0\} \cup \{0\}\right\}$$

Corollary 4.4.1. *Algorithm 4 runs in time $\mathcal{O}(|N|)$ with the `min_jumps` decision rule.*

Proof. Consider lines 2 to 6 from Algorithm 4. Because the `min_jumps` decision rule considers all incoming arcs for each node, all arcs are considered exactly once. Since the number of arcs on some decision diagram $D = (N, A)$ is at most $2|N| - 2$, lines 2 to 6 from Algorithm 4 run in time $\mathcal{O}(f(|N|) = \mathcal{O}(|N|)$. Then Algorithm 4 with the `min_jumps` finds a CG-cut in time $\mathcal{O}(|N|) + \mathcal{O}(|N|) = \mathcal{O}(|N|)$ □

4.3.2 The min_local_cost decision rule

Another logical choice for a decision rule is one that directly aims to minimize $g(\lambda)$. However, since Algorithm 4 considers each node separately from r to t, any node label will have to be determined without the information of all successive layers. Therefore, for each decision, not enough information regarding $g(\lambda)$ is available to decide on a label that will minimize $g(\lambda)$, or even the local cost function $\zeta_v(\lambda_v)$

What can be done, however, is considering only the information that is available so far, i.e. the incoming arcs of v. This is what our second decision rule `min_local_cost` does; it considers all possible node labels among incoming arcs (among which, according to Lemma 4.3, exists a label that minimizes $\zeta_v(\lambda_v)$), and selects the label that minimizes the cost along the incoming arcs, i.e.:

$$\texttt{min\_local\_cost(v, } \lambda\texttt{, } \hat{\pi}\texttt{, } \hat{y}\texttt{)} := \underset{\substack{\lambda_v \in \{\text{fr}(\lambda_u + \hat{\pi}_{(u,v)})\} \\ (u,v) \in A}}{\arg\min} \{\sum_{(u,v)\in A} \hat{y}_{(u,v)} \text{fr}\left(\lambda_v - \lambda_u - \hat{\pi}_{(u,v)}\right)\}$$

Corollary 4.4.2. *Algorithm 4 runs in time $\mathcal{O}(|N|)$ with the `min_local_cost` decision rule.*

Proof. The proof is analogous to the proof of Corollary 4.4.1. □

4.4 Experiments

In the previous section, we have shown that Algorithm 4 finds a CG-cut for some decision diagram $D = (N, A)$ with a fractional optimal solution $\hat{y}$ to the LP relaxation (F^{rel}) given some input vector $\hat{\pi}$. Moreover, we have shown two decision rules such that this algorithm finds CG-cuts in time linear in the number of nodes. However, we have yet to show any results on the performance of Algorithm 4 and the CG-cut reusage scheme as a whole. Specifically, it is unknown 1) if reused CG-cuts have sufficient quality in general, 2) what the quality is of the cuts generated with Algorithm 4, and 3) to what extent the quality of reused CG-cuts changes over successive iterations. To shed light on these matters, we ran a series of experiments. We explicate our approach for collecting data in Section 4.4.1, we describe our method for selecting test instances in Section 4.4.2, and we explain how Algorithm 4 and exact solution methods for Formulations (6) and (12) were implemented in Section 4.4.3. In the two sections afterwards, we show the results of our experiments.

4.4.1 Collected data

The objective of the experiments is to measure the quality of reused CG-cuts found with both Algorithm 4 and Formulation (12). To meet this objective, we generated a sample of 10 initial CG-cuts for several graph coloring instances, and recorded the quality of the reused CG-cuts over 50 successive iterations. As a performance measure for quality, we used the violation, which we measured as $\lceil\beta\rceil - \alpha^\intercal\hat{y}$ for a CG-cut $\alpha^\intercal y \geq \lceil\beta\rceil$ and some fractional optimal solution $\hat{y}$. This violation roughly measures how well a fractional optimal point is separated from the resulting solution set. For a CG-cut to separate a fractional optimal point—and be useful in strengthening the formulation—this violation needs to be positive. We wish to stress that, for two CG-cuts $\alpha_1^\intercal y \geq \lceil\gamma_1\rceil$ and $\alpha_2^\intercal y \geq \lceil\gamma_2\rceil$ that have an identical violation with respect to $\hat{y}$, one cut may remove more from the polyhedron than the other depending on the norms of α_1 and α_2. The violation is therefore not an exact measurement of quality.

To conduct the experiments, we considered the LP relaxation (F^{rel}) in each iteration of the Algorithm 3 and recorded whether the solution was fractional. We call any iteration of Algorithm 3 where the recorded optimal solution to the LP relaxation (F^{rel}) is fractional a *fractional iteration*. We firstly observed that, in the earlier iterations of Algorithm 3, the the LP relaxation (F^{rel}) yields integer optimal solutions. Secondly, for the first few fractional iterations of many instances, the LP relaxation (F^{rel}) was solved before 10 CG-cuts could be generated. We therefore decided to start collecting data only after a set number of fractional iterations τ. We define the first iteration after τ fractional iterations—the iteration where data collection starts—to be the *starting epoch*. The later the starting epoch, the larger the decision diagram and the more variables Formulation (6) and Formulation (12) have. The starting epoch should therefore stay relatively small to remain tractability.

At the starting epoch, a sample of 10 CG-cuts was generated by repeating the following procedure 10 times:

1. find a fractional optimal solution to the LP relaxation (F^{rel}),
2. find a maximum violation CG-cut $\alpha^\intercal y \geq \gamma$ that separates this solution using Formulation (6), and
3. add the CG-cut to the LP relaxation (F^{rel}).

For each of these CG-cuts, a pair consisting of the $\hat{\pi}$-values and the γ-value was recorded.

For the 50 successive fractional iterations of Algorithm 3, we considered the fractional optimal solution $\hat{y}$ to the LP relaxation (F^{rel}), and we generated a maximum violation CG-cut that separates this point using Formulation (6). The violation of this CG-cut serves as a benchmark for the maximum violation that can be achieved for any CG-cut separating this point. Then, for each of the recorded $(\hat{\pi}, \gamma)$-pairs, we generated a CG-cut using Formulation (12), a CG-cut using Algorithm 4 with the `min_jumps` decision rule, and a CG-cut using Algorithm 4 with the `min_local_cost` decision rule. The violation of the CG-cut generated using Formulation (12) then represents a benchmark for the maximum violation that can be achieved for any reused CG-cut separating $\hat{y}$, such as the ones generated with Algorithm 4.

4.4.2 Graph instance selection

The DIMACS benchmark set of the second DIMACS challenge [19] provides with a total of 137 graphs to benchmark the performance of a graph coloring algorithm. The benchmark set consists of a number of large graphs, some graph types that are particularly hard to color, and some graphs based on real life data. From among this benchmark set, we did preliminary testing on several instances, and observed that solving Formulation (12) 500 times (10 CG-cuts for 50 iterations) on all instances with 100 or more nodes is intractable. We therefore only examined instances with 100 or fewer nodes. We also filtered all instances that were solved before reaching 50 iterations of fractional solutions, and we filtered all instances that required a large number of iterations of Algorithm 3 (and thereby a large decision diagram) before a sensible starting epoch—that is, a fractional iteration where the LP relaxation requires more than 10 CG-cuts to be solved to optimality.

Instance	$\|V\|$	$\|E\|$	χ	τ
1-FullIns_4	93	593	5	50
3-insertions_3	56	110	4	50
david	87	812	11	150
myciel4	23	71	5	250
myciel5	47	236	6	100
queen7_7	49	952	7	50

Table 1: Numerical summary of the DIMACS instances selected for experimentation, with number of nodes ($|V|$), number of edges ($|E|$), the chromatic number (χ) and number of fractional iterations to the starting epoch (τ) per instance. Values for τ were determined experimentally, while $|V|$, $|E|$ and χ were taken from [16].

From the remaining instances, we selected the largest instance of each graph type such that solving Formulation (12) 10 times for 50 iterations after the starting epoch was still tractable. These instances were 1) `myciel5` which is a 6-colorable graph obtained using the Mycielski transformation, 2-3) `1-FullIns_4` and `3_insertions_3`, which are generalizations of the Mycielski graphs with inserted nodes to increase graph size but not density, 4) `david`, which is a graph where each node represents a character from Charles Dicken's book David Copperfield and where two nodes are connected if the corresponding characters meet each other in the book, and 5) `queen7_7`, representing a 7×7 chessboard where each node corresponds to a square and two nodes are connected by an edge if the corresponding squares are in the same row, column or diagonal. For each of these instances, we experimentally determined a good starting epoch.

We included an additional instance to investigate the effects of delaying the starting epoch. This instance was `myciel4`, which is small enough such that generating CG-cuts after many fractional iterations is still tractable, while still requiring a large number of fractional iterations before solving. For `myciel4`, a fractional solution requiring 10 CG-cuts can be found already at the 50^{th} fractional iteration, but we started collecting data for this instance after 250 fractional iterations. A summary of the selected instances can be found in Table 1.

4.4.3 Implementation details

Algorithm 3, Formulation (6), Formulation (12) (posed as an MILP) and Algorithm 4 were implemented in Python, and we solved each MILP formulation using the Gurobi software suite [14]. We set $\varepsilon = 0.001$ for Formulation (6) and we limited the domain of λ, μ and π to $[0, 0.9999]$ for both Formulation (6) and Formulation (12). To prevent numerical instability, we rounded each number to 6 decimals. For the variable ordering, we used the Minimum Width heuristic (Algorithm 1).

We noticed that, for several instances of Formulation (6) and Formulation (12), an optimal solution was found quickly but the MILP-solver required a lot of time to prove optimality of that solution. We sampled a number of CG-cuts and found that, for every sample cut, an optimal solution was found before exploring 200000 branch & bound nodes while proving optimality typically required over 1 million nodes. We therefore set a limit of 500000 branch & bound nodes to reduce computation times. As a result, some CG-cuts found with Formulation (6) and Formulation (12) might not be truly optimal in our experiments.

To improve upon the solutions of Algorithm 4, we implemented a local search algorithm that continuously loops over all nodes and replaces node labels with the node label of another predecessor if it improves the objective function. The procedure terminates once a set limit to the number of improvements has been reached or if no improvements can be found anymore. Looping over each node $v \in N$ once and testing all possible labels takes $\mathcal{O}(|A|^2) = \mathcal{O}(|N|^2)$ time. However, the algorithm may need to to loop over the set of nodes N multiple times. Among our entire data set though, the largest number of updates we recorded was 5 nodes, so the number of times the local search algorithm loops over N stays limited.

	1-FullIns_4	3_insertions_3	david	myciel4	myciel5	queen7_7
OPT_reused	0.582	0.754	0.718	0.914	0.558	0.654
min_jumps	0.22	0.656	0	0.48	0.29	0.226
min_local_cost	0.204	0.4	0.718	0.286	0.284	0.548

Table 2: Proportion of cuts with a violation larger than 0 per instance of all reused CG-cuts generated with Formulation (12) (the OPT_reused row), Algorithm 4 with the `min_jumps` decision rule (the min_jumps row) and Algorithm 4 with the `min_local_cost` rule (the min_local_cost row).

4.4.4 Performance of the CG-cut reusage scheme

Figure 8 shows, per instance, the comparisons between the performance of a maximum violation CG-cut generated using Formulation (6) and the performance of reused CG-cuts generated using Formulation (12) with ten different pairs of $(\hat{\pi}, \gamma)$-values. Table 2 shows the proportion of reused CG-cuts with a positive violation. In nearly all instances, the mean violation for the reused CG-cuts is consistently positive. Furthermore, for each instance, the number of generated cuts with positive violation was at least 50%.

The results seems to indicate that the larger the maximum violation of the optimum CG-cut, the larger the maximum violation of a reused CG-cut; for the instances `3_insertions_3`, `david`, `myciel5` and `queen7_7`, the performance of reused CG-cuts with the highest violation was in most iterations identical to the performance of the maximum violation CG-cut generated with Formulation (6). However, in most iterations of the instances `david`, `myciel5` and `queen7_7`, this theoretical optimum hovers around 0.5 and the mean of the reused cuts is barely positive. For the `3_insertions_3` instance and the `myciel4` instance, on the other hand, the CG-cuts generated with Formulation (6) had a much larger violation, and the mean of the reused cuts is larger as well. This idea is further supported by the results in Table 2, which shows that both the `3_insertions_3` instance and the `myciel4` instance had the highest number of CG-cuts with positive violation. The `1-FullIns_4` instance seems to be a counterexample to this idea, as the violations of the CG-cuts generated with Formulation (6) are also comparatively large. It should be noted that only two out of the ten original cuts for this instance had a violation higher than 0.5, so potentially, the quality of a reused CG-cut also depends on the quality of the initial CG-cut.

The quality of the reused CG-cuts did not seem to substantially increase or decrease with the number of iterations, while we expected the quality to degrade over time. We hypothesize that this behavior might occur once the performance of reused CG-cuts is measured over a larger number of iterations.

Lastly, in the earlier iterations of the `myciel4` instance, the highest violation reused CG-cuts seem to outperform the maximum violation CG-cut. Theoretically, this is impossible, but we likely generated these results because the true optimum for the maximum violation CG-cut was not found within the 500000 branch & bound node limit.

4.4.5 Performance of Algorithm 4

Figure 9 shows, for each instance, the mean violations of all reused CG-cuts generated with Algorithm 4 for both the `min_jumps` decision rule and the `min_local_cost` decision rule, compared to the mean violations of the reused CG-cuts generated with Formulation (12). Figure 10 depicts per instance the maximum violations among CG-cuts in each iteration for these three series. Table 2 shows the proportion of reused CG-cuts with a positive violation for Algorithn 4 with either decision rule.

In almost all cases, the mean violation for Algorithm 4 with either decision rule is below 0, indicating that most of the generated CG-cuts do not add any value to the formulation. Furthermore, in nearly all instances, at least one of the decision rules fails to produce more than 150 CG-cuts with a positive violation (Table 2). On the other hand, in nearly every iteration of each instance, there is at least one CG-cut generated with Algorithm 4 that has a similar or even identical performance to the reused CG-cuts found with Formulation (12). In the case of `3_insertions_3` even, in each iteration, Algorithm 4 produced a CG-cut with either decision rule that performed as well

as Formulation (12).

As evidenced by Figure 9 and Table 2 for the `david` instance, the `min_local_cost` decision rule outperformed the min_jumps decision rule, while the converse was true for the `myciel4` instance. Table 2 also shows that the `min_jumps` decision rule was generally better for the `3_insertions_3` instance, while the `min_local_cost` decision rule had a better performance for the `queen7_7` instance. These results indicate that one decision rule does not strictly outperforms the other. Rather, the performance of the decision rules seems to be largely dependent on the instance itself.

The results on the `david` instance with the `min_local_cost` decision rule may seem a bit counter-intuitive given that 70% of the cuts had a positive violation while the mean violation was below 0 in almost every iteration. This happened because two pairs of $(\hat{\pi}, \gamma)$-values consistenly produced CG-cuts with a violation of -3.0 or less, while all CG-cuts with a positive violation had a positive violation of 0.5.

Lastly, we again observe that the quality of the reused CG-cuts does not seem to substantially increase or decrease over the 50 recorded iterations.

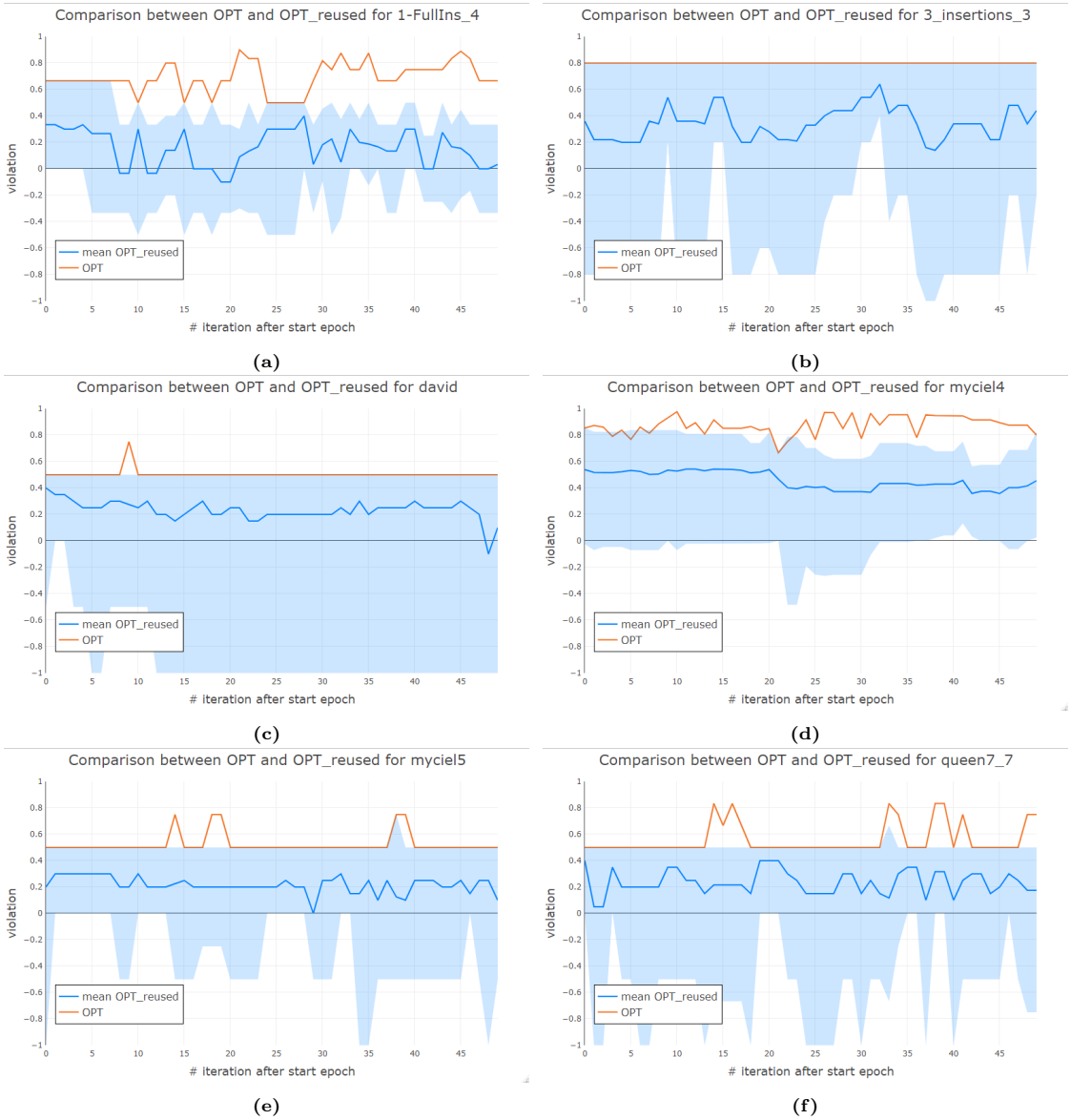


Figure 8: Comparison of the theoretical maximum violation CG-cut in each iteration (OPT series, generated with Formulation (6)) and the mean violation of all reused CG-cuts (mean OPT_reused series, generated with Formulation (12)) over 50 successive fractional iterations for the six selected graph instances. The shaded blue area represents the spread of the violation values of all reused CG-cuts.

Figure 9: Comparison of the mean violations of all reused CG-cuts generated with either Formulation (12) (mean OPT_reused series), or Algorithm 4 with the `min_jumps` decision rule (mean min_jumps series), or Algorithm 4 with the `min_local_cost` decision rule (mean min_local_cost series) over 50 successive fractional iterations for the six selected graph instances.

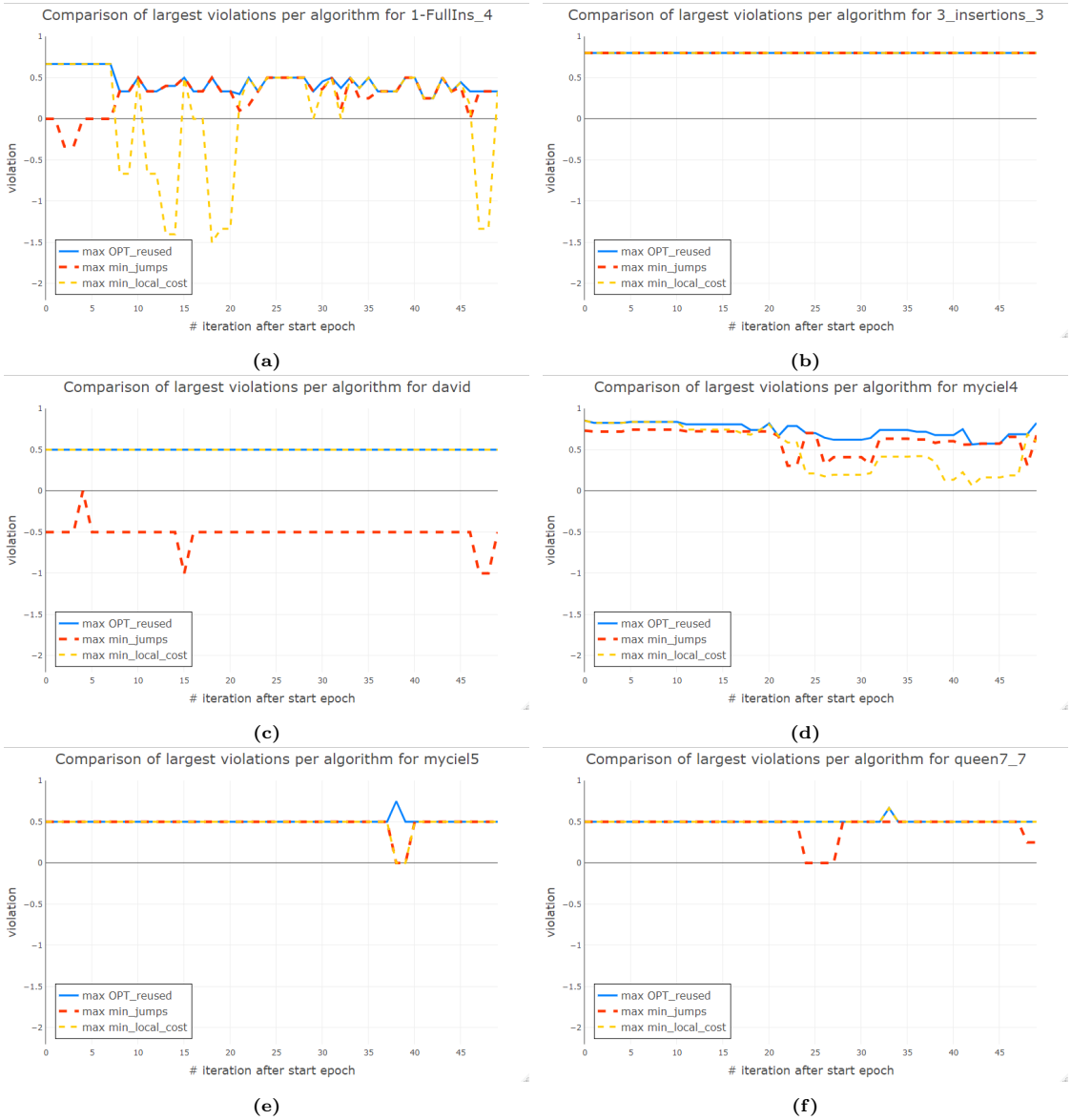


Figure 10: Comparison of the maximum violation among all reused CG-cuts generated with either Formulation (12) (mean OPT_reused series), or Algorithm 4 with the `min_jumps` decision rule (mean min_jumps series), or Algorithm 4 with the `min_local_cost` decision rule (mean min_local_cost series) over 50 successive fractional iterations for the six selected graph instances. Each data point represents the maximum violation present in the data set, so all data points in the same series do not necessarily come from the same pair of $(\hat{\pi}, \gamma)$ pair

4.5 Discussion

From the results in Figure 8 and Table 2, we conclude that the CG-cut reusage scheme as a whole definitely has potential to produce useful CG-cuts. This holds especially true for the `myciel4` instance, for which 90% of the reused CG-cuts had a positive violation. Generally, the results also indicate that the larger the violation of the optimum, the larger the mean violation of a reused CG-cut. We have two potential explanations for this: 1) for instances where CG-cuts with large violations can be found, it is also easier to find reused CG-cuts with large violations, and 2) the performance of a reused CG-cut depends on the pair of $(\hat{\pi}, \gamma)$-values that were used to generate the cut.

Explanation 1) seems elementary; the violation of a maximum violation CG-cut is the largest that can be achieved. Naturally, if that number increases, then so does the upper limit of the violation for reused CG-cuts. Explanation 2) seems to be supported by the results on the `1-FullIns_4` instance; most of the CG-cuts used for producing the pairs of $(\hat{\pi}, \gamma)$-values had a violation of 0.5. In the later iterations of that instance, most maximum violation CG-cuts had a violation of 0.7 or higher while the violations of the reused CG-cuts did not exceed 0.5. This seems to indicate that the quality of a reused CG-cut is dependent on the pairs of $(\hat{\pi}, \gamma)$-values used to generate the cut.

Both of the explanations can be true at the same time, but we think that more evidence is required to give a definitive answer. We currently only gathered data for a number of simpler instances of the DIMACS benchmark set, and we started collecting data after 50 to 250 fractional iterations, while all instances in the sample required many more fractional iterations before finding an optimal solution. We think that more evidence for either explanation can be gathered by investigating more complex instances, either through selecting harder graph instances or by postponing the starting epoch. In general, we hypothesize that the CG-cut reusage scheme also just works better after many fractional iterations, given that it performed best for the instance with the latest starting epoch (`myciel4`).

The general quality of the reused CG-cuts produced with Algorithm 4 was poor, as the mean violation of all cuts generated with Algorithm 4 was consistently lower than 0 and the number of CG-cuts with positive violation was less than 30% in many cases. However, in most instances, Algorithm 4 found a reused CG-cut that had the same violation as the best CG-cut found with Formulation (12). Since a fractional point needs just one CG-cut for separation, only the CG-cut with the highest violation matters. As a result, Algorithm 4 was for most of these instances competitive with Formulation (12).

We also observed that neither decision rule strictly dominates the other, and that the performance of the decision rule seems to depend on the instance. The difference in performance when looking at one instance in isolation can be large, as evidenced by our results on the `david` instance, so picking the right decision rule for an instance is important.

Overall, we did not observe that the quality of the reused CG-cuts demonstrably changes over the iterations that we measured, while we did expect to see a decrease in quality as the number of iterations increases. We currently hypothesize that a larger experiment horizon is required for demonstrating such effects, though it could very well be the case that the quality of reused CG-cuts remains stable over time.

Additionally, the clear computational bottlenecks for gathering the data were Formulations (6) and (12). In many iterations, solving either of these formulations took more time than solving flow model (F) itself, which defeats the purpose of generating such CG-cuts. With this note on tractability in mind, we conclude that Algorithm 4 is the preferred choice over Formulation (12) for producing reused CG-cuts. While the average CG-cuts produced with Algorithm 4 were overall much worse, it produced in nearly every instance at least one CG-cut that had the same violation as the best reused CG-cut found with Formulation (12). These CG-cuts are the only ones that truly matter because the fractional point $\hat{y}$ needs to be cut off only once. Furthermore, Algorithm 4 finds a CG-cut in time $\mathcal{O}(|N|)$ with either decision rule (and $\mathcal{O}(|N|^2)$ when applying the local search), so it is inexpensive to generate a set of CG-cuts and check if any of the generated cuts add value, and one can even test both decision rules to select the better cut. We thus think that the tractability of Algorithm 4 vastly outweighs the slight reduction in performance compared to Formulation (6).

Lastly, there were some limitations with our approach. First of all, the quality of any CG-cut is only roughly measured by the violation. For larger decision diagrams, the vector α is larger and is therefore more likely to have a larger norm, so the results on instances with wildly different sizes of decision diagrams are less comparable. Secondly, we only tested our implementation on smaller DIMACS-instances, and we do not know if our results are generalizable to larger instances. Thirdly, we do not know how our results relate to practice. Four out of our six tested instances—all the Mycielskians—are purely theoretical graphs that are notoriously hard to color, though the `david` and `queen7_7` are more reminiscent of practical problems.

5 Conclusion

In this thesis, we studied the ILP formulation of the constrained network flow problem (flow model (F)) that is solved in the decision diagram based graph coloring algorithm by van Hoeve [15], [16] (Algorithm 3). We firstly studied facet-defining inequalities for integer hull of the ILP formulation, and found that nearly all facet-defining inequalities represented or were implied by r-t cuts. For this specific problem, each r-t cut corresponds to an objective cut. To find objective bound values for the objective cuts, we showed that the objective function of the sequence of ILPs is monotonically increasing in the number of iterations (Theorem 3.1). We furthermore showed that the increase in objective function between iterations cannot be larger than 1 (Theorem 3.2).

Secondly, we investigated the possibility of generating CG-cuts in some iteration of van Hoeve's graph coloring algorithm (Formulation (6)) using multiplier vectors that assign a value in $[0, 1)$ to each arc, each node and each layer. We subsequently developed Formulation (12), which takes the layer-values of a multiplier vector as input and completes the multiplier vector to yield a valid CG-cut. We introduced a minimum cost r-arborescence problem, Problem 4.1, and showed that an optimal solution to this problem translates to a multiplier vector that optimizes Formulation (12).

Furthermore, we developed Algorithm 4, a linear time greedy algorithm that finds (not necessarily optimal) solutions to Formulation (12) with either of two decision rules. We conducted a series of experiments to test the performance of Algorithm 4 and the CG-cut reusage scheme in general. Overall, our results indicate that the CG-cut reusage scheme seems to produce useful CG-cuts, and the best cuts found with Algorithm 4 are competitive with the best possible reused CG-cuts. For some instances, one decision rule was clearly better than the other, but overall neither decision rule seemed to strictly outperform the other.

5.1 Recommendations for future research

Firstly, we discussed the theoretical possibility of using objective cuts to aid the solution of flow model (F) in Chapter 3, but the practical merit of these objective cuts has not yet been investigated. We do hypothesize that the growth of the objective function generally resembles logarithmic growth, which implies that the value of objective cuts increases with the number of iterations. We recommend investigating the effects of objective cuts on solution times.

Van Hoeve did implement a slightly adapted version of his graph coloring algorithm (Algorithm 3) that separates all conflicts it finds in the path decomposition [16]. When multiple conflicts are separated, the objective function increases faster with the number of iterations, which theoretically diminishes the value of objective cuts. As such, the compatibility of objective cuts with multiple conflict resolution should be researched too when measuring the effects of objective cuts on solution times.

Regarding the CG-cut reusage scheme, we also have a few recommendations. Firstly, similar to the objective cuts, it is unknown to what extent the cuts generated with Formulation (12) and Algorithm 4 aid in solving flow model (F). We recommend studying the effects of these types of cuts on solutions times.

Secondly, because we observed that Formulation (6) and (12) were hard to solve, we suggest using cheaper methods of finding CG-cuts. In particular, it could be interesting to test the quality of CG-cuts generated using random $\hat{\pi}$-values. Of course, the performance of such methods should also be tested.

Thirdly, we experimentally investigated the quality of reused CG-cuts generated with Formulation (12) and Algorithm 4, but we only experimented on theoretical graph instances that are relatively small. We hypothesize that the quality of the reused CG-cuts increases once the gap between the relaxed optimum and the integer optimum of flow model (F) becomes larger. We recommend verifying this hypothesis on larger instances, more difficult instances and instances that have practical applicability to see if this technique is promising for solving those instances.

Lastly, the complexity of Problem 4.1 is still an open question. If an optimal solution to this problem can be found in polynomial time, then Formulation (12) can be optimized in polynomial

time, and it will become more attractive to complete multiplier vectors using this formulation. We therefore recommend researching the complexity of Problem 4.1. Given the variable arc costs in this problem, we expect that it is not polynomial time solvable, so we advise starting with reductions from known NP-complete problems.

References

[1] K. I. Aardal, S. P. van Hoesel, A. M. Koster, C. Mannino, and A. Sassano, "Models and solution techniques for frequency assignment problems," *Annals of Operations Research*, vol. 153, no. 1, pp. 79–129, May 2007. DOI: 10.1007/s10479-007-0178-0.

[2] K. Altınel, N. Aras, Z. Şuvak, and Z. C. Taşkın, "Minimum cost noncrossing flow problem on layered networks," *Discrete Applied Mathematics*, vol. 261, pp. 2–21, 2019, ISSN: 0166-218X. DOI: 10.1016/j.dam.2018.09.016. [Online]. Available: https://doi.org/10.1016/j.dam.2018.09.016.

[3] M. Behle, "Binary Decision Diagrams and Integer Programming," *Dissertation*, 2007.

[4] D. Bergman, A. A. Cire, W. J. Van Hoeve, and J. N. Hooker, "Variable ordering for the application of BDDs to the maximum independent set problem," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7298 LNCS, no. March 2014, pp. 34–49, 2012, ISSN: 03029743. DOI: 10.1007/978-3-642-29828-8_3.

[5] K. Bestuzheva, M. Besançon, W.-K. Chen, *et al.*, "The SCIP Optimization Suite 8.0," Optimization Online, Tech. Rep., Dec. 2021. [Online]. Available: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[6] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 993–1002, 1996. DOI: 10.1109/12.537122.

[7] Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986. DOI: 10.1109/TC.1986.1676819.

[8] M. W. Carter, "A Survey of Practical Applications of Examination Timetabling Algorithms," *Operations Research*, vol. 34, no. 2, pp. 193–202, 1986. [Online]. Available: http://www.jstor.org/stable/170814.

[9] M. P. Castro, A. A. Cire, and J. C. Beck, "Decision Diagrams for Discrete Optimization: A Survey of Recent Advances," *INFORMS Journal on Computing*, vol. 34, no. 4, pp. 2271–2295, 2022. DOI: 10.1287/ijoc.2022.1170. [Online]. Available: https://doi.org/10.1287/ijoc.2022.1170.

[10] A. Cire and J. Hooker, "The separation problem for binary decision diagrams," in *Proceedings of ISAIM*, Jan. 2014.

[11] A. A. Cire, A. Diamant, T. Yunes, and A. Carrasco, "A Network-Based Formulation for Scheduling Clinical Rotations," *Production and Operations Management*, vol. 28, no. 5, pp. 1186–1205, 2019. DOI: https://doi.org/10.1111/poms.12978. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/poms.12978.

[12] M. Conforti, G. Cornuéjols, and G. Zambelli, *Integer Programming*. Cham: Springer, 2014.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1979, ISBN: 0716710447.

[14] L. Gurobi Optimization, *Gurobi Optimizer Reference Manual*, 2023. [Online]. Available: https://www.gurobi.com.

[15] W.-J. van Hoeve, "Graph Coloring Lower Bounds from Decision Diagrams," in *Integer Programming and Combinatorial Optimization*, D. Bienstock and G. Zambelli, Eds., Cham: Springer International Publishing, 2020, pp. 405–418, ISBN: 978-3-030-45771-6.

[16] W.-J. van Hoeve, "Graph coloring with decision diagrams," *Mathematical Programming*, vol. 192, no. 1–2, pp. 631–674, May 2021. DOI: 10.1007/s10107-021-01662-x.

[17] A. Jabrayilov and P. Mutzel, "New Integer Linear Programming Models for the Vertex Coloring Problem," in *New Integer Linear Programming Models for the Vertex Coloring Problem*, M. A. Bender, M. Farach-Colton, and M. A. Mosteiro, Eds., Cham: Springer International Publishing, 2018, pp. 640–652, ISBN: 978-3-319-77403-9.

[18] K. Jansen, "Integral Flow with Disjoint Bundles," *Nordic J. of Computing*, vol. 1, no. 2, pp. 264–267, Jun. 1994, ISSN: 1236-6064.

[19] D. S. Johnson and M. A. Trick, "Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13 1993," *American Mathematical Soc.*, vol. 26, 1996.

[20] N. Kamiyama, "Arborescence Problems in Directed Graphs: Theorems and Algorithms," *Interdisciplinary Information Sciences*, vol. 20, no. 1, pp. 51–70, 2014. DOI: 10.4036/iis.2014.51.

[21] A. Karahalios and W.-J. van Hoeve, "Variable ordering for decision diagrams: A portfolio approach," *Constraints*, vol. 27, no. 1–2, pp. 116–133, Jan. 2022. DOI: 10.1007/s10601-021-09325-6.

[22] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Ma*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds., Boston, MA: Springer US, 1972, pp. 85–103, ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9.

[23] B. Korte and J. Vygen, *Combinatorial Optimization Theory and algorithms*. Springer Berlin Heidelberg, 2012.

[24] A. M. C. A. Koster, A. Eisenblätter, and M. Grötschel, "Frequency planning and ramifications of coloring," *Discussiones Mathematicae Graph Theory*, vol. 22, no. 1, pp. 51–88, 2002.

[25] R. Lewis, "A survey of metaheuristic-based techniques for university timetabling problems," *OR Spectrum*, vol. 30, no. 1, pp. 167–190, Jul. 2007. DOI: 10.1007/s00291-007-0097-0.

[26] R. Marappan and G. Sethumadhavan, "Solution to Graph Coloring Using Genetic and Tabu Search Procedures," *Arabian Journal for Science and Engineering*, vol. 43, 2017. DOI: 10.1007/s13369-017-2686-9.

[27] D. Marx, "Graph Coloring Problems and Their Applications in Scheduling," *Periodica Polytechnica, Electrical Engineering*, vol. 48, 2003.

[28] A. Mehrotra and M. A. Trick, "A Column Generation Approach for Graph Coloring," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 344–354, 1996. DOI: 10.1287/ijoc.8.4.344. [Online]. Available: https://doi.org/10.1287/ijoc.8.4.344.

[29] I. Méndez-Díaz and P. Zabala, "A cutting plane algorithm for graph coloring," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 159–179, 2008, ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2006.07.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X0700100X.

[30] R. R. Meyer, "On the existence of optimal solutions to integer and mixed-integer programming problems," *Mathematical Programming*, vol. 7, no. 1, pp. 223–235, Dec. 1974. DOI: 10.1007/bf01585518.

[31] D. R. Morrison, E. C. Sewell, and S. H. Jacobson, "Solving the Pricing Problem in a Branch-and-Price Algorithm for Graph Coloring Using Zero-Suppressed Binary Decision Diagrams," *INFORMS Journal on Computing*, vol. 28, no. 1, pp. 67–82, 2016. DOI: 10.1287/ijoc.2015.0667. [Online]. Available: https://doi.org/10.1287/ijoc.2015.0667.

[32] T. Mostafaie, F. Modarres Khiyabani, and N. J. Navimipour, "A systematic study on meta-heuristic approaches for solving the graph coloring problem," *Computers & Operations Research*, vol. 120, p. 104 850, 2020, ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2019.104850. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054819302928.

[33] U. Pferschy and J. Schauer, "The Maximum Flow Problem with Disjunctive Constraints," *J. Comb. Optim.*, vol. 26, no. 1, pp. 109–119, Jul. 2013, ISSN: 1382-6905. DOI: 10.1007/s10878-011-9438-7. [Online]. Available: https://doi.org/10.1007/s10878-011-9438-7.

[34] Z. Şuvak, İ. K. Altınel, and N. Aras, "Minimum cost flow problem with conflicts," *Networks*, vol. 78, no. 4, pp. 421–442, 2021. DOI: https://doi.org/10.1002/net.22021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.22021.

[35] C. Tjandraatmadja and W.-J. van Hoeve, "Target Cuts from Relaxed Decision Diagrams," *INFORMS Journal on Computing*, vol. 31, no. 2, pp. 285–301, 2019. DOI: 10.1287/ijoc.2018.0830. [Online]. Available: https://doi.org/10.1287/ijoc.2018.0830.

[36] M. Walter, *No Title*, 2016. [Online]. Available: http://polyhedra-oracles.bitbucket.org/.

[37] M. Walter and V. Kaibel, "Investigating polyhedra by oracles and analyzing simple extensions of polytopes," Ph.D. dissertation, 2015, pp. 17–20.