Optimization Methods based on Decision Diagrams for Constraint Programming, AI Planning, and Mathematical Programming

by

Margarita Paz Castro

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Optimization Methods based on Decision Diagrams for
Constraint Programming, AI Planning, and Mathematical Programming

Margarita Paz Castro
Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2020

Decision diagrams (DDs) are graphical structures that can be used to solve discrete optimization problems by representing the set of feasible solutions as paths in a graph. This graphical encoding of the feasibility set can represent complex combinatorial structures and is the foundation of several novel optimization techniques. Due to their flexibility, DDs have become an attractive optimization tool for researchers in different fields, including operations research and computer science.

This dissertation investigates new techniques to use DDs in conjunction with existing discrete optimization approaches based on constraint programming (CP), artificial intelligence (AI), and integer programming (IP). The central thesis of this dissertation is that DDs are effective tools to capture complex combinatorial structures of discrete optimization problems that are not fully exploited by general-purpose solvers. Thus, combinations of DDs with existing technologies can achieve state-of-the-art performance on challenging optimization problems.

Throughout this work, we address this thesis by developing novel DD procedures that leverage methodologies from different optimization fields to solve discrete optimization problems. Our first project employs Lagrangian duality to strengthen DD bounds for pickup-and-delivery problems. The second project explores new ways to generate admissible heuristics for AI planning tasks by combining DD relaxations with AI planning techniques. This work also studies the relationship between DD heuristics and existing admissible heuristics in the community. Lastly, we propose a novel combinatorial lifting procedure and two cutting plane approaches based on DDs for general-form binary optimization problems. We show theoretical guarantees for our lifting procedure (e.g., conditions to obtain facet-defining inequalities) and provide a thorough theoretical analysis of our two cutting plane procedures.

We apply our DD techniques to different problems, extending the usability of DDs in the field. Our first work extends the literature of DDs for sequencing problems by considering capacity constraints and proposing a DD construction procedure based on this restriction. We also present two DD encodings for delete-free AI planning and analyze the properties of both representations. Our last project introduces a new DD network flow formulation and proposes a novel DD encoding for second-order cone inequalities.

Acknowledgements

First and foremost, I want to thank my supervisors, Chris Beck and Andre Cire. I wouldn't be the researcher that I am now without all your guidance and support. I have grown so much in these last five years because of your advice on research and life. You two always made time for me and I will be eternally grateful for that. In particular, I want to thank Chris for pushing me to do better, editing all my documents, and appreciate my sense of humor. I want to thank Andre for teaching me to be strong and fight until the end, having my back, and lifting me up when I was feeling down.

I want to thank the members of my Ph.D. committee for their advice during my studies. I'm grateful to my internal committee members, Merve Bodur and Scott Sanner, for your advice and suggestions over the years. Special thanks to Merve for her kindness, friendship, and support since the first day we meet. I'm happy to have such a strong female role model in my life. I'm also thankful to my external committee members, Cole Smith and Tim Chan, for your comments on my dissertations and interesting discussion during the defense.

Thank you to all the TIDEL members, you were like a family to me. I'm grateful to the TIDEL members that were in the lab when I started, Tony, Chan, Wen-Yang, and Buser, I had so much fun those first two years and I learned so much from you. Thank you to Chiara for being my best friend during this time, you help me more than you imagine. Thank you to Eldan and Kyle, it was great going through the Ph.D. program together and sharing all our ups and downs throughout the years. I also want to thank Tanya, Arik, and Michael, for their friendship during their time at TIDEL. Lastly, I want to thank the newer members of the lab, Arnoosh, Victor, Luke, Jason, Anton, Alex, Giovanni, Ryo, Louis, and Jasper. I wish you all the best.

I'm also thankful to the members of the KR group, which welcomed me with open arms and made me an honorary member. Thank you to Sheila McIlraith for her kindness and advice. I also want to thanks Leon, Toryn, Mayaan, and all the other KR members throughout the years that join us for lunch every day in the Pratt lounge room. Those lunches were one of the most fun parts of my days, reading Toryn's comics and complaining about research.

I would also like to thank my family who made me the person that I am now. To my parents for giving me the love and supports that I needed to get to this point. To my older siblings, Diego, Esteban, and Alejandra, for taking care of me when I was little and being my role models. And to my youngest brother, Vicente, for growing up with me and been one of my first friends. Lastly, to my beautiful nieces and nephew, Olivia, Marina, Cristina, and Emilio, for filling my heart with love, laughter, and games.

Lastly, to the love of my life, Rodrigo, thank you for been there for me in the good and bad times. There are no words to express how much you have helped me throughout the years with your strength, unconditional love, and meticulous analysis of every situation. I'm sure that my journey in the Ph.D. program would not have been as good as it was without you in my life.

Contents

Chapter 1

Introduction

Discrete optimization is a branch of applied mathematics that aims at finding optimal solutions for problems where the decision variables take a finite set of values (e.g., binary or integer values). Many decision-making problems in industry involve discrete decisions such as scheduling resources in hospitals (Cardoen et al., 2010), routing vehicles for transportation companies (Toth and Vigo, 2014), and allocating job requests in cloud computing platforms (Singh and Chana, 2016). Different research communities have proposed general-purpose strategies to solve these problems, including integer programming (IP) in the operations research (OR) community and constraint programming (CP) in computer science (CS).

Despite the many advances in discrete optimization solvers, there are still several challenges involving large and complex problems. Thus, it is necessary to develop new methodologies that can address these challenges and push the boundaries of the field. To do so, we need robust technologies that can exploit structural properties of a wide range of problems. Also, these methodologies should be compatible with existing techniques to complement each other and leverage their strengths, in order to develop new general-purpose solvers that can tackle challenging discrete optimization problems.

Decision diagrams (DDs) are a promising technology to advance the field of discrete optimization. A DD is a graphical structure that represents the set of feasible solutions of a discrete problem. The diagram encodes the feasibility set as paths from its root node to its terminal node. The following example illustrates a DD for an integer linear programming (ILP) problem.

Example 1.1 Consider the ILP model $\mathcal{P}$ with a linear cost function and a feasibility set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : \ 5x_1 + 4x_2 + 3x_3 - x_4 \leq 6\}$. The left-hand side illustration in Figure 1.1 depicts a DD that exactly represents the feasible set $\mathcal{X}$.

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & 2x_1 + x_2 + 3x_3 + x_4 \qquad (\mathcal{P}) \\ \text{s.t.} \quad & 5x_1 + 4x_2 + 3x_3 - x_4 \leq 6, \\ & \boldsymbol{x} \in \{0,1\}^4. \end{aligned}$$

The DD is a layered directed acyclic graph with a root node $\mathbf{r}$ and a terminal node $\mathbf{t}$. Each layer is associated to a variable as shown in the figure. Arcs are labeled with variable assignments, where $x_i = 1$ and $x_i = 0$ assignments are illustrated with solid and dashed arrows, respectively. Each $\mathbf{r} - \mathbf{t}$ path in the DD represents a feasible solution of $\mathcal{X}$, and each solution of $\mathcal{X}$ is encoded as an $\mathbf{r} - \mathbf{t}$ path. DDs fulfilling these properties are known as exact DDs for $\mathcal{X}$.

The bold path $(\mathbf{r}, u_1, u_3, u_6, \mathbf{t})$ corresponds to an optimal solution for $\mathcal{P}$, i.e., point $\boldsymbol{x}^* = (0, 1, 1, 1)$. We can identify this optimal solution using a longest-path procedure where the length of each arc is given by its label (i.e., variable assignment) times the respective coefficient in the linear objective. For example, arc $(\mathbf{r}, u_1)$ has length 0 and arc $(\mathbf{r}, u_2)$ has length 2. □

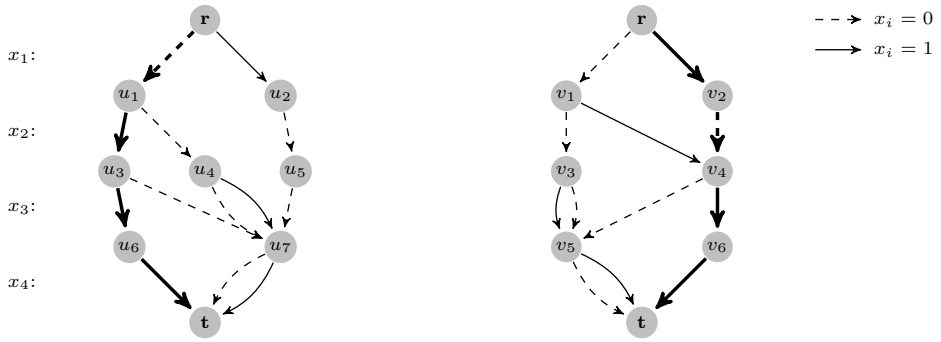


Figure 1.1: An exact DD (left) and relaxed DD (right) for problem $\mathcal{P}$. Bold paths represent optimal solutions over the DDs.

DDs have shown promising results for solving optimization problems when combined with IP and CP methodologies (Bergman et al., 2016a). This graphical representation of the solution set is flexible and can be easily integrated into a wide range for existing techniques. For example, DDs can be combined with IP solvers using a network flow formulation over the graph, or with CP technologies by developing a propagation procedure for the DD. Also, the DD graphical encoding is a suitable alternative to model complex combinatorial structures that are hard to represent with linear inequalities. For example, DDs can encode non-linear objective functions (Bergman and Cire, 2018) and quadratic constraints (Bergman and Lozano, 2020).

One of the main limitations of DDs is their exponential growth, i.e., the size of the graph grows exponentially with respect to the number of variables. Andersen et al. (2007) propose limited size DDs (i.e., relaxed DDs) to overcome this issue. A relaxed DD over-approximates the set of feasible solutions by representing infeasible paths. This relaxed graphical structure provides tight discrete relaxations for several problems and, thus, computes strong dual bounds.

Example 1.2 The right-hand side illustration in Figure 1.1 is a relaxed DD for $\mathcal{P}$. It represents all points in the feasibility set $\mathcal{X}$ as $\mathbf{r} - \mathbf{t}$ paths, but some paths corresponds to infeasible assignments. For example, the bold path $(\mathbf{r}, v_2, v_4, v_6, \mathbf{t})$ corresponds to the infeasible solution $(1, 0, 1, 1)$.

We can compute valid dual bounds for $\mathcal{P}$ using the longest-path algorithm over the relaxed DD described in Example 1.1. The bold path in the relaxed DD depicts the longest path with value 6. This value is a valid upper bound for $\mathcal{P}$ since all points in $\mathcal{X}$ are also encoded inside the relaxed DD. Conversely, the longest path in the exact DD has a cost of 5 and is the optimal value of $\mathcal{P}$. □

Thesis Statement

This dissertation explores new procedures that combine DDs with existing technologies in the OR and CS literature to solve discrete optimization problems. The central thesis of this dissertation is as follows:

> *DDs are effective tools to capture complex combinatorial structures of discrete optimization problems that are not fully exploited by general-purpose solvers. Thus, combinations of DDs*

with existing technologies can achieve state-of-the-art performance on challenging optimization problems.

This dissertation approaches this thesis by presenting novel techniques based on DDs. We explore new procedures that integrate DD-based methodologies into general-purposes solvers and study their properties and limitations. Moreover, we tackle challenging discrete optimization applications that are new to the DD literature. Thus, this dissertation extends the set of DD-based methodologies and the type of problems that can be addressed with them. The following sections provide an overview of the dissertation and present the main contributions of this work.

1.1 Dissertation Overview

This dissertation contains two preliminary chapters (Chapters 2 and 3) that formally present DDs and review the literature on DDs for discrete optimization. The following three chapters (i.e., Chapters 4, 5, and 6) correspond to the main contributions of this dissertation where we tackle different discrete optimization problems and develop novel DD procedures to solve them. Lastly, Chapter 7 includes the overall conclusions and future work directions. A summary of each chapter follows.

Chapter 2 presents the necessary background on DDs for this dissertation. We formally define DDs in the context of discrete optimization and present basic procedures to build and manipulate them. We also introduce relaxed DDs and explain basic algorithms to create tight relaxations for common combinatorial structures. The chapter ends with a general discussion on how to integrate DDs with IP and CP technologies.

Chapter 3 reviews the literature on DDs for discrete optimization problems. This survey focuses on works that employ DDs to model the set of solutions for combinatorial problems. We organize the literature review considering the DD representation of the problem (i.e., exact or approximate) and the techniques employed.

Chapter 4 addresses the multi-commodity pickup-and-delivery traveling salesman problem (i.e., m-PDTSP). The problem considers a vehicle with limited capacity and different size commodities that need to be picked up and delivered to different locations. Thus, the m-PDTSP generalizes several vehicle routing variants such as the traveling salesman problem (TSP), the sequential order problem, and the pickup-and-delivery TSP. We present a novel approach to tackle this problem that considers a DD relaxation enhanced with Lagrangian penalties. This work also introduces new procedures to build DDs that focus on the capacity restriction of the problem, and theoretical guarantees of these novel techniques. The empirical evaluation shows that our technique surpasses state-of-the-art methodologies and closes 33 open instances in the literature. This work was published in the *INFORMS Journal of Computing* (Castro et al., 2020a).

Chapter 5 considers a planning problem from the artificial intelligence (AI) community, i.e., delete-free AI planning. We present two novel DD encodings of the problem to create admissible heuristics (i.e., dual bounds). The first approach is a multivalued decision diagram (MDD) that models the sequential aspect of the problem. The second encoding corresponds to a binary decision diagram (BDD) that ignores the sequential constraints and models the combinatorial structure of the problem. We present compilation procedures that guarantee the admissibility and consistency of the DD-based heuristics and relate our techniques with admissible heuristics in the literature. The chapter includes an extensive empirical evaluation that highlights the strengths and weakness of each DD encoding, and shows the

potential of these approaches when compared to state-of-the-art techniques. This work led to a conference paper at *The International Conference on Automated Planning and Scheduling* (Castro et al., 2019) and a journal publication in the *Journal of Artificial Intelligence Research* (Castro et al., 2020c).

Chapter 6 introduces a novel cut-and-lift procedure based on DDs for binary optimization problems. We present a general framework that can be applied to a large range of binary optimization problems and show its applicability for second-order conic programming. We identify conditions for which our lifted inequalities are facet-defining and derive a new DD-based cut generation linear program. Such a model serves as a basis for a max-flow combinatorial algorithm over the DD that can be applied to derive valid cuts more efficiently. Our numerical results show strong performance when incorporated into a state-of-the-art IP solver, significantly reducing the root node gap, increasing the number of problems solved, and reducing the run-time by a factor of three on average. This work is under review in *Mathematical Programming* (Castro et al., 2020b) and won two student paper competitions in 2020: first place at the Canadian Operation Research Society and runner-up at the INFORMS Computing Society.

Finally, we conclude this dissertation in Chapter 7 with a set of final remarks and future research directions.

1.2 Summary of Contributions

The contributions of this dissertation include new DD encodings for novel applications and new DD procedures to solve discrete optimization problems. We study the theoretical properties of our DD-based techniques and evaluate their empirical performance with respect to state-of-the-art procedures. The following lists include the main contributions of each chapter in terms of models, algorithms, theory, and empirical results.

Chapter 4: Multi-Commodity Pickup-and-Delivery

1. A novel relaxed DD encoding for the m-PDTSP that provides valid dual bounds. We present structural results and strategies for constructing relaxed DDs that take into account both tour constraints and vehicle capacities, extending previous work on DDs for sequencing problems.

2. A Lagrangian technique to strengthen the DD dual bounds. We introduce Lagrange multipliers that penalize DD solutions which do not represent valid Hamiltonian tours or which violate precedence and capacity constraints. Thus, the technique exploits discrete and linear relaxations information, benefiting from DD and IP technologies.

3. A numerical study that evaluates our DD construction strategies and the performance of different DD-based Lagrangian relaxations. We incorporate our DD relaxation into a CP solver and evaluate the quality of our bounds and the solution performance with respect to state-of-the-art techniques. Our methodology provides improvements over the existing dataset, which is more pronounced when the instances have a small vehicle capacity relative to the commodity weights.

Chapter 5: Delete-Free AI Planning

1. An MDD encoding of a delete-free planning task and a BDD representation of its sequential relaxation. We propose construction procedures for each graphical structure that guarantee the

admissibility and consistency of their resulting heuristics. We explore the theoretical properties of our relaxed DDs and relate their heuristics to existing techniques in the literature.

2. A set of novel procedures to extract additional information from relaxed DDs. We present pre-processing procedures to identify landmarks and redundant operators. We also consider solution extraction techniques from relaxed DDs to compute primal bounds.

3. A novel critical-path algorithm to compute dual bounds from relaxed MDDs. The procedure returns bounds that dominate the shortest path alternative. Our critical-path bounds can be applied to other sequential problems that consider multiple dependencies.

4. An extensive empirical analysis that highlights the advantages and disadvantages of using DD-based admissible heuristics instead of linear programming (LP) relaxations. We identify domain characteristics that are suited for our BDD and MDD heuristics. We show that small relaxed DDs have a competitive performance with respect to LP relaxations on most instances.

Chapter 6: Cut Generation and Lifting for Binary Optimization Problems

1. A general combinatorial lifting procedure applicable to any binary problem encoded using one or multiple (relaxed) DDs. We show theoretical properties of the lifted inequality, including sufficient conditions to obtain facet-defining inequalities. The procedure relates to several lifting algorithms based on 0-1 disjunctions and it is the first one to leverage the combinatorial structure of the problem via DDs.

2. A novel cutting-plane algorithm defined over a DD. This procedure extends the literature on DD-based cuts by proposing a new cut generation approach that formulates the separation problem as a joint-capacity max-flow problem. We show that our cuts define the convex hull of the feasibility set and present a theoretical comparison with existing DD-based cutting-plane approaches. We also introduce a tractable but weaker alternative to our cuts and prove that these cuts are stronger when using reduced DDs.

3. A novel DD representation of second-order cone (SOC) inequalities. The DD encoding is based on a recursive model for SOC constraints and extends the set of non-linear constraints encodings based on DDs. Our construction procedure is an extension of the iterative refinement for linear constraints and can construct exact and relaxed DDs.

4. A numerical analysis that evaluates the effectiveness of our combinatorial cut-and-lifting procedure for SOC problems. We tested our approach over the well-known SOC knapsack constraints and general-form SOC inequalities coming from chance-constrained stochastic problems. Our procedure outperforms a state-of-the-art IP solver in both scenarios and achieves better performance than existing cut-and-lifting techniques for SOC knapsack constraints.

Chapter 2

Background on Decision Diagrams

This chapter introduces Decision Diagrams (DDs) for discrete optimization, including relevant procedures and the notation employed in this dissertation. We start by defining DDs for a general optimization problem and use a binary knapsack problem as a running example throughout this chapter.

Consider a maximization problem DO of general form to present the theoretical properties and algorithms of DDs. The problem considers an n-dimension integer variable $\boldsymbol{x} \in \mathbb{Z}^n$ with finite domain, i.e., the domain D_i of x_i is a finite subset of $\mathbb{Z}$ for each $i \in I = \{1, \dots, n\}$. We denote the feasible set of DO by $\mathcal{X} \subseteq D_1 \times \cdots \times D_n$, which could be represented by one or more constraints, and consider a objective function $f : \mathbb{Z}^n \to \mathbb{R}$. The model is given by:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{Z}^n \end{aligned} \tag{DO}$$

DDs are a graphical representation of all the points in $\mathcal{X}$. Specifically, a DD is a layered graph where each layer is associated with a variable in $\boldsymbol{x}$ and arcs emanating from that layer have labels that correspond to possible variable-value assignments. We say that a DD $\mathcal{D}$ exactly represents $\mathcal{X}$ if all paths in $\mathcal{D}$ have a one-to-one relationship to points in $\mathcal{X}$.

Example 2.1 Consider a knapsack problem with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and a linear objective $f(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x}$ with cost vector $\boldsymbol{c} = (4, 2, 5, 1)$. Figure 2.1 illustrates the set of feasible solution in $\mathcal{X}$ over two graphs. Here, dashed arrows represent arcs associated with a zero-value variable assignment (i.e., $v_a = 0$) and solid arrows correspond to arcs with a one-value variable assignments (i.e., $v_a = 1$). The left graph in Figure 2.1 corresponds to a decision tree (e.g., in a branch-and-bound procedure) where paths from the root $\mathbf{r}$ to each leaf node correspond to valid variable assignments. Conversely, the right graph illustrates a DD for $\mathcal{X}$. Each DD layer is associated with a variable and arcs denote valid variable-value assignments. Note that there is a one-to-one correspondence between root-to-leaf paths in the decision tree and $\mathbf{r} - \mathbf{t}$ paths in the DD, i.e., the DD exactly represents $\mathcal{X}$. □

Example 2.1 illustrates a compact DD representation of a discrete optimization problem. As discussed in Section 2.3, this graphical encoding is quite flexible and can be used in conjunction with existing methodologies to efficiently solve optimization tasks. For example, DDs can be seen as global constraint in Constraint Programming (CP) and, thus, can be used for inference and propagation (Hoda et al., 2010). Alternatively, the convex combination of solutions in a DD can be represented as a network-flow

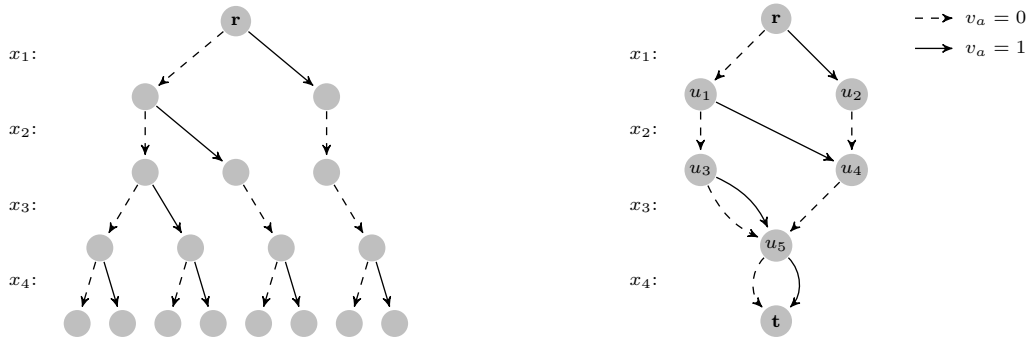


Figure 2.1: A decision tree and a DD for $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$.

model (Behle, 2007), i.e., an extended linear reformulation of $\mathcal{X}$.

We now formally define DDs and introduce the notation used throughout this dissertation. A DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ is a layered-directed acyclic graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$. The node set $\mathcal{N}$ is partitioned into $n+1$ layers $\mathcal{N} = (\mathcal{N}_1, \dots, \mathcal{N}_{n+1})$. The first and last layers are the singletons $\mathcal{N}_1 = \{\mathbf{r}\}$ and $\mathcal{N}_{n+1} = \{\mathbf{t}\}$, respectively, where $\mathbf{r}$ is the root node and $\mathbf{t}$ is the terminal node. An arc $a = (u, u') \in \mathcal{A}$ has a source node $s(a) = u$ and a target node $t(a) = u'$ in consecutive layers, i.e., $u' \in \mathcal{N}_{i+1}$ whenever $u \in \mathcal{N}_i$ for every $i \in I$.

The points in $\mathcal{X}$ are mapped to paths in the network, as follows. Without loss of generality, arcs emanating from layer $\mathcal{N}_i$, for each $i \in I$, are associated with values in the domain of variable x_i. Every arc $a \in \mathcal{A}$ with source $s(a) \in \mathcal{N}_i$ has a value $v_a \in D_i$, and each node $u \in \mathcal{N}_i$ has at most $|D_i|$ emanating arcs, each one with a different value. Given an arc-specified $\mathbf{r} - \mathbf{t}$ path $p = (a_1, \dots, a_n)$ with $s(a_1) = \mathbf{r}$ and $t(a_n) = \mathbf{t}$, we let $\boldsymbol{x}^p = (v_{a_1}, v_{a_2}, \dots, v_{a_n}) \in D_1 \times \cdots \times D_n$ be the n-dimensional point encoded by path p. Then, the set of points represented by the DD is

$$\text{Sol}(\mathcal{D}) = \bigcup_{p \in \mathcal{P}} \{\boldsymbol{x}^p\},$$

where $\mathcal{P}$ is the set of all $\mathbf{r} - \mathbf{t}$ paths in $\mathcal{D}$. Thus, $\mathcal{D}$ exactly represents $\mathcal{X}$ if $\text{Sol}(\mathcal{D}) = \mathcal{X}$, i.e., there is a one-to-one correspondence between points in $\mathcal{X}$ and paths in $\mathcal{P}$.

We can naively construct a DD for $\mathcal{X}$ by enumerating all points in $\mathcal{X}$ in a decision tree and then merging nodes in the same layer with equivalent sub-trees (see Figure 2.1). However, this procedure is impractical due to the potential exponential size of $\mathcal{X}$. The following sections explain how to construct DD in a systematic and tractable manner, while introducing alternatives to overcome their potential exponential size.

Outline. The reminder of this chapter is as follows. Section 2.1 presents how to construct a DD for any recursive model and introduces a reduction procedure to decrease its size. Section 2.2 defines a relaxed DD as a limited-sized DD that over-approximates the feasible set of the problem. This section includes a theoretical framework on the creation of valid DD relaxations and examples for commonly used combinatorial structures. Lastly, Section 2.3 discusses how to use DDs in conjunction with existing technologies to solve discrete optimization problems.

2.1 Decision Diagrams and Recursive Formulations

DDs are graphical structures that encode the feasible set of any discrete optimization problem. However, there are currently no efficient algorithms that can create DDs for an arbitrary problem DO. As depicted in Figure 2.1, a naive alternative is to construct the decision tree defining $\mathcal{X}$ and create a DD by merging nodes with equivalent sub-trees. Unfortunately, this procedure is intractable due to the potentially exponential size of the feasible set $\mathcal{X}$.

A common practice in the literature is to construct DDs based on a recursive formulation of DO. As discussed by Hooker (2013), there is a strong relationship between DDs and the state transition graph in the DP literature. DPs represent optimization problems via a recursive model where decisions are made sequentially (Bertsekas, 2017). Then, the state transition graph can be obtained by unfolding the recursive model for any possible state in the system.

In the following, we detail how to construct a DD for DO based on a recursive reformulation. We start by introducing a general-form recursive model ROP using syntax from the DP literature (Bertsekas, 2017). We consider a $n+1$ stage recursive model with decision variables $\boldsymbol{x} \in \mathbb{Z}^n$, i.e., one decision variable x_i for each stage $i \in I$. For a given stage $i \in \{1, \dots, n+1\}$, state variables $\boldsymbol{S} \in \mathcal{S}_i$ represent the current information of the system given the decisions made so far, where set $\mathcal{S}_i$ is the state space at stage i. The set of feasible assignments of variable x_i at a state $\boldsymbol{S} \in \mathcal{S}_i$ is given by the feasibility set $X_i(\boldsymbol{S}) \subseteq D_i$. To move from one stage to the next, the transition function $\phi_i : \mathcal{S}_i \times X_i \to \mathcal{S}_{i+1}$ maps the current state and decision variable in stage i to a state in the following stage, i.e., $\boldsymbol{S}' = \phi_i(\boldsymbol{S}, x)$ with $\boldsymbol{S}' \in \mathcal{S}_{i+1}$. Each state $\boldsymbol{S} \in \mathcal{S}_i$ and decision variable $x \in X_i(\boldsymbol{S})$ at stage $i \in \{1, \dots, n+1\}$ has an immediate cost $f_i(\boldsymbol{S}, x)$ and the total cost is given by the accumulated costs over all stages. Consider $h_i : \mathcal{S}_i \to \mathbb{R}$ as the maximum accumulated cost from a state in stage i to any last-stage state. Then, the recursive model ROP is given by:

$$h_i(\boldsymbol{S}) = \max_{x \in X_i(\boldsymbol{S})} \{f_i(\boldsymbol{S}, x) + h_{i+1}(\phi_i(\boldsymbol{S}, x))\}, \qquad \forall i \in I, \tag{ROP}$$

where the optimal value of the system is the accumulated value at the initial state, i.e., $h_1(\boldsymbol{S}_1)$. Without loss of generality, we assume that the accumulated cost for all last-stage state variables $\boldsymbol{S} \in \mathcal{S}_{n+1}$ is $h_{n+1}(\boldsymbol{S}) = 0$ and that there is a single initial state, i.e., $\mathcal{S}_1 = \{\boldsymbol{S}_1\}$.

Example 2.2 Consider the knapsack problem introduced in Example 2.1 with weight vector $\boldsymbol{w} = (7, 5, 4, 1)$. We consider decision variables $\boldsymbol{x} \in \{0, 1\}^4$ and a single state variable $\boldsymbol{S} = Q$ that represents the load of the knapsack at each stage. Then, for an initial state $Q_1 = 0$, the recursive model for this knapsack instance is given by

$$h_i(Q) = \max_{x \in X_i(Q)} \{c_i x + h_{i+1}(Q + w_i x)\}, \qquad \forall i \in \{1, \dots, 5\}. \tag{R-KNP}$$

The transition function $\phi_i(Q, x) = Q + w_i x$ updates the load of the knapsack, while the immediate cost function $f_i(Q, x) = c_i x$ corresponds to the gain of choosing item i. Since the weight of each item is positive, the feasibility set for state $Q \in \mathcal{S}_i$ is given by $X_i(Q) = \{x \in \{0, 1\} : Q + w_i x \leq 8\}$, for all stages $i \in \{1, \dots, 4\}$. □

2.1.1 Construction and Optimization

Given a recursive model ROP, we can construct a DD for $\mathcal{X}$ in a similar fashion as its state transition graph, i.e., by unfolding the recursive model one stage at a time in sequential order. The main idea is to associate a unique state $\boldsymbol{S} \in \mathcal{S}_i$ to each DD node in layer $\mathcal{N}_i$ and $i \in I$. Specifically, we denote by $\boldsymbol{S}(u)$ to the state $\boldsymbol{S}$ associated with node $u \in \mathcal{N}$. Starting from $i = 1$, the root node $\mathbf{r}$ corresponds to the initial state $\boldsymbol{S}_1$, i.e., $\boldsymbol{S}(\mathbf{r}) = \boldsymbol{S}_1$. Then, each node in layer $\mathcal{N}_i$ is associated with a state $\boldsymbol{S} \in \mathcal{S}_i$ such that its emanating arcs have values in $X_i(\boldsymbol{S})$ and point to nodes in layer $\mathcal{N}_{i+1}$ with state values given by the transition function $\phi_i(\cdot)$.

Algorithm 1 depicts this procedure known as top-down construction in the DD literature (Bergman et al., 2011). The algorithm receives the recursive model ROP as input. It then unfolds the recursion in sequential order by creating a node for each new observed state. Arcs emanating from a node $u \in \mathcal{N}_i$ are directed to nodes in the next layer following the transition function, i.e., $u' = t(a) \in \mathcal{N}_{i+1}$ for arc $a \in \mathcal{A}^{\text{out}}(u)$ if and only if $\boldsymbol{S}(u') = \phi_i(\boldsymbol{S}(u), v_a)$. Lastly, since the final layer considers a single terminal node $\mathbf{t}$, this node represents the union of all last-stage states.

Algorithm 1 DD Top-Down Construction Procedure

procedure TopDownDD(ROP)
  Create DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with $n + 1$ empty layers.
  Create the root and terminal node, i.e., $\mathbf{r} \in \mathcal{N}_1$ and $\mathbf{t} \in \mathcal{N}_{n+1}$
  Assign the initial state to the root node, i.e., $\boldsymbol{S}(\mathbf{r}) = \boldsymbol{S}_1$
  for $i \in \{1, \dots, n-1\}$ **do**
    for $u \in \mathcal{N}_i$ **do**
      Create an arc a emanating from u for each possible value in $X_i(\boldsymbol{S}(u))$
      for $a \in \mathcal{A}^{\text{out}}(u)$ **do**
        if there exists node $u' \in \mathcal{N}_{i+1}$ with $\boldsymbol{S}(u') = \phi_i(\boldsymbol{S}(u), v_a)$ **then**
          Direct arc a to node u', i.e., $t(a) = u'$
        else
          Create a new node u' in $\mathcal{N}_{i+1}$ with $\boldsymbol{S}(u') = \phi_i(\boldsymbol{S}(u), v_a)$ and point arc a to u'
  %% Last layer %%
  for $u \in \mathcal{N}_n$ **do**
    Create an arc a emanating from u for each possible value in $X_n(\boldsymbol{S}(u))$ with target $t(a) = \mathbf{t}$
  return $\mathcal{D}$

Notice that in the above procedure all nodes in a layer are associated with different states. Moreover, we construct an arc for each possible value of x_i given by its current state and stage $i \in I$. Thus, the procedure creates an exact DD for the feasible set $\mathcal{X}$ defined by ROP.

Example 2.3 Consider the knapsack problem in Example 2.1 with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and recursive model R-KNP. Figure 2.2 depicts the top-down construction procedure, including the resulting DD (right graph). The left graph shows the resulting diagram after creating all nodes in layer $\mathcal{N}_2$ where the state of each node is written below it. Similarly, the middle graph illustrates the diagram after creating all nodes in layer $\mathcal{N}_3$. Notice that all nodes in the same layer have different states and the graph structure is defined by the transition function in R-KNP. □

As in deterministic DP models (Bertsekas, 2017), we can use a longest-path algorithm over $\mathcal{D}$ to obtain the optimal value of ROP. Intuitively, we assign each arc $a \in \mathcal{A}$ a length given by the immediate cost function over its value v_a and the state of its emanating node, i.e., $\ell_a = f_i(\boldsymbol{S}(u), v_a)$ for any arc a emanating from node $u \in \mathcal{N}_i$. Then, the longest $\mathbf{r} - \mathbf{t}$ path corresponds to the optimal solution over $\mathcal{D}$.

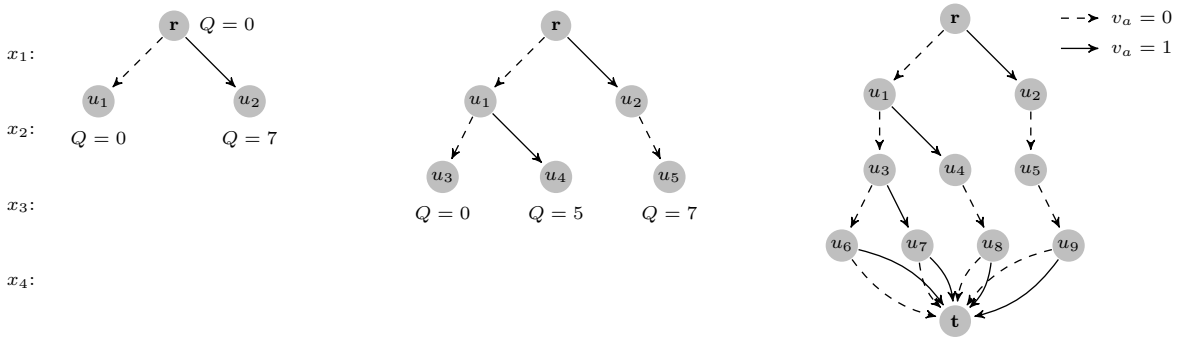


Figure 2.2: Top-down DD construction procedure for R-KNP. Values below nodes correspond to states.

The longest-path algorithm starts at the terminal node $\mathbf{t}$ and traverses each layer in decreasing index order, i.e., we process all nodes in layer $\mathcal{N}_i$ before moving to layer $\mathcal{N}_{i-1}$. Consider $h(u)$ as the maximum cumulative value from node $u \in \mathcal{N}_i$ to $\mathbf{t}$, i.e., the value function over the state associated with u, $h(u) = h_i(\boldsymbol{S}(u))$. We calculate the longest-path value by assigning $h(\mathbf{t}) = 0$ and applying the following recursion:

$$h(u) = \max_{a \in \mathcal{A}^{\text{out}}(u)} \{\ell_a + h(t(a))\}, \qquad \forall u \in \mathcal{N}_i,\ i \in I.$$

The above recursion is equivalent to the recursive model ROP due to our arc length definition and the top-down DD construction procedure. Thus, the longest-path value $h(\mathbf{r})$ corresponds to the optimal value of ROP, i.e., $h_1(\boldsymbol{S}_1)$.

Example 2.4 Consider the knapsack problem in Example 2.1 with cost vector $\boldsymbol{c} = (4, 2, 5, 1)$ and its top-down DD in Figure 2.2 (right). Starting at the terminal node we have $h(\mathbf{t}) = 0$ and value $h(u_9) = h(u_8) = h(u_7) = h(u_6) = 1$ for nodes in layer $\mathcal{N}_4$. The longest path is given by $p = (\mathbf{r}, u_1, u_3, u_7, \mathbf{t})$ with value $h_1(\mathbf{r}) = 6$. □

2.1.2 Reduction Procedure

Algorithm 1 constructs a DD that has a node in layer $\mathcal{N}_i$ for each reachable state at stage $i \in \{1, \ldots, n\}$. However, the number of states in each stage can grow exponentially in n. One way to overcome this issue and decrease the size of a DD $\mathcal{D}$ is to apply a reduction procedure, i.e., merge nodes with different state values that have equivalent sub-graphs. By doing so, the resulting DD maintains the same solution set but can potentially have a considerably smaller number of nodes.

Formally, a *reduced* DD $\mathcal{D}$ is the smallest network (with respect to number of nodes) that represents the set of points $\text{Sol}(\mathcal{D})$ for a given variable ordering. Given any DD $\mathcal{D}$ we can construct a reduced DD in polynomial-time over the number of nodes in $\mathcal{D}$ (Bryant, 1986). The main idea is to merge nodes in the same layer that have equivalent solutions. Intuitively, we say that two nodes $u, u' \in \mathcal{N}_i$ represent equivalent solutions if the solution set given by the $u-\mathbf{t}$ and $u'-\mathbf{t}$ paths are equal. Thus, $u, u' \in \mathcal{N}_i$ have equivalent solutions, $u \sim u'$, if for each arc $a \in \mathcal{A}^{\text{out}}(u)$ there exists an arc $a' \in \mathcal{A}^{\text{out}}(u')$ with equal value and target node, i.e., $v_a = v_{a'}$ and $t(a) = t(a')$, and vice-versa. We consider $[u] = \{u' \in \mathcal{N}_i : u' \sim u\}$ as the set of equivalent solution nodes given any node $u \in \mathcal{N}_i$.

Algorithm 2 illustrates a variant of DD reduction procedure introduced by Bryant (1986) that is

Algorithm 2 DD Reduction Procedure

1: **procedure** ReduceDD($\mathcal{D}$)
2: $i = n$
3: **while** $i \geq 0$ **do**
4: **for** $u \in \mathcal{N}_i$ **do**
5: Find all equivalent nodes to u, i.e., all $u' \in [u]$
6: Redirect incoming arcs, i.e., for each $u' \in [u]$ and $a \in \mathcal{A}^{\text{in}}(u')$ set $t(a) = u$
7: Eliminate equivalent nodes, i.e., $\mathcal{N}_i = (\mathcal{N}_i \setminus [u]) \cup \{u\}$
8: $i = i - 1$
9: **return** $\mathcal{D}$

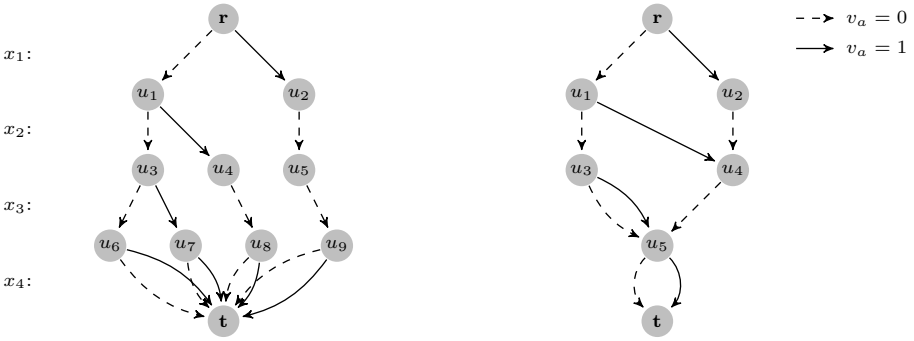


Figure 2.3: Two exact DD for $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$. The right graph is the reduced DD obtained after applying Algorithm 2 to the left DD.

better suited for discrete optimization (Bergman et al., 2016a). Specifically, we omit the step that removes redundant nodes and replaces them with long arcs. While this step can further reduce the nodes in each layer, it changes the semantics of the DD presentation since long arcs group several variable assignments. We avoid long arcs because the layered DD representation is more appropriate for the procedures presented in this dissertation (e.g., Lagrangian duality and network flow models).

Therefore, the reduction procedure employed in this dissertation is as follows. Starting from the second to last layer, we iterate over each layer identifying and merging equivalent nodes (lines 2-5). Specifically, for every node $u \in \mathcal{N}$, the procedure redirects all incoming arcs of nodes in $[u]$ to a single node and eliminates the remaining nodes in the set (lines 6-7).

Example 2.5 Consider the knapsack problem introduced in Example 2.1 and two exact DDs for its feasibility set depicted in Figure 2.3. The right most DD, $\mathcal{D}_2$, in Figure 2.3 is a reduced DD obtained when applying Algorithm 2 to the left most DD, $\mathcal{D}_1$. Notice that in the fourth layer all nodes are equivalent and are replaced by a single node u_5 in the reduced DD. Similarly, there are two equivalent nodes in the third layer of $\mathcal{D}_1$ after the fourth layer reduction (i.e., $u_4 \sim u_5$) which are replaced by a single node u_4 in the reduced DD. □

As illustrated in Example 2.5, the reduction algorithm significantly decreases the size of the diagrams both in terms of nodes and edges. Despite this reduction, the size of a reduced DD can also be exponential in the number of variables n (Bergman and Cire, 2016c). Moreover, the procedure requires an initial DD that can be significantly larger than the resulting reduced DD. In the following section, we present a DD variant that tackles this problem by introducing infeasible paths to limit the diagram size.

2.2 Relaxed DDs

So far we considered an exact DD $\mathcal{D}$ for $\mathcal{X}$ (i.e., $\mathcal{X} = \text{Sol}(\mathcal{D})$) which can have exponential size in the number of variables n. To overcome this problem, Andersen et al. (2007) propose to over-approximate $\mathcal{X}$ using a relaxed DD, i.e., a DD $\mathcal{D}$ with solution set such that $\mathcal{X} \subseteq \text{Sol}(\mathcal{D})$. Thus, every point in $\mathcal{X}$ maps to a $\mathbf{r}-\mathbf{t}$ path in $\mathcal{D}$ but the converse is not necessarily true. Relaxed DDs allow us to construct a discrete relaxation of $\mathcal{X}$ that can be used, for instance, to compute bounds for an optimization problem (see Section 2.2.4).

Recall that each node in an exact DD $\mathcal{D}$ is associated with a unique state. However, while exact DD nodes corresponds to states in ROP, a node in a relaxed DD represents *relaxed states*, i.e., the union of multiple states from the same stage. Thus, we construct a limited-size relaxed DD that over-approximates $\mathcal{X}$ by representing relaxed states in each DD node.

We present relaxed states following the definitions and formalism introduced by Hooker (2017). Given two states $\boldsymbol{S}, \boldsymbol{S}' \in \mathcal{S}_i$, we define $\widetilde{\boldsymbol{S}} = \boldsymbol{S} \oplus \boldsymbol{S}'$ as its merged state where $\oplus$ is an appropriate merging operator, i.e., the following properties hold for any stage $i \in I$:

(C1) The set of feasible assignments over states $\boldsymbol{S}$ and $\boldsymbol{S}'$ are also feasible over $\widetilde{\boldsymbol{S}}$, i.e., $X_i(\boldsymbol{S}), X_i(\boldsymbol{S}') \subseteq X_i(\widetilde{\boldsymbol{S}})$.

(C2) The immediate cost at state $\widetilde{\boldsymbol{S}}$ is greater than or equal to the immediate cost at states $\boldsymbol{S}$ and $\boldsymbol{S}'$. Thus, for any $x \in X_i(\boldsymbol{S})$ and $x' \in X_i(\boldsymbol{S}')$ we have $f_i(\boldsymbol{S}, x) \leq f_i(\widetilde{\boldsymbol{S}}, x)$ and $f_i(\boldsymbol{S}', x') \leq f_i(\widetilde{\boldsymbol{S}}, x')$, respectively.

Following the above conditions, we say that $\widetilde{\boldsymbol{S}}$ relaxes a state $\boldsymbol{S} \in \mathcal{S}_i$ if $X_i(\boldsymbol{S}) \subseteq X_i(\widetilde{\boldsymbol{S}})$ and the immediate cost function over any $x \in X_i(\boldsymbol{S})$ is larger for $\widetilde{\boldsymbol{S}}$ (i.e., $f_i(\boldsymbol{S}, x) \leq f_i(\widetilde{\boldsymbol{S}}, x)$). These two properties (C1) and (C2) are necessary (but not sufficient) conditions to define a proper relaxation of ROP. Hooker (2017) shows that operator $\oplus$ defines a proper relaxation for ROP if, in addition to (C1) and (C2), we impose a condition over the transition function:

(C3) If $\widetilde{\boldsymbol{S}}$ relaxes state $\boldsymbol{S} \in \mathcal{S}_i$, then, given any value $x \in X_i(\boldsymbol{S})$, $\phi_i(\widetilde{\boldsymbol{S}}, x)$ relaxes state $\phi_i(\boldsymbol{S}, x)$, for all $i \in I$. Thus, $\widetilde{\boldsymbol{S}}$ defines a relaxed state in the following stage for each feasible decision variable assignment.

Thus, we can construct a valid relaxation for ROP (i.e., over-approximate its feasible set and cost) with any merging operator $\oplus$ that satisfies conditions (C1)-(C3). Notice that these conditions are quite general and, as such, they allow us to construct relaxed DDs for a wide range of problems. Example 2.6 shows a merging operator and relaxed state for the knapsack problem. In addition, Sections 2.2.2 and 2.2.3 illustrate merging operators for two common problem structures in the literature.

Example 2.6 Consider the knapsack problem introduced in Example 2.1 with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and recursive model R-KNP. We define the merge operator over states $Q, Q' \in \mathcal{S}_i$ at stage $i \in \{1, \ldots, 4\}$ as the minimum over both quantities, i.e., $Q \oplus Q' = \min\{Q, Q'\}$.

Notice that operator $\oplus$ satisfies (C1) since any solution feasible for a knapsack load Q or Q' is also feasible for $\min\{Q, Q'\}$. Condition (C3) also holds for $\oplus$ since the transition function is an increasing function over Q (see Example 2.2). Lastly, (C2) holds since the cost function $f_i(Q, x) = c_i x$ is independent of the current state Q. □

2.2.1 Construction Procedures

We now present two relaxed DD construction procedures employed in the literature (Bergman et al., 2016a). Both algorithms construct relaxed DDs by limiting the number of nodes in each layer. Formally, the width of a DD $w(\mathcal{D})$ is the maximum number of nodes in each layer, i.e., $w(\mathcal{D}) = \max\{|\mathcal{N}_i| : i \in I\}$. Thus, we limit the size of $\mathcal{D}$ by bounding its width with a value $\mathcal{W} \in \mathbb{Z}^+$ as $w(\mathcal{D}) \leq \mathcal{W}$.

The first construction procedure corresponds to a slight variant of the top-down algorithm for exact DDs (see Algorithm 1). The main idea is to introduce an additional step that merges nodes when the maximum width is exceeded (Bergman et al., 2011). As in Algorithm 1, we construct a relaxed DD starting from the root node $\mathbf{r}$ and create nodes for reachable states at each stage in consecutive index order. The procedure continues until we reach a layer where the number of nodes is larger than the maximum width, i.e., $|\mathcal{N}_i| > \mathcal{W}$ for some $i \in I$. In such case, we employ a merging procedure that selects and merges nodes until we satisfy the maximum-width requirement.

This top-down algorithm has been successfully used to create strong relaxations, e.g., for the set covering problem (Bergman et al., 2011). However, its naive version merges nodes without information on how this will affect the relaxation in the following stages. While there exist alternatives that include a look-ahead step to decide which nodes to merge, these procedures can be quite computationally expensive (Horn et al., 2018).

An alternative is to construct a relaxed DD using the iterative refinement procedure (Andersen et al., 2007). In contrast to the top-down algorithm, this procedure starts with an initial relaxed DD and iteratively increase its width by splitting nodes, i.e., we refine the DD in each iteration. The main advantage is that we can use information on the resulting DD to guide the refinement in the next iteration. Examples of this idea include satisfying multiple constraints over the DD one at a time (Ciré and Hooker, 2014) and iteratively improve DD bounds (Bergman and Cire, 2016c).

Algorithm 3 Relaxed DD Iterative Refinement Construction Procedure

```
1:  procedure ConstructDD(ROP, 𝒲, ⊕)
2:      𝒟 = WidthOneDD({D_1, ..., D_n})
3:      while 𝒟 has been modified do
4:          for i ∈ I do
5:              UpdateDDNodesTop(𝒩_i)
6:              SplitDDNodes(𝒩_i, 𝒲)
7:              FilterDDArcs(𝒩_i)
8:          UpdateDDNodesTop(𝒩_{n+1})
9:          UpdateDDBottom(𝒟)              %% Optional step %%
10:     return 𝒟
```

Algorithm 3 illustrates the iterative refinement procedure given a recursive model ROP, a maximum width $\mathcal{W}$, and a merging operator $\oplus$. The procedure starts by constructing a width-one DD, i.e., a DD $\mathcal{D}$ where each layer $\mathcal{N}_i$, with $i \in I$, has a single node and emanating arcs for each value in D_i. The root node is associated with the initial state (i.e., $\boldsymbol{S}(\mathbf{r}) = \boldsymbol{S}_1$) and each following node corresponds to a relaxed state computed using the merging operator and its incoming arcs:

$$\boldsymbol{S}(u) = \bigoplus_{a \in \mathcal{A}^{\text{in}}(u)} \phi_{i-1}(\boldsymbol{S}(s(a)), v_a), \qquad \forall u \in \mathcal{N}_i,\ i \in \{2, \ldots, n+1\}. \tag{2.1}$$

Then, the procedure iteratively refines $\mathcal{D}$ until it cannot be updated any further (lines 3-8). The

refinement is done one layer at a time in increasing index order and it is divided into three sub-routines: update relaxed states in each node (UpdateDDNodesTop), split nodes (SplitDDNodes), and filter infeasible arcs (FilterDDArcs). The UpdateDDNodesTop($\mathcal{N}_i$) routine iterates over all the nodes in layer $\mathcal{N}_i$ and updates their relaxed states using equation (2.1).

As shown in Algorithm 4, the SplitDDNodes($\mathcal{N}_i$) sub-routine iterates over the nodes in layer $\mathcal{N}_i$ and splits them into two (or more) nodes to improve the DD relaxation. Intuitively, the procedure chooses nodes that have relaxed states representing the union of two or more states. The decision on how to choose and split nodes is problem specific and generally done heuristically. For each chosen node $u \in \mathcal{N}_i$, the procedure partitions its set of incoming arcs and redirects one partition to a newly created node $u' \in \mathcal{N}_i$ (lines 4-6). We create emanating arcs for u' by duplicating the emanating arcs of u, i.e., these new arcs have the same value and target nodes as the arcs in $\mathcal{A}^{\text{out}}(u)$. This last step guarantees that the modified DD has the same set of solutions. The procedure ends when we reach the width limit or there are no more nodes to split.

Algorithm 4 Relaxed DD Split Nodes Procedure

1: **procedure** SplitDDNodes($\mathcal{N}_i$, $\mathcal{W}$)
2: **for** $u \in \mathcal{N}_i$ **do**
3: **if** $\boldsymbol{S}(u)$ represents the union of two or more states **then**
4: Partition $\mathcal{A}^{\text{in}}(u)$ into two sets, i.e., $\mathcal{A}^{\text{in}}(u) = \mathcal{A}_1 \cup \mathcal{A}_2$ and $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$
5: Create a new node u' in layer $\mathcal{N}_i$
6: Redirect arcs in $\mathcal{A}_2$ to u', i.e., for all arcs $a \in \mathcal{A}_2$, $t(a) = u'$
7: Duplicate $\mathcal{A}^{\text{out}}(u)$, and set their source nodes to be u', i.e., $\mathcal{A}^{\text{out}}(u') = \mathcal{A}^{\text{out}}_{dup}(u)$
8: **if** $|\mathcal{N}_i| = \mathcal{W}$ **then return**

The third sub-routine in the iterative refinement algorithm is FilterDDArcs($\mathcal{N}_i$). This procedure iterates over the arcs emanating from layer $\mathcal{N}_i$ to identify and remove *infeasible arcs*. Formally, we say that an arc $a \in \mathcal{A}$ is infeasible if all $\mathbf{r} - \mathbf{t}$ paths traversing a correspond to infeasible solutions, i.e., $\boldsymbol{x}^p \notin \mathcal{X}$ for all $\mathbf{r} - \mathbf{t}$ paths $p \in \mathcal{P}$ containing arc a. We identify infeasible arcs by checking a set of conditions (i.e., filter rules) defined over the relaxed state in their source nodes. These rules are problem specific and, as shown in Example 2.7, their effectiveness is highly dependent on number of states that each relaxed state represents.

Example 2.7 Consider the knapsack problem with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and recursive model R-KNP. We construct a relaxed DD for $\mathcal{X}$ as follows. For each node $u \in \mathcal{N}_i$, we consider a relaxed states $\boldsymbol{S}(u) = (Q_{\min}(u), Q_{\max}(u))$ where $Q_{\min}(u)$ and $Q_{\max}(u)$ represent the minimum and maximum load of the knapsack at node u and stage $i \in \{1, \ldots, 5\}$, respectively. Thus, the merging operator is given by $\oplus = (\min, \max)$. We choose this relaxed state representation to identify if a node encodes multiple states and, therefore, if it is a candidate for splitting. Intuitively, a node $u \in \mathcal{N}$ represents a single state of R-KNP if $Q_{\min}(u) = Q_{\max}(u)$.

Lastly, we identify if an arc $a \in \mathcal{A}$ with source $s(a) \in \mathcal{N}_i$ is infeasible if

$$Q_{\min}(s(a)) + w_i v_a > 8 \tag{KP-R1}$$

for any $i \in I$. If arc a satisfies filtering rule KP-R1, then all paths traversing a represent solutions with a knapsack load above its limit. Thus, we can remove arc a from $\mathcal{D}$.

Figure 2.4 illustrates the iterative refinement procedure for R-KNP with maximum width $\mathcal{W} = 2$

and relaxed states as defined above. The left most DD corresponds to a width-one DD for this problem, where the values next to each node represents its relaxed state. The middle DD depicts the update, split, and filter sub-routines in layer $\mathcal{N}_2$. Node $u_1 \in \mathcal{N}_2$ is split into two (i.e., u_1 and $\bar{u}_1$), each node with one incoming arc and relaxed states updated accordingly. Also, notice that arc $a = (u_1, \bar{u}_1)$ with value $v_a = 1$ is infeasible, since it satisfies KP-R1. The right graph shows the resulting DD after refining layer $\mathcal{N}_3$. In this DD, filtering rule KP-R1 identifies infeasible arc $a = (\bar{u_2}, u_3)$ with value $v_a = 1$.

While filtering rule KP-R1 identifies several infeasible arcs during the DD construction procedure, there exist paths in the right most DD of Figure 2.4 that correspond to infeasible solutions, e.g., path $(\mathbf{r}, u_1, u_2, u_3, \mathbf{t})$ associated with point $\boldsymbol{x} = (0, 1, 1, 1)$ is infeasible. This path cannot be removed from the DD since all its arcs are also associated with feasible solutions, i.e., all the arcs violate KP-R1. We can remove this path from the DD if we further split node u_2. □

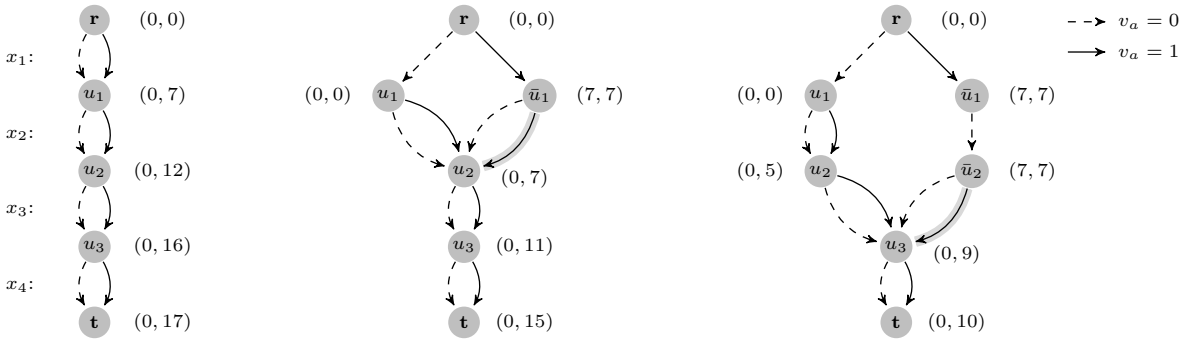


Figure 2.4: Relaxed DD construction procedure for $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and $\mathcal{W} = 2$. The figure shows a width-one DD (left), a DD after splitting layer $\mathcal{N}_2$ (middle), and a DD after splitting layer $\mathcal{N}_3$ (right). Values next to nodes represent relaxed states. Highlighted arrows correspond to infeasible arcs that can be removed.

We can construct a relaxed DD for any recursive model ROP with the sub-routines described so far and an appropriate merging operator. However, as in the top-down construction case, the relaxed states store information from the current and preceding stages, but not from the remaining stages. An alternative is to create additional relaxed states for each node $u \in \mathcal{N}$ that take into account the decisions made over all $u - \mathbf{t}$ paths, i.e., *bottom-up* relaxed states. The main idea is to use bottom-up relaxed states to strengthen the relaxation by creating additional filtering rules.

Algorithm 5 Bottom-Up Procedure for Iterative Refinement

```
1: procedure UpdateDDBottom(𝒟)
2:     i = n + 1
3:     while i ≥ 0 do
4:         UpdateDDNodesBottom(𝒩_i)
5:         FilterDDArcs(𝒩_{i−1})
6:         i = i − 1
```

Algorithm 5 depicts the bottom-up procedure, UpdateDDBottom($\mathcal{D}$). The algorithm iterates over each layer in reverse order updating the bottom-up states in each layer (i.e., UpdateDDNodesBottom($\mathcal{N}_i$)) and filtering its incoming arcs (i.e., FilterDDArcs($\mathcal{N}_{i-1}$)). The bottom-up states, denote by $\boldsymbol{S}^{\uparrow}$, are usually defined similarly to the top-down relaxed states considering the recursion in inverse order. Specifically, given a bottom-up transition function $\phi_i^{\uparrow}(\cdot)$ and merging operator $\oplus$, we update the bottom-up states

for every DD node as

$$\boldsymbol{S}^{\uparrow}(u) = \bigoplus_{a \in \mathcal{A}^{\text{out}}(u)} \phi_i^{\uparrow}(\boldsymbol{S}^{\uparrow}(t(a)), v_a), \qquad \forall u \in \mathcal{N}_i,\ i \in \{n, n-1, \dots, 1\}, \tag{2.2}$$

and associate the terminal node with an appropriate initial bottom-up state, i.e., $\boldsymbol{S}^{\uparrow}(\mathbf{t}) = \boldsymbol{S}^{\uparrow}_{n+1}$. We can then create new filtering rules that take into account the information store in the bottom-up states to identify additional infeasible arcs. Example 2.8 illustrates bottom-up states and a filtering rule for the knapsack problem. Sections 2.2.2 and 2.2.3 present additional examples of relaxed states (top-down and bottom-up) and filtering rules for problem structure that emerge in many applications.

Example 2.8 Consider the knapsack problem with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and recursive model R-KNP. Notice that we can change the variable ordering of recursion R-KNP without altering the set of feasible solutions and optimal value. Then, we use this fact to create the bottom-up state $\boldsymbol{S}^{\uparrow}(u) = Q^{\uparrow}_{\min}(u)$ as the minimum knapsack load for all partial solutions represented by $u - \mathbf{t}$ paths. The bottom-up updates follow the same structure as the top-down updates shown in Example 2.7. Thus, we set $Q^{\uparrow}_{\min}(\mathbf{t}) = 0$ and update the bottom-up states as:

$$Q^{\uparrow}_{\min}(u) = \min_{a \in \mathcal{A}^{\text{out}}(u)} Q^{\uparrow}_{\min}(t(a)) + w_i v_a, \qquad \forall u \in \mathcal{N}_i,\ i \in \{4, 3, \dots, 1\}.$$

Notice that for any arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$, the expression $Q_{\min}(s(a)) + w_i v_a + Q^{\uparrow}_{\min}(t(a))$ underestimates the load of the knapsack given by all $\mathbf{r} - \mathbf{t}$ paths traversing arc a. Then, the following condition is a valid filtering rule for the problem:

$$Q_{\min}(s(a)) + w_i v_a + Q^{\uparrow}_{\min}(t(a)) > 8.$$

Thus, all arcs satisfying this condition are only part of solutions that violate the knapsack constraint. □

2.2.2 Special Case: Separable Inequalities

We now present the relaxed states and filtering rules for feasible sets defined by separable functions (Andersen et al., 2007; Hoda et al., 2010). Consider $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{Z}^n : \sum_{i \in I} g_i(x_i) \leq b\}$ where $g_i : D_i \to \mathbb{R}$ is a real function for each $i \in I$ and $b \in \mathbb{R}$ is the right-hand-side constant. Notice that separable inequalities arise in a wide range of problems, e.g., problems defined by linear constraints. Here we assume that $\mathcal{X}$ has a single inequality but the relaxed states and filtering rules can be extended for multiple inequalities.

As for our knapsack running example, we define $\mathcal{X}$ recursively considering a single state variable $\boldsymbol{S} = Q$ that represents value $\sum_{j=1}^{i-1} g_j(x_j)$ at stage $i \in \{2, \dots, n+1\}$. Then, the initial state is $\boldsymbol{S}_1 = 0$ and the transition function is given by $\phi_i(Q, x) = Q + g_i(x)$ for all $i \in I$. The set of feasible decision is given by $X_i(Q) = D_i$ for stages $i \in \{1, \dots, n-1\}$ and $X_n(Q) = \{x \in D_n : Q + g_n(x) \leq b\}$ for the last decision variable.

Similarly to Example 2.7, we define two (top-down) relaxed states $\boldsymbol{S}(u) = (Q_{\min}(u), Q_{\max}(u))$ for each node $u \in \mathcal{N}_i$ that respectively under-estimate and over-estimate the value of Q considering all $\mathbf{r} - u$ paths. We compute these values using the merge operator $\oplus = (\min, \max)$ and the initial state

$\boldsymbol{S}(\mathbf{r}) = (0, 0)$. Then, the relaxed states for each node $u \in \mathcal{N}_i$ are updated recursively as follows:

$$Q_{\min}(u) = \min_{a \in \mathcal{A}^{\text{in}}(u)} \{Q_{\min}(s(a)) + g_{i-1}(v_a)\}, \tag{2.3}$$

$$Q_{\max}(u) = \max_{a \in \mathcal{A}^{\text{in}}(u)} \{Q_{\max}(s(a)) + g_{i-1}(v_a)\}. \tag{2.4}$$

These two relaxed states can be used during the splitting procedure to identify nodes that represent multiple states, i.e., any node $u \in \mathcal{N}$ with $Q_{\min}(u) < Q_{\max}(u)$. Also, we use the relaxed states to create a filtering rule for any arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_n$:

$$Q_{\min}(s(a)) + g_n(v_a) > b. \tag{SI-R1}$$

The validity of this rule follows from the relaxed state updates (2.3) and the inequality defining $\mathcal{X}$. However, rule SI-R1 only detects infeasible arcs in the second-to-last layer and its effectiveness depends on the node relaxation degree in that layer (i.e., the $Q_{\max}(u) - Q_{\min}(u)$ difference). Fortunately, we can introduce bottom-up states for this problem structure and create filtering rules that can identify infeasible arcs in any layer.

We define bottom-up states for our relaxed DD following the same ideas introduced in the knapsack problem (see Example 2.8). Consider the bottom-up state $\boldsymbol{S}^{\uparrow}(u) = Q^{\uparrow}_{\min}(u)$ for any node $u \in \mathcal{N}_i$ that represent the minimum value of $\sum_{j=i}^{n} g_j(x_j)$ for all $u-\mathbf{t}$ paths. Thus, we assign $Q^{\uparrow}_{\min}(\mathbf{t}) = 0$ and update the bottom-up states for every node $u \in \mathcal{N}_i$ and stage $i \in I$ as

$$Q^{\uparrow}_{\min}(u) = \min_{a \in \mathcal{A}^{\text{out}}(u)} \{Q^{\uparrow}_{\min}(t(a)) + g_i(v_a)\}. \tag{2.5}$$

For a given arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$, we under-approximate the last-stage state value $Q \in \mathcal{S}_{n+1}$ for all paths traversing a using the top-down and bottom-up states as $Q_{\min}(s(a)) + g_i(v_a) + Q^{\uparrow}_{\min}(t(a))$. The validity of this bound follows directly from the state updates (2.3) and (2.5). Thus, the following inequality is a valid filtering rule to identify infeasible arcs emanating from layer $\mathcal{N}_i$ for any $i \in I$:

$$Q_{\min}(s(a)) + g_i(v_a) + Q^{\uparrow}_{\min}(t(a)) > b. \tag{SI-R2}$$

Note that SI-R1 is a special case of SI-R2 since the terminal node has bottom-up state $Q^{\uparrow}_{\min}(\mathbf{t}) = 0$. Moreover, filtering rule SI-R2 identifies all infeasible arcs with respect to $\mathcal{X}$ (Andersen et al., 2007), a property known as *DD consistency* in the CP literature (see Section 2.3.1).

2.2.3 Special Case: All-Different Structure

We now present relaxed states and filtering rules for combinatorial structures where all variables have to take different values (Andersen et al., 2007; Hoda et al., 2010), i.e., $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{Z}^n : x_i \neq x_j, \text{ for any } i, j \in I,\ i \neq j\}$. This structure is known as the `All-different` global constraint in the CP literature and has many applications, including sequencing and assignment problems (Rossi et al., 2006). We represent set $\mathcal{X}$ considering a state variable $\boldsymbol{S} = E \in \mathcal{S}_i$ as the set of values assigned to variables at stage $i \in I$, i.e., $E \subseteq D_1 \cup \cdots \cup D_i$ with cardinality $|E| = i$. The initial state is the empty set $\boldsymbol{S}_1 = \emptyset$, i.e., no variable has been assigned to a value yet. For each stage $i \in I$, the transition function $\phi_i(E, x) = E \cup \{x\}$ adds the value assigned to x to set E and the feasible set $X_i(E) = \{x \in D_i : x \notin E\}$ enforce x to take a value

different to the ones in E.

We consider two relaxed states $\boldsymbol{S}(u) = (E_{\text{all}}(u), E_{\text{some}}(u))$ to approximate E at node $u \in \mathcal{N}$. Specifically, $E_{\text{all}}(u)$ represents the set of values that appear in *all* $\mathbf{r} - u$ paths, while $E_{\text{some}}(u)$ corresponds to the set of values that are assigned to at least *some* arc in a $\mathbf{r} - u$ paths. Then, we set the initial state as $\boldsymbol{S}(\mathbf{r}) = (\emptyset, \emptyset)$ and update the relaxed states for a node $u \in \mathcal{N} \setminus \{\mathbf{r}\}$ as follows:

$$E_{\text{all}}(u) = \bigcap_{a \in \mathcal{A}^{\text{in}}(u)} \{E_{\text{all}}(s(a)) \cup \{v_a\}\}, \tag{2.6}$$

$$E_{\text{some}}(u) = \bigcup_{a \in \mathcal{A}^{\text{in}}(u)} \{E_{\text{some}}(s(a)) \cup \{v_a\}\}. \tag{2.7}$$

Notice that a relaxed state $\boldsymbol{S}(u) = (E_{\text{all}}(u), E_{\text{some}}(u))$ at node $u \in \mathcal{N}$ represent a single exact state if $E_{\text{all}}(u) = E_{\text{some}}(u)$. Thus, we can use the relaxed states to identify possible nodes to split, i.e., whenever node u has $E_{\text{all}}(u) \subset E_{\text{some}}(u)$. Moreover, we can identify if an arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$ is infeasible if one of the following filtering rules hold:

$$v_a \in E_{\text{all}}(s(a)), \tag{AD-R1}$$

$$|E_{\text{some}}(s(a))| = i \text{ and } v_a \in E_{\text{some}}(s(a)). \tag{AD-R2}$$

The validity of AD-R1 follows directly from the `All-different` structure $\mathcal{X}$. Condition AD-R2 states that if all $\mathbf{r} - s(a)$ paths have i variables and at most i different values, then arc a needs to have a value outside of $E_{\text{some}}(s(a))$, otherwise all paths traversing a repeat at least one value.

As for the separable inequality case, we define bottom-up states considering the recursion in inverse order. Consider bottom-up states $\boldsymbol{S}^{\uparrow}(u) = (E^{\uparrow}_{\text{all}}(u), E^{\uparrow}_{\text{some}}(u))$ for node $u \in \mathcal{N}$ as the set of values that appear in all or at least some $u - \mathbf{t}$ path, respectively. Thus, the terminal node has relaxed state $\boldsymbol{S}^{\uparrow}(\mathbf{t}) = (\emptyset, \emptyset)$ and the bottom-up states at node $u \in \mathcal{N} \setminus \{\mathbf{t}\}$ are given by:

$$E^{\uparrow}_{\text{all}}(u) = \bigcap_{a \in \mathcal{A}^{\text{out}}(u)} \{E^{\uparrow}_{\text{all}}(t(a)) \cup \{v_a\}\}, \tag{2.8}$$

$$E^{\uparrow}_{\text{some}}(u) = \bigcup_{a \in \mathcal{A}^{\text{out}}(u)} \{E^{\uparrow}_{\text{some}}(t(a)) \cup \{v_a\}\}. \tag{2.9}$$

We use the bottom-up states to define the following filtering rules to identify infeasible arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$ for any $i \in I$:

$$v_a \in E^{\uparrow}_{\text{all}}(t(a)), \tag{AD-R3}$$

$$|E^{\uparrow}_{\text{some}}(t(a))| = n - i \text{ and } v_a \in E^{\uparrow}_{\text{some}}(t(a)), \tag{AD-R4}$$

$$|E_{\text{some}}(s(a)) \cup \{v_a\} \cup E^{\uparrow}_{\text{some}}(t(a))| < n. \tag{AD-R5}$$

Filtering rules AD-R3 and AD-R4 are analogous to AD-R1 and AD-R2, respectively, but over the bottom-up states instead. Rule AD-R5 checks whether all paths traversing a repeat a value by comparing the number of different values and variables.

2.2.4 Objective Bounds and Filtering

So far we have introduced how to create relaxed DDs for different combinatorial structures ignoring their objective functions. We now introduce two common objective functions and show how to create valid relaxations for the recursive model ROP. Moreover, we present cost-based filtering rules to remove sub-optimal solutions from the relaxed DDs. We first introduce the simplest case where we have a state-independent objective function and then show a procedure for the sum of set-up times objective for sequencing problems.

State-Independent Objective

Consider a recursive maximization problem ROP and an objective function that is independent of the current state, i.e., $f_i(\boldsymbol{S}, x) = f_i(x)$ for any state $\boldsymbol{S} \in \mathcal{S}_i$ and stage $i \in I$. Examples of this objective include linear functions $f(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x}$ where $f_i(x) = c_i x_i$. Note that in this case there is no need to impose condition (C3) over the merging operator $\oplus$ since the immediate cost is independent of the (relaxed) state.

We compute an upper bound for ROP using a longest-path procedure over $\mathcal{D}$ (Andersen et al., 2007). For each arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$ with $i \in I$, we assign arc a a length $\ell_a = f_i(v_a)$. Every node $u \in \mathcal{N}$ has a value $h(u)$ that corresponds to the longest-path value for all $\mathbf{r} - u$ path. The longest-path values are calculated traversing the DD once in a top-down fashion by setting $h(\mathbf{r}) = 0$ and using the following equation:

$$h(u) = \max_{a \in \mathcal{A}^{\text{in}}(u)} \{h(s(a)) + \ell_a\}, \qquad \forall u \in \mathcal{N} \setminus \{\mathbf{r}\}. \tag{2.10}$$

The longest-path value is hence given by $h(\mathbf{t})$. Since $\mathcal{D}$ is a relaxed DD for $\mathcal{X}$ (i.e., $\mathcal{X} \subseteq \text{Sol}(\mathcal{D})$), the longest-path value over $\mathcal{D}$ is a valid upper bound for ROP:

$$h(\mathbf{t}) = \max_{\boldsymbol{x} \in \text{Sol}(\mathcal{D})} f(\boldsymbol{x}) \geq \max_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}).$$

Besides bound computation, we can also refine the relaxed DD $\mathcal{D}$ by removing arcs that represent sub-optimal solutions with respect to any given lower bound lb (Hadzic and Hooker, 2007). To do so, we consider a bottom-up version of the longest-path procedure and apply a filtering rule similar to SI-R2. To this end, consider $h^\uparrow(u)$ as the longest-path value for all $u - \mathbf{t}$ paths, where $h^\uparrow(\mathbf{t}) = 0$ and

$$h^\uparrow(u) = \max_{a \in \mathcal{A}^{\text{out}}(u)} \{h^\uparrow(t(a)) + \ell_a\}, \qquad \forall u \in \mathcal{N} \setminus \{\mathbf{t}\}. \tag{2.11}$$

We identify and remove sub-optimal paths from $\mathcal{D}$ using a cost-based filtering. Given any arc $a \in \mathcal{A}$, all $\mathbf{r} - \mathbf{t}$ paths traversing a correspond to sub-optimal solutions if the following condition holds:

$$h(s(a)) + \ell_a + h^\uparrow(t(a)) < lb. \tag{CB-R1}$$

The right-hand-side of CB-R1 corresponds to the longest-path value for all $\mathbf{r} - \mathbf{t}$ paths traversing arc a. Thus, if this value is smaller than our lower bound lb, we can deduce that all paths traversing a represent sub-optimal solutions.

Minimizing the Sum of Setup Times

Relaxed DDs has been successfully applied to many sequencing problems to obtain stronger bounds when compared to linear relaxations (Hooker, 2017, 2019). One of the most common objectives for these problems is minimize the sum of setup times. We introduce this objective over a simple sequencing problem and present a shortest-path procedure to get valid lower bounds (Cire and van Hoeve, 2012, 2013).

Consider a sequencing problem defined by a set of jobs $J = \{1, \dots, n\}$ that need to be executed one at a time in a single machine. The set of feasible solutions can be modeled by an `All-different` structure as $\mathcal{X} = \{\boldsymbol{x} \in J^n : x_i \neq x_j, \text{ for any } i, j \in I,\ i \neq j\}$ where x_i represents the i-th job executed for all $i \in I$. Thus, we can create a relaxed DD for $\mathcal{X}$ using the relaxed states and filtering rules from Section 2.2.3.

The sum of setup times objective considers a cost $c_{j,j'}$ that represents the setup time to process job j' immediately after job j, for each $j, j' \in J$, $j \neq j'$. The objective is to find a feasible sequence $\boldsymbol{x} \in \mathcal{X}$ such that the total setup time $f(\boldsymbol{x}) = \sum_{i=2}^{n} c_{x_{i-1},x_i}$ is minimized. Notice that this cost structure depends on the previous and current decision, so the states of a recursive model must save the last decision made (Held and Karp, 1962). However, we can compute lower bounds for this problem without any additional state information by considering a cumulative arc length instead (Cire and van Hoeve, 2013). Each arc $a \in \mathcal{A}$ has a length that represents the minimum sum of setup times considering all $\mathbf{r} - t(a)$ paths that contain arc a. The arc length for any arc $a \in \mathcal{A}$ is given by

$$\ell_a = \begin{cases} 0, & a \in \mathcal{A}^{\text{out}}(\mathbf{r}), \\ \min_{a' \in \mathcal{A}^{\text{in}}(s(a))} \{\ell_{a'} + c_{v_{a'},v_a}\}, & \text{otherwise.} \end{cases} \tag{2.12}$$

A lower bound for this problem is given by the minimum arc length in the last layer, i.e., $\min\{\ell_a : a \in \mathcal{A}^{\text{in}}(\mathbf{t})\}$. Conversely, we can compute the same bound by traversing the relaxed DD in inverse order as follows. Consider $\ell_a^{\uparrow}$ as the minimum sum of setup times considering all $s(a) - \mathbf{t}$ paths that contain arc a, i.e.,

$$\ell_a^{\uparrow} = \begin{cases} 0, & a \in \mathcal{A}^{\text{in}}(\mathbf{t}), \\ \min_{a' \in \mathcal{A}^{\text{out}}(t(a))} \{\ell_{a'}^{\uparrow} + c_{v_a,v_{a'}}\}, & \text{otherwise.} \end{cases} \tag{2.13}$$

Notice that we can use the arc length values given by (2.12) and (2.13) to compute the minimum setup time for all $\mathbf{r} - \mathbf{t}$ paths traversing an arc a as the sum $\ell_a + \ell_a^{\uparrow}$. Thus, for a given upper bound ub, we can identify and remove arc $a \in \mathcal{A}$ representing sub-optimal solutions if the bellow cost-based filtering rule holds:

$$\ell_a + \ell_a^{\uparrow} > ub. \tag{CB-R2}$$

While here we present the most basic sum of setup times cost structure, this can be extended, for example, by considering release and due dates (Cire and van Hoeve, 2013). In addition, Kinable et al. (2017) extended this cost structure for position dependent cost, i.e., the setup time $c_{j,j'}^{i}$ for any two distinct jobs $j, j' \in J$ depends on the sequence position i. In such case, equations (2.12) and (2.13) need to consider the layer of the preceding arcs.

2.3 Integration of DDs with Existing Technologies

This section briefly introduces how to use DDs with two methodologies to solve discrete optimization problems: Constraint Programming (CP) and Integer Programming (IP). While here we focus on these two technologies, there exist other alternatives in the literature, e.g., using DDs as a stand-alone solver (Bergman et al., 2016b). We refer the reader to Chapter 3 for a literature review on how to employ DDs to solve discrete optimization problems.

2.3.1 Constraint Programming

CP is a discrete optimization paradigm that utilizes tree-search and logical inference over combinatorial structures to prune variables domains and find optimal solutions (Rossi et al., 2006). The feasible set $\mathcal{X}$ is given by a set of global constraints, i.e., general-form constraints that represent sub-structures of the problem. Each constraint has a propagator algorithm that, given the current domain of the variables, identifies and removes variable assignments that are infeasible. Thus, a CP solver uses tree-search to explore the space of variable assignments and employs the constraint propagators to identify parts of the space that correspond to infeasible assignments.

In this context, a DD for a combinatorial structure can be seen as a global constraint for a CP solver. The DD is constructed to have an exact (or relaxed) graphical representation of a feasible set $\mathcal{X}$. Then, given the current variable domains, we propagate this information in the DD by removing arcs and updating the graph accordingly. The new domains inferred by the DD (i.e., its possible arc values in each layer) are then communicated back to the CP solver, and the process iterates until we find an optimal solution and/or exhaust the search space.

The main advantage of DDs in comparison to existing global constraints is that they allow the encoding of multiple combinatorial structures in a single diagram. In fact, relaxed DDs where initially proposed in the CP literature as a replacement of the domain store (Andersen et al., 2007). The domain store represents all possible variable assignments and updates its values by propagating each global constraint separately during search. Conversely, a relaxed DD can encode all the constraints of the problem and, thus, capture the interaction of several constraints at once during propagation. However, constructing such relaxed DD is impractical in many problem, so researchers have mostly focus on using a DD to represent a subset of the constraints defining $\mathcal{X}$ (Bergman et al., 2016a).

Hoda et al. (2010) introduced the concept of *DD consistency* to identify the inference capabilities of a relaxed DD for a given constraint $\mathcal{C}$. Formally, we say that a relaxed DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ is consistent with respect to constraint $\mathcal{C}$ if eliminating any arc $a \in \mathcal{A}$ will result in removing a solution that is feasible for $\mathcal{C}$. In other words, we achieve DD consistency if the set of filtering rules is able to identify all infeasible arcs. This property can be achieve in polynomial time over the DD for simple problem structures, e.g., for linear constraints (see Section 2.2.2). However, achieving DD consistency is NP-hard for a wide range of constraints, e.g., for the *Sequence* global constraint (Bergman et al., 2014c).

Lastly, DDs have been used to compute bounds for weighted CP problems. The idea is to create a relaxed DD for the complete problem and then compute bounds using, for example, the longest path procedures introduced in Section 2.2.4. The DD bounds can be used in a branch-and-bound tree search to avoid exploring sub-optimal branches and prove optimality. Since CP solvers have weak bound procedures (if any), DD bounds can have a large impact on reducing the search space (Bergman et al., 2016a). Notable examples include sequencing problems (Cire and van Hoeve, 2013; Kinable et al., 2017),

where DD bounds are usually tighter than linear programming alternatives (Hooker, 2019). We refer the reader to Chapter 3 for more details on how researchers have integrate DDs in CP technologies.

2.3.2 Integer Programming

IP solvers also employ a tree-search algorithm to explore the decision space of discrete optimization problems. However, IP technologies rely on a continuous relaxation of the problem to guide the search and avoid infeasible and sub-optimal portions of the search space. In this context, DDs are graphical models of $\mathcal{X}$ that can be encoded as a linear network flow model, $\text{NF}(\mathcal{D})$, for any $\mathcal{D} = (\mathcal{N}, \mathcal{A})$. This flow model, introduce by Behle (2007), represents the convex combination of the paths in $\mathcal{P}$ as flows over $\mathcal{D}$ and, thus, can be easily integrated to any IP modeling tool. We refer the reader to Chapter 3 for a description on network flow model $\text{NF}(\mathcal{D})$.

The polytope $\text{NF}(\mathcal{D})$ has the advantage that its projection over the $\boldsymbol{x}$ variables is equivalent to the convex hull of all solutions represented by $\mathcal{D}$ (Behle (2007), Theorem 4.1). Thus, given any DD $\mathcal{D}$, the network flow model $\text{NF}(\mathcal{D})$ is an extended linear formulation for $\text{Sol}(\mathcal{D})$. This model has been used, for example, for linearization of non-linear objectives (Bergman and Cire, 2018) and to develop cut-generation procedures (Becker et al., 2005; Tjandraatmadja and van Hoeve, 2019). Also, we can use $\text{NF}(\mathcal{D})$ to employ techniques in the IP literature to enhance a DD relaxation. For example, Bergman et al. (2015b) introduce a Lagrangian relaxation procedure over DDs to penalize infeasible paths and, thus, improve DD relaxations. Further examples of DD and IP interactions can be found in Chapter 3.

Chapter 3

Literature Review

This chapter presents a survey on Decision Diagrams (DDs) for discrete optimization. We focus on papers that represent the set of solutions to a combinatorial problem with a DD, which includes encoding a subset of the constraints with an exact or approximate (i.e., relaxed or restricted) DD. Thus, we mostly consider papers related to the DD for optimization framework introduced by Bergman et al. (2016a), which covers papers published in the last two decades.

We organize this survey considering different DD-based techniques to solve discrete optimization problems, and the research advances in these directions. In particular, the type of DD (i.e., exact, relaxed, or restricted) relates to the DD procedure to tackle the problem. This difference arises from the theoretical guarantees of each DD structure. On one side, exact DDs model the entire solution space, so any property that holds for the DD is also valid for the original problem. Conversely, approximate DDs provide weaker information about the problem that specific procedures can be leverage (e.g., dual bounds in a branch-and-bound scheme). Thus, our classification considers the use of exact or approximate DDs.

Works employing exact DDs are classified into four groups: (i) modeling, (ii) solution extraction, (iii) feasibility checking, and (iv) solution-space analysis. We divide each class depending on the specific approach the authors used. In particular, different discrete optimization methodologies integrate and exploit DDs in different ways. For example, a DD model can be formulated as a network flow model and integrated into any Integer Programming (IP) formulation. Conversely, a DD model can be coupled with a propagation procedure to create a global constraint for a Constraint Programming (CP) solver. Table 3.1 presents the paper classification for exact DDs and their respective sub-classes. The last column represents the main optimization paradigm used in each sub-class, where "Other" corresponds to a technology different to IP and CP, and "-" represents a general procedure that is not related to a particular optimization technique.

Similarly, papers that consider approximate DDs do so for two purposes: (i) bound computation and (ii) propagation. Several works in this area have proposed different procedures to construct relaxed DDs. Thus, we divide the literature of approximate DDs into three classes where the last considers several algorithms to construct tight DD relaxations. Table 3.2 summaries the works that focus on approximate DDs for each class and sub-class. As in Table 3.1, the last column corresponds to the primary technology used in conjunction with DDs. Note that for completeness Tables 3.1 and 3.2 include published works from this dissertation.

Table 3.1: Paper classification for exact DDs.

Sub-class	Papers	Technology
	Modeling	
Recursive Model	Hooker (2013), Hooker (2017), Nadarajah and Cire (2017).	DP
Network Flow	Behle (2007), Bergman and Cire (2016a), Latour et al. (2017), Haus et al. (2017), Bergman and Lozano (2020), Lozano et al. (2018), Bergman and Cire (2018), Serra et al. (2019), Hosseininasab and van Hoeve (2019), Latour et al. (2019), Cire et al. (2019), Ploskas et al. (2019).	LP/IP
Global Constraints	Cheng and Yap (2008), Cheng and Yap (2010), Perez and Régin (2014), Amilhastre et al. (2014), Perez and Régin (2015b), Perez and Régin (2015a), Perez and Régin (2016), Roy et al. (2016), Perez and Régin (2017c), Perez and Régin (2018), Verhaeghe et al. (2018), Verhaeghe et al. (2019), de Uña et al. (2019).	CP
	Solution Extraction	
General	Hadzic et al. (2004).	Other
Column Generation	Morrison et al. (2016), Kowalczyk and Leus (2018), Raghunathan et al. (2018).	LP/IP
	Feasibility Checking	
General	Nishino et al. (2015), Xue and van Hoeve (2019).	Other
Cutting Planes	Becker et al. (2005), Behle (2007), Tjandraatmadja and van Hoeve (2019), Davarnia and van Hoeve (2020), Castro et al. (2020b).	LP/IP
Benders Decomposition	Bergman and Lozano (2020), Lozano and Smith (2018), Guo et al. (2019).	LP/IP
Inference	Subbarayan (2008), Hadzic et al. (2009), Gange et al. (2011), Gange et al. (2013), Kell et al. (2015).	CP
	Solution-Space Analysis	
Post-Optimiality Analysis	Hadzic and Hooker (2006), Hadzic and Hooker (2007), Serra and Hooker (2019).	-
Solution Enumeration	Löbbing and Wegener (1996), Bergman and Cire (2016b), Bergman et al. (2018), Haus and Michini (2017).	-
Polyhedral Analysis	Behle and Eisenbrand (2007), Tjandraatmadja and van Hoeve (2019).	LP/IP

Outline. This literature review is organized as follows. Section 3.1 introduces the origins of DDs and some of the most important papers of the field. This section also presents several DD variants that are relevant to discrete optimization. Section 3.2 presents the four uses of exact DDs and shows how to integrate them into IP and CP methodologies. Similarly, Section 3.3 reviews the two primary usages of approximate DDs and several works presenting procedures to construct DD relaxations. Lastly, Section 3.4 presents the conclusions of this chapter and introduces general research directions.

3.1 Early Works on Decision Diagrams

Binary Decision Diagrams (BDDs) were the first type of DDs introduced in the literature to represent and analyze digital functions (Akers, 1978). This graphical representation is based on the work of Lee (1959) who introduce a binary-decision program representation for switching circuits.

After their introduction to the circuit design community, BDDs gain popularity as a graphical representation of Boolean functions (Bryant, 1986). These BDDs are closely related to our definition in Chapter 2 since they are layered graphs where each layer is associated with a single variable and emanating edges represent variable assignments, i.e., Ordered Binary Decision Diagrams (OBDDs). This seminal work also presents several procedures to manipulate BDDs, including the reduction algorithm described in Chapter 2.

Due to their flexible representation of Boolean formulas, BDDs have been used in many applications, including verification, finite-state system analysis, and probabilistic inference, to name a few (Bryant, 1992; Wegener, 2000). Moreover, several researchers have extended this graphical structure to represent

Table 3.2: Paper classification for approximate DDs.

Sub-class	Papers	Technology
	Bound Computation	
Dual Bounds	Andersen et al. (2007), Cire and van Hoeve (2013), Kinable et al. (2017), Castro et al. (2020a).	CP
	Kell and Van Hoeve (2013), Bergman et al. (2014a), Hooker (2017), van den Bogaerdt and de Weerdt (2018), Maschler and Raidl (2018), Castro et al. (2019), Castro et al. (2020c), van Hoeve (2020).	-
Lagrangian Bounds	Bergman et al. (2015a), Bergman et al. (2015b), Hooker (2019), Castro et al. (2020a).	LP/IP
Primal Bounds	Kell and Van Hoeve (2013), Bergman et al. (2014d), ONeil and Hoffman (2019), Horn and Raidl (2019).	Other
Branch-and-Bound	Bergman et al. (2014b), Bergman et al. (2016b), González et al. (2020).	-
	Propagation	
	Andersen et al. (2007), Hadzic et al. (2008b), Hoda et al. (2010), Cire and van Hoeve (2012), Cire and van Hoeve (2013), Bergman et al. (2014c), Perez and Régin (2017b), Perez and Régin (2017a), Kinable et al. (2017).	CP
	Constructing Relaxed DDs	
Methodology	Andersen et al. (2007), Hadzic et al. (2008a), Ciré and Hooker (2014), Bergman and Cire (2016c), Bergman and Cire (2017), Römer et al. (2018), Horn et al. (2018).	-
Merge/Split	Bergman et al. (2011), Frohner and Raidl (2019), Frohner and Raidl (2019).	-
Variable Ordering	Behle (2008), Bergman et al. (2011), Cappart et al. (2019).	-

more complex functions. One of the most relevant extensions to this literature review are Multi-valued Decision Diagrams (MDDs) (Srinivasan et al., 1990), i.e., DDs that allow arcs to represent discrete values. Another important variant is Zero-Suppressed Decision Diagrams (ZDDs), which represent Boolean expressions in a smaller graph by removing redundant edges and nodes (Minato, 1993). Lastly, Algebraic Decision Diagrams (ADDs) and Affine ADDs (AADDs) (Bahar et al., 1997; Sanner and McAllester, 2005) extend DDs to compactly represent complex algebraic functions.

3.2 Exact Representation of Solutions

Exact DDs encode the set of solutions of a discrete optimization problem as paths over a directed acyclic graph. This graphical representation is quite flexible and can be applied to a wide range of procedures. As shown in Table 3.1, we distinguish four different purposes in constructing an exact DD: modeling, solution extraction, feasibility checking, and solution-space analysis. In the following, we review each of these purposes and show how to integrate them into IP and CP solvers.

This section focuses on works where one or multiple exact DDs represent either the complete problem or a subset of its constraints. We first present different modeling techniques based on DDs. We then review works that use DDs to extract solutions or to check feasibility. The last subsection describes several enumeration procedures and post-optimality analysis algorithms over DDs.

3.2.1 Modeling

One of the most common purposes of DDs is to model complex combinatorial structures. A DD can represent any combinatorial problem since it enumerates the feasible solutions as paths. This characteristic is particularly appealing for problems that consider constraints that are usually hard to represent with standard methodologies, e.g., non-linear inequalities. This section presents different algorithms to

encode a combinatorial problem into a DD. We then describe modeling techniques that integrate DDs into IP and CP methodologies.

Recursive Models

A simple procedure to represent a combinatorial problem as a DD is to enumerates all the solutions in a tree and then merges nodes with equivalent solutions. This is a naive alternative that is impractical in many applications due to the exponential growth of the solution set with respect to the number of variables. Thus, most papers in the literature create DDs using algorithms that avoid explicitly enumerating all the solutions first.

A common approach to create DDs is to first reformulate the combinatorial problem as a recursive model (see Section 2.1 for additional details). Hooker (2013) studies the relationship between DDs and the state-transition graph of a recursive model and shows that DDs can be seen as a compact representation of the state-transition graph. Thus, we can create a DD by building the state-transition graph and merging nodes representing equivalent solutions. However, this algorithm might be computationally intractable depending on the size of the state-transition graph.

Bergman et al. (2016b) revisited this idea and presented a general procedure to create DDs based on the top-down algorithms to build relaxed DDs (see Section 2.2 for further details). The authors present recursive models for several classic combinatorial problems (e.g., the maximum independent set) and show that their procedure can efficiently create exact and relaxed DDs based on such recursive models. Hooker (2017) analyzes the relationship between DDs and recursive models for sequencing problems. The author formalizes some of the ideas introduce by Bergman et al. (2016b) to create valid DD relaxations and presents a general framework to define recursive models that are suited for exact and relaxed DDs. We refer the reader to Chapter 2 for a detailed explanations on how to construct DDs based on recursive models based on the works of Bergman et al. (2016b) and Hooker (2017).

Most of the papers in this review use a recursive model to create a DD for two reasons. First, several discrete problems have a natural recursive formulation, e.g., the knapsack problem and several sequencing problems. Second, there exists a wide range of algorithms in the literature to construct a DD from a general recursive formulation. Thus, this is the most systematic procedure to build a DD. However, there are specific cases where other mechanisms are more suited to create a DD encoding, e.g., global constraints that represent a list of feasible solutions (Cheng and Yap, 2008).

Network Flow Formulation

One of the most appealing characteristics of DDs for the mathematical programming community is their network flow reformulation (Behle, 2007). This formulation can be integrated into any IP model by adding new variables and constraints. Moreover, the network flow model is a core component for advanced procedures that combine DDs and IP methodologies, e.g., cutting planes, Benders decomposition, and Lagrangian relaxations. We now review notable applications of this reformulation as a modeling tool and briefly describe an extension for probabilistic problems.

Given a DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, the network flow model $\text{NF}(\mathcal{D})$ uses a set of variables $\boldsymbol{y} \in \mathbb{R}_+^{|\mathcal{A}|}$ to represent the flow traversing each DD arc and the original variables $\boldsymbol{x} \in \mathbb{R}^n$ to limit the flow in each layer. Equalities (3.1a) and (3.1b) are balance-of-flow constraints over $\mathcal{D}$. Constraint (3.1c) links the arcs of $\mathcal{D}$

with solutions $\boldsymbol{x}$.

$$\begin{aligned}\mathrm{NF}(\mathcal{D}) = \{(\boldsymbol{x};\boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}_+^{|\mathcal{A}|} :& \\ \sum_{a \in \mathcal{A}^{\mathrm{out}}(u)} y_a - \sum_{a \in \mathcal{A}^{\mathrm{in}}(u)} y_a = 0, & \qquad \forall u \in \mathcal{N} \setminus \{\mathbf{r}, \mathbf{t}\}, && \text{(3.1a)} \\ \sum_{a \in \mathcal{A}^{\mathrm{out}}(\mathbf{r})} y_a = \sum_{a \in \mathcal{A}^{\mathrm{in}}(\mathbf{t})} y_a = 1, & && \text{(3.1b)} \\ \sum_{a \in \mathcal{A}: s(a) \in \mathcal{N}_i} v_a y_a = x_i, & \qquad \forall i \in I \Big\}. && \text{(3.1c)}\end{aligned}$$

Intuitively, the flow over each path $p \in \mathcal{P}$ can be seen as the weight of its corresponding point $\boldsymbol{x}^p$. By restricting the total flow to have value one, the flow variables represent a convex combination of the points in $\mathrm{Sol}(\mathcal{D})$. In particular, the polytope $\mathrm{NF}(\mathcal{D})$ projected over the $\boldsymbol{x}$ variables is equivalent to the convex hull of all solutions represented by $\mathcal{D}$, i.e., $\mathrm{Proj}_{\mathrm{x}}(\mathrm{NF}(\mathcal{D})) = \mathrm{conv}(\mathcal{X})$ (Behle, 2007; Tjandraatmadja and van Hoeve, 2019). Thus, the network flow model $\mathrm{NF}(\mathcal{D})$ is an ideal linear formulation of the solution set of $\mathcal{D}$. This property is appealing for the mathematical programming community since it can be used to create extended linear formulations of complex combinatorial structures.

Recent works have shown the advantage of using a DD network flow formulation as a modeling mechanism in a wide variety of real-world applications. Cire et al. (2019) tackle a clinical rotation scheduling problem for medical students that need to stay in a subset of hospitals to fulfill their training. The authors create a DD-based network flow model to represent all feasible schedules and couple it with additional constraints to model service requirements, preferences, and other restrictions. Their experimental results show the superior performance of their network flow model when compare to a traditional mixed integer linear programming (MILP) formulation.

Another notable application is the design of a heat exchange circuit (Ploskas et al., 2019). The problem considers a set of tubes in a circuit that needs to be connected to maximize the heat exchange. The authors create a DD to represent all possible tube configurations and use its network flow formulation to encode the tube configurations in a MILP model. Their model is significantly smaller than an existing MILP formulation and manages to solve large instances in a reasonable time limit.

While these last two works create a single DD to represent the complex combinatorial component of the problems, several works consider multiple DD network flow models together. Bergman and Cire (2016a) first propose the idea of decomposing a problem using multiple DDs that represents specific aspects of the problem. Their procedure creates a network flow model $\mathrm{NF}(\mathcal{D})$ for each DD $\mathcal{D}$ where the $\boldsymbol{x}$ variables are common to each model, i.e., (3.1c) are linking constraints that synchronize the solutions among all DDs. Lozano et al. (2018) study the complexity of this multiple network flow model and show that it is NP-hard in the general case. The authors also propose a cutting-plane algorithm that solves a maximum flow problem over the DDs to derive cuts and solve the problem more efficiently.

Despite its complexity, the idea of using multiple DDs has been particularly successful in representing non-linear function as network flow models. Bergman and Cire (2018) employ this procedure to represent non-linear objective functions that admit a recursive formulation. Their technique assumes that the objective function corresponds to the sum of non-linear functions and considers one DD for each function. Bergman and Lozano (2020) present a similar decomposition for quadratically constrained problems. Their procedure decomposes the matrix of a quadratic constraint as the sum of multiple smaller matrices,

where a DD encodes the solution set induced by each sub-matrix. The authors then use a network flow formulation with linking constraints to linearize the quadratic constraint. Lastly, Nadarajah and Cire (2017) used multiple DDs to create an approximated linear program (ALP) in the context of Dynamic Programming. The authors create multiple DD-based network flow models to approximate the value function for each state in state-space.

Recent works also use these network flow models based on multiple DDs to solve challenging applications. Serra et al. (2019) consider a problem that assigns train trips and commuter vehicles to an uncertain set of passengers to minimize the number of commuter trips and total traveling time. The problem considers a DD for each possible destination and scenario to model the set of passengers associated with a commuter vehicle. Hosseininasab and van Hoeve (2019) use a similar strategy to tackle the multiple sequence alignment problem where the authors create a DD flow model to represent all pairwise sequence alignments and use linking constraints to synchronize the DD solutions. The overall problem is solved using a Logic-Based Benders Decomposition (LBBD), where the master problem corresponds to the DD flow models with linking constraints, and the sub-problems enforce additional constraints over the chosen alignments.

While the aforementioned works employ the network flow model of Behle (2007), other authors have presented variants for DDs that encode stochastic constraints. Latour et al. (2017, 2019) represent probabilistic constraints with DDs where the parameters follow a probability distribution that depends on the decision variables. The authors use the DD to model the probability of each constraint by encoding the decision variables and the stochastic parameters inside the DD. The DD is reformulated into a quadratic constraint model that is linearized and introduced into a MILP formulation of the problem.

Haus et al. (2017) also consider a variant of the network flow model for a class of two-stage stochastic programs. Their problem considers endogenous uncertainty, i.e., the first stage decision influence the stochastic process. The authors aggregate the possible scenarios with equal cost using multiple DDs and relate these scenarios with the first stage decisions. Similar to Latour et al. (2017), each DD computes the probability of achieving a cost value. The probability computation is a quadratic model that is linearized to obtain a MILP reformulation.

We note that most of the works described here create a DD network flow model to represent a subset of constraints that are generally hard to encode with linear inequalities. For example, several works model the sequencing aspect of the problem using a DD since other alternatives consider big-M constraints that have a loose linear relaxation (Cire et al., 2019; Ploskas et al., 2019; Serra et al., 2019; Hosseininasab and van Hoeve, 2019). Alternatively, some authors employ DDs as linearization tools for non-linear expressions (Bergman and Cire, 2018; Bergman and Lozano, 2020) and probabilistic structures (Latour et al., 2017, 2019; Haus et al., 2017). Thus, a DD network flow formulation is most beneficial when standard procedures lead to poor relaxations and the DD encoding is small enough. Section 3.2.3 reviews alternative procedures (i.e., decompositions and cutting planes) for problems where the DD is too large to include directly into an IP/LP formulation.

Global Constraints

In contrast to IP technologies, CP solvers represent combinatorial problems using global constraints, i.e., general-form constraints that encode sub-structures of the problem (Rossi et al., 2006). A global constraint has an inference procedure that retrieves information about feasible variable assignments and

a propagation mechanism to update the set feasible solutions inside the constraint. In this context, a DD that represents a set of solutions is a global constraint where the inference procedure checks the arc values to determine the current domain of the variables and the propagation procedure removes arcs to update the set of feasible solutions.

Andersen et al. (2007) introduce a general framework for exact and relaxed DDs in CP solvers. The authors encode the complete solution set (e.g., domain store) with a DD and propagate the branching decisions. Since the DD could grow exponentially, the authors approximate the solution space using a relaxed DD (see Section 3.3.2 for further details). Alternatively, we can represent sub-structures of the problem using exact DDs.

Several researchers have represented different global constraints with DDs and even pre-compile sub-problem structures into DDs to enhance propagation (de Uña et al., 2019). This line of research has inspired new DD variants that are suitable for CP technologies, such as non-deterministic DDs (Amilhastre et al., 2014). In the following, we review several works that represent global constraints using exact DDs. Section 3.3.2 discusses the alternative of encoding global constraints with relaxed DDs.

One of the main applications of DDs in the CP community is to encode global constraints that explicitly enumerate the set of solutions. Cheng and Yap (2008) first propose to model `Table` constraints (i.e., a list of feasible solutions) using an exact DD. The authors present an algorithm to convert a `Table` constraint into a DD by first representing the set of solutions in a tree and then merging identical sub-trees to obtain a reduced DD. Their DD construction procedure was later improved by Cheng and Yap (2010) and generalized to consider negative `Table` constraints (i.e., a list of infeasible solutions).

These preliminary works became the stepping stone for DD-based `Table` constraints and similar structures. Perez and Régin (2014) present a new algorithm for DD inference over `Table` constraints that shows superior run-time performance compare to existing methodologies. The same authors also introduce new algorithms to construct DDs from specialized `Table` constraints (Perez and Régin, 2015b). Moreover, the authors present improved procedures to manipulate DDs, e.g., algorithms to reduce a DD or to add/remove solutions (Perez and Régin, 2015a, 2016), and introduce a parallelization strategy for these algorithms (Perez and Régin, 2018). Lastly, Perez and Régin (2017c) study DD-based propagation for linear cost functions. Their work improves the cost-based propagator of a DD (i.e., propagation of a linear cost function) and introduces a DD propagator for soft constraints (i.e., constraints that allow infeasible solutions with a cost penalty).

Verhaeghe et al. (2018) also study how to efficiently encode `Table` constraints into DDs. The authors propose a related data structure called semi-DD (or sDD) where the middle layer is non-deterministic. Intuitively, an sDD is a DD obtained by connecting the leaf nodes of two trees representing partial solutions for half of the variables. The main advantage of this structure is that the maximum number of nodes in each layer can be exponentially smaller than a standard DD. The authors introduce several mechanisms to construct and manipulate an sDD, including a reduction procedure and an algorithm to remove solutions. In a related work, Verhaeghe et al. (2019) propose a new DD variant to model `Smart-Table` constraints. Their basic-smart DDs (or bs-DDs) can represent unary constraints (e.g., $x_1 \geq 1$) over arcs to compactly encode sets of feasible values. Therefore, bs-DDs avoid having multiple arcs with the same source and target node.

So far, we have reviewed DD encodings for existing constraints. However, there are some works that use exact and relaxed DDs to build new global constrains. Roy et al. (2016) introduce a new global constraint based on DDs that models binary relations over sequences of temporal events. The

authors empirically show that their DD-based global constraint was superior to a classical scheduling representation of the same temporal relations. We refer the reader to Section 3.3.2 for further examples of relaxed DD encodings of global constraints.

One of the main advantages of the DD representation of a global constraint is that it achieves generalized arc consistency (GAC). We say that a variable x is GAC for a constraint $\mathcal{C}$ if for every value of its domain there exists a feasible solution with respect to $\mathcal{C}$. Since the DD represents all feasible solutions of a global constraint $\mathcal{C}$, we can check in polynomial time if a variable assignment is related to a feasible solution or not. Another important characteristic is that CP solvers construct a DD only once and can efficiently update the graph to eliminate infeasible assignments according to other constraints or branching decisions. We note that these two properties do not hold for relaxed DDs and, thus, it is necessary to build sophisticated procedures to efficiently integrate relaxed DD with CP technologies when exact representation are too large.

3.2.2 Solution Extraction

The second purpose for constructing exact DDs is to extract feasible solutions. Since DDs represent all the solutions of a combinatorial problem, we can easily extract solutions by choosing any path in the DD. In particular, we can use this property to extract solutions with certain structure or solutions that are optimal for a specific linear objective function.

Hadzic et al. (2004) first explore this idea for a manufacturing problem. The authors create a DD to represent all product configurations and propose a polynomial-time algorithm to extract solutions that satisfy a set of configuration requirements specified by a user. Their algorithm ignores all arcs that represent infeasible configurations and returns a sub-diagram without the ignored arcs. While the authors tested their procedure in a manufacturing problem, the same idea can be applied for any other combinatorial problem to extract solutions with user-specified variable assignments.

Column Generation

Solution extraction has been mostly used as a sub-routine of the column generation procedures in IP/LP technologies. Morrison et al. (2016) propose a general scheme that applies DDs in a branch-and-price algorithm (Barnhart et al., 1998). The DD solves the pricing problem of the column generation sub-routine, i.e., the DD returns a promising solution and adds a new column to the master problem. The authors separate the last queried solution from the DD to avoid duplicated solutions. Their implementation also considers a branching rule over the original variables, which can be easily integrated into the DD by ignoring arcs with specific values.

There are three advantages of the Morrison et al. (2016) procedure to solve the pricing problem in comparison to other methodologies. First, the DD can solve the pricing problem in polynomial time (with respect to its size) for any linear objective function. Thus, their implementation constructs the DD only once and can solve the pricing problem multiple times. Second, the DD can return all optimal solutions for the pricing problem and, thus, add multiple columns in each iteration. Third, we can easily include branching decisions over the DD without changing the complexity of the pricing problem. Therefore, this procedure can be a better alternative to other methodologies when there is a compact DD representation of the pricing problem.

Morrison et al. (2016) tested their procedure in instances of the graph coloring problem, but their

technique can be applied to other problems. Kowalczyk and Leus (2018) implement the branch-and-price procedure of Morrison et al. (2016) for a parallel machine scheduling problems. Similarly to the previous work, a DD solves the pricing problem by representing all possible job assignments to machines. Raghunathan et al. (2018) also apply a similar DD-based branch-and-price routine to solve a transportation scheduling problem. Their application has multiple pricing problems, each one modeling the tours of an origin-destination pair using a DD.

Solution extraction has the advantage that we can decompose the problem into two parts. The first part, encoded with a DD, represents a relaxation of the original problem to create candidate solutions. The second part of the problem checks the feasibility of the extracted solution and gives feedback to the DD to extract new solutions. Notice that this general mechanism is a generalization of the column generation procedure and, thus, it can be applied with other technologies. We also point out that this decomposition scheme is suited for problems where the solution extraction and feasibility checking are relatively easy to solve and a DD compactly represents the set of solutions.

3.2.3 Feasibility Checking

Conversely to solution extraction, we can use an external procedure to generate candidate solutions and check feasibility with a DD. This general idea attracted the attention of researchers in the IP and CP community since there are several feasibility checking procedures for these technologies. The main advantage of using exact DDs for feasibility checking is that we construct the DD only once and repeatedly call it for each new candidate solutions.

Besides the specialized IP and CP algorithms for feasibility checking, two works apply this idea with other technologies. In both applications, the solution generation procedure solves a relatively easy problem to generate solutions fast, and the DD encodes the set of constraints that are hard to represent. Nishino et al. (2015) use a DD for feasibility checking to solve the constrained shortest path problem. Their algorithm solves the shortest path over a graph using Dijkstra's algorithm (Dijkstra et al., 1959) and represents the additional constraints in a DD. The authors check the feasibility of the partial shortest path at each step of the Dijkstra algorithm and discarded it if the DD finds that it is infeasible.

The second work creates candidate solutions using machine learning (Xue and van Hoeve, 2019). The authors embed a DD into a generative adversarial neural network (GAN) where the GAN generates solutions for a traveling salesman problem with preferences. Notice that there is no straight forward way to introduce tour preference in the optimization problem, so generating solutions with a GAN that mimics drivers' preferences is a suitable alternative. Since the GAN can generate infeasible tours, the authors encode the constraints of the problem in a DD to identify infeasible candidate solutions and, thus, guarantee feasibility.

The following subsections review feasibility checking algorithms in the IP and CP community where DDs are suitable alternatives to other methodologies. As with the procedures described above, the DD can compactly encode the set of hard constraints for the problem and check feasibility in polynomial time with respect to its size. Thus, these techniques are suited for combinatorial structures that lead to small DDs or where alternative procedures (e.g., an IP model) take exponential time to check feasibility.

Cutting Planes

Cut generation is a mathematical programming procedure to separates infeasible points from the solution set by generating inequalities that are valid for the problem but violated by the infeasible points. This separation problem is NP-hard in the general case, so most cutting plane procedures focus on specific problem structure or use a relaxation of the original problem to generate valid cuts (Cornuéjols, 2008). In this context, DDs are an attractive alternative to solve the separation problem since they provide compact representations of several combinatorial problems, and their network flow formulation enables the construction of cut generation linear programs (CGLP).

Becker et al. (2005) first proposed the idea of using DDs to solve the separation problem for ILP models in a branch-and-cut framework. Their technique considers an exact DD that represents a subset of the constraints and a sub-gradient procedure to solve the DD-based separation problem. The sub-gradient algorithm optimizes the DD considering the coefficients of a candidate linear inequality as the objective function. The optimal solution generates a gradient that updates the candidate inequality, and the procedure continues until the inequality separates the infeasible point. Davarnia and van Hoeve (2020) use a similar cutting-plane procedure to create outer-approximations for non-linear constraints. Their implementation creates a DD for each non-linear separable inequality and uses a sub-gradient routine with a normalization step to avoid unbounded solutions of the separation problem. The authors integrate their cutting plane approach in a framework that generates outer-approximations for convex and non-convex inequalities.

Behle (2007) also build on the separation problem of Becker et al. (2005) and propose a CGLP based on the DD network flow model instead of the sub-gradient routine to generate valid inequalities. The authors also propose two logical constraints inferred from a DD during the branch-and-cut search. The first one returns an exclusion cut when the DD is infeasible given the current branching decisions. Conversely, the DD returns an implication cut when the values of some variables are fixed after updating the DD with the branching decisions.

Tjandraatmadja and van Hoeve (2019) introduce an alternative method to create valid inequalities based on DDs. Their algorithm generates target cuts based on the polar set of an ILP model. Intuitively, the procedure returns a valid inequality that is orthogonal to the line between the infeasible point and an interior point in the solution set. To find such inequality, the authors propose a CGLP based on the network flow formulation of a DD. The authors show how to enhance their cutting plane procedure to generate facet-defining inequalities, i.e., valid inequalities that define the convex-hull of the solution set.

Cutting-plane procedures are usually coupled with constraint enhancement routines such that the resulting cut removes a larger portion of the fractional space. However, there is currently a limited number of DD-based algorithms to enhance valid inequalities. Becker et al. (2005) introduce a simple procedure to change the coefficients of a valid inequality to increase its face dimension. However, their technique does not guarantee that the resulting inequality would have a higher dimension nor that it will separate the set of points that the original constraint separates. Behle (2007) propose a DD-based lifting procedure for cover cut inequalities based on classic techniques for these constraints (Wolsey and Nemhauser, 1999). Their algorithm only differs from the original techniques in that the DD is used to solve the new coefficient sub-problem instead of using an integer or linear program.

As with other DD procedures reviewed so far, these cutting plane techniques leverage the fact that the DD is built only once and can be used to find optimal solutions with different objective functions. For example, the sub-gradient routines use this property to create new sub-gradients based on the DD

optimal solutions. The same is true for the CGLP models and the inequality enhancement procedures. We note that these cutting plane techniques are also valid if we replace the exact DD with a relaxed DD (Tjandraatmadja and van Hoeve, 2019). In this case, the generated inequalities are valid for the original problem but the cutting plane procedure will not separate all infeasible solutions.

Benders Decomposition

Benders decomposition is another IP methodology that has benefited from DDs (Benders, 1962). As in the branch-and-price applications, each sub-problem corresponds to a single DD. However, the sub-problems return valid inequalities that remove the candidate solution proposed by the master problem. Bergman and Lozano (2020) first study this approach for quadratically constrained integer problems. Their procedure decomposes the quadratic matrix into multiple smaller matrices where a DD can easily represent the solution set of each component matrix. Since encoding the decomposition as a network flow model is computationally inefficient, the authors propose a Benders decomposition scheme where each sub-problem solves the shortest path problem over its corresponding DD to generate a Benders cut.

Similar ideas have been also used to solve stochastic programming problems. Lozano and Smith (2018) introduce a DD-based Benders decomposition for a family of two-stage binary stochastic programming problems. The authors use a DD to represent the sub-problem and solve a max-flow problem over the DD to create a valid cut. They test their procedure over a two-stage traveling salesman problem where the first stage decisions determine the set of locations to visit, and the second stage decisions represent the chosen tour for all scenarios. Guo et al. (2019) apply this same procedure to tackle a stochastic distributed operation room scheduling problem.

In all these applications the DD represents a combinatorial problem that might take a considerable time for an IP model to solve. Since the sub-problems of these Benders decompositions are relatively small (e.g., 20 to 40 cities for the traveling salesman problem), the resulting DD is small enough to store in memory and can solve the sub-problems in fractions of a second. Moreover, the set of solutions in the DD does not change, so the overhead of constructing the DD is negligible if we consider that we need to solve the sub-problem thousands of times.

Inference

The CP community has also considered DDs for feasibility checking to infer no-good assignments. A no-good is an infeasible set of variable assignments (Schiex and Verfaillie, 1994). CP solvers employ no-goods to avoid exploring portions of the solution space that it has already shown to be infeasible.

Subbarayan (2008) first proposed the idea of using DDs to infer no-good assignments in CP solvers. The author shows that the problem of finding a minimum no-good over a DD is NP-hard and proposes a heuristic procedure that traverses the DD to find small sets of no-good assignments. Gange et al. (2011) revisit this idea and present an incremental algorithm to find no-goods that searches just a portion of the DD in the vicinity of the last set of arcs removed from a DD. Gange et al. (2013) extend this no-good algorithm for cost-based reasoning, i.e., to identify assignments that lead to sub-optimal solutions. A similar idea was implemented for SAT solvers (Kell et al., 2015) where a DD represents a subset of the constraints (i.e., clauses) and identifies no-goods for clause generation.

We note that no-good inference is related to the cutting plane procedures in IP technologies. A no-good set can be encoded as a linear constraint and added into an IP solver in a branch-and-cut procedure. In fact, the DD-based logic constraints introduce by Behle (2007) are a particular type of

no-goods that the DD infers during the search. Thus, these advanced technologies for no-good inference in the CP literature can also benefit IP solvers.

3.2.4 Solution-Space Analysis

DDs provide a compact representation of the solution space, which is also useful if we want to enumerate all the solutions and analyze them. For example, we can analyze the set of (near) optimal solutions or study the convex hull of the solutions of a DD. We review different procedures to analyze the solutions encoded in a DD and algorithms to enumerate solutions with specific characteristics.

Post Optimality Analysis

Post-optimality analysis is one of the earliest applications of DDs in the discrete optimization community (Hadzic and Hooker, 2006). The authors introduce three procedures for post-optimality analysis over an exact DD. The first one is a cost-based domain analysis that identifies the set of variable assignments that are present in at least one near-optimal solution. The second procedure is conditional cost-based domain analysis, which restricts a subset of variables to take specific values and then performs cost-based domain analysis over the subset of solutions. Lastly, their frequency analysis computes the percentage of solutions with a particular variable assignment. Of all these procedures, cost-based domain analysis is the only one that has been further studied in the literature.

Since representing the set of solutions using a DD is intractable for most combinatorial problems, Hadzic and Hooker (2007) introduce the concept of sound DDs for post-optimality analysis. A DD is sound for a specific post-optimality analysis if it yields the same results that an exact DD would. Thus, a sound DD might represent a larger solution set than an exact DD but preserves certain properties to correctly perform the analysis. The authors present a pruning and contraction procedure to create sound DDs for cost-based domain analysis. While the resulting sound DD is significantly smaller than an exact DD, their procedure requires a starting DD that is either exact or represents all feasible solutions within a cost range. Thus, creating sound DDs with this procedure is intractable for large problems.

Recently, Serra and Hooker (2019) revisited the idea of sound DDs for post-optimality analysis and provided new insights. The authors present several theoretical results related to sound DDs, including a sound-reduction algorithm that constructs the smallest sound DD. They also introduce a general construction procedure for ILP models that creates a sound DD from a branching tree. Their experiments illustrate the advantages of their technique for a wide range of ILP benchmark instances, where the resulting sound DD is a more suitable alternative to post-optimality analysis than, for example, branch-and-bound enumeration.

Solution Enumeration

Recent works have also employed DDs to represent the set of non-dominated solutions for a multi-objective discrete optimization problem, i.e., the Pareto frontier. Bergman and Cire (2016b) first tackle the problem by representing the set of feasible solutions with an exact DD and then enumerating the non-dominated solutions using a multi-criteria shortest path algorithm over the DD. This work was extended by Bergman et al. (2018) where the authors present three procedures that modify the DD while preserving the set of non-dominated solutions. The authors also propose a bidirectional multi-criteria shortest path procedure to enumerate the non-dominated solutions, which outperforms the unidirectional

approach (Bergman and Cire, 2016b). Moreover, the DD-based methodology outperforms state-of-the-art algorithms in three different problem sets with three or more objective functions.

So far, all the papers discussed in this section enumerate solutions to analyze optimal or near-optimal solutions. Conversely, the following two works employ the DD structure to obtain insightful information on the combinatorial set and, thus, ignore the objective function. Löbbing and Wegener (1996) create an exact DD to count the number of knight's tours in a chessboard, which at the time was computationally impossible with standard enumeration techniques. Recent work by Haus and Michini (2017) construct a DD to encode all the members of an independent set. The authors analyze the size of the resulting DD and show that the DD representation has polynomial size in the number of variables for packing and set covering problems with specific characteristics. Thus, this work opens the possibility of solution set analysis for problems with tractable DD size.

Polyhedral Analysis

We now review two works that employ DDs for polyhedral analysis, an important area of research in mathematical programming due to its relevance for solving IP. The first paper introduces a procedure to enumerate vertices and facets of the convex hull of a DD solution set (Behle and Eisenbrand, 2007), i.e., $\text{conv}(\text{Sol}(\mathcal{D}))$. Their technique considers a binary solution set in an exact DD, where each path in the DD corresponds to a 0/1 vertex of the convex hull polytope. The procedure to enumerate facets starts with an initial facet that is rotated over the DD to obtain a new facet.

Tjandraatmadja and van Hoeve (2019) also present a polyhedral analysis procedure based on DDs. The authors introduce a mechanism to certify the dimension of any inequality that is a face of $\text{conv}(\text{Sol}(\mathcal{D}))$. Their procedure finds a set of affine independent points in the DD that are tight for the face. To do so, the authors solve a flow problem over the DD and use a heuristic procedure to generate the affine independent points based on the flow values of each arc in the DD. Then, the number of affine independent points gives a lower bound on the dimension of the face.

We note that polyhedron analysis is intractable for general IP models since constructing the convex hull of all the solutions is NP-hard (Wolsey and Nemhauser, 1999). The DD provides a more manageable procedure to enumerate all the solutions and, thus, gives valuable insights to the polyhedral structure of challenging problems that have a compact DD encoding.

3.3 DDs for Approximating the Set of Solutions

The size of a DD can grow exponentially with the number of variables. Thus, constructing an exact DD might be impractical for many applications if the number of decision variables is too large. To overcome this limitation, Andersen et al. (2007) propose a limited size DD that over-approximates the solution set, i.e., a relaxed DD. A relaxed DD is a discrete relaxation of the problem and, as such, optimizing over a relaxed DD provides a valid dual bound for the original problem (see Section 2.2 for more details).

Alternatively, Bergman et al. (2014d) introduce restricted DDs, i.e., a limited size DD that represents a subset of the solution space. Restricted DDs can be build using a top-down procedure that eliminates nodes when the maximum width limit is exceeded. Thus, the solution set of a restricted DD $\mathcal{D}$ is a subset of the feasible region $\mathcal{X}$, i.e., $\text{Sol}(\mathcal{D}) \subseteq \mathcal{X}$. In contrast to relaxed DDs, optimizing over a restricted DD provides a primal bound since all paths in the restricted DD represent feasible solutions.

Approximate DDs (i.e., relaxed and restricted DDs) caught the attention of many researchers since they can provide strong bounds with limited size. We distinguish three main research areas related to approximate DDs (see Table 3.2). The first research area focuses on bound computation and specialized search procedures that employ these bounds. The second area studies propagation procedures based on relaxed DDs for CP solvers. The last area focuses on constructing relaxed DDs to obtain tight dual bounds.

3.3.1 Bound Computation

This section reviews several bounding mechanisms based on approximate DDs. We first focus on dual bounds provided by relaxed DDs and explain how to improve these bounds using Lagrangian duality. We then review works that compute primal bounds using relaxed and restricted DDs. The section ends with a description of DD-based branch-and-bound procedures and suggestions on how to efficiently introduce DD bounds in a standard branch-and-bound search.

Dual Bounds

Andersen et al. (2007) first introduce the idea of using relaxed DDs to obtain dual bounds for an optimization problem. A relaxed DD over-approximates the set of feasible solutions, so optimizing over it will return a valid dual bound for the problem. We refer the reader to Section 2.2.4 for a description of how to obtain DD dual bounds for different objective functions.

Researchers have tested these bounds in a wide variety of combinatorial problems, including set covering (Bergman et al., 2011), multidimensional bin packing (Kell and Van Hoeve, 2013), maximum independent set (Bergman et al., 2014a, 2016b), maximum cut, maximum 2-satisfiability (Bergman et al., 2016b), and graph coloring (van Hoeve, 2020). All these applications create a single relaxed DD to approximate the set of feasible solutions and compute bounds using the shortest path procedure over the DD or its network flow model. These papers show competitive bounds for small and medium size instances and small-width DDs. However, larger relaxed DDs are needed to create tight bounds of bigger instances.

Besides these combinatorial problems, one of the most popular applications of approximate DDs is for sequencing problems. These problems are generally formulated recursively and come with a natural variable order (i.e., choose elements in sequential order), which facilitates the construction of relaxed and restricted DDs. Moreover, several works have shown the effectiveness of DDs in classical sequencing problems, e.g., the traveling salesman and job sequencing (Cire and van Hoeve, 2013; Hooker, 2017, 2019), which have motivated extensions to more complex variants.

Cire and van Hoeve (2013) first apply relaxed DDs to solve sequencing problems. The authors present a general framework to create relaxed DDs and compute dual bounds. Their procedure is integrated into a branch-and-bound scheme that updates the DD during search to compute more accurate bounds. The authors test their DD dual bounds with satisfactory results in several sequencing problems, including the traveling salesman problem (TSP) with time windows, TSP with precedence constraints, and sequencing problems with makespan and total tardiness minimization. Similarly, Kinable et al. (2017) tackle several TSP variants with sequence-dependent cost using a DD relaxation embedded in a CP solver. The authors employ an additive bound procedure to strengthen the DD relaxation with an LP relaxation. Their technique outperforms previous methodologies in the literature and introduces a general approach to

enhance DD relaxations.

Several researchers study the quality of the DD bounds for sequencing problems and compare them to the LP relaxation. Hooker (2017) formalizes some of the main components to create relaxed DDs for general sequencing problems and presents preliminary results on the bound quality for different DD construction procedures. van den Bogaerdt and de Weerdt (2018) use this framework to create bounds for the multi-machine scheduling problems with encouraging performance. Similarly, Maschler and Raidl (2018) study the DD bound quality for a prize-collecting sequencing problem and compare the bounds given by a top-down and an iterative refinement construction scheme.

One of the main reasons for these high-quality DD bounds is that we can always modify the DD to obtain better bounds. As discussed in Section 3.3.3, there is a large body of literature on how to create tight DD relaxations that will lead to stronger dual bounds. An alternative is to increase the width of a relaxed DD to obtain better dual bounds. This flexibility makes DDs quite appealing since we can modify the relaxed DD to perform well for the application at hand.

The simplest alternative to improve the DD dual bounds is to increase the width limit to obtain a tighter relaxation. However, there are two considerations to keep in mind when increasing the DD width. First, larger relaxed DDs require more computational resources, i.e., memory and time to construct the DD and calculate the dual bounds. Second, empirical results in several papers show diminishing returns in terms of bound improvements, i.e., the bound improvement decreases as the DD width increases (Bergman et al., 2014a, 2016a). Thus, a user needs to find a DD width that will lead to computationally efficient relaxed DDs that provide informative dual bounds.

Lagrangian Bounds

An alternative for computing better dual bounds is to enhance the DD relaxation with dual information from an ILP formulation. To explain this idea consider a minimization problem with feasible set $\mathcal{X} \subseteq \mathbb{Z}^n$, a linear objective function $\boldsymbol{c}^\top \boldsymbol{x}$, and a relaxed DD $\mathcal{D}$ that over-approximates $\mathcal{X}$, i.e., $\mathcal{X} \subseteq \text{Sol}(\mathcal{D})$. The idea is to incorporate a set of m valid inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$ into the original problem as penalties to the objective function to avoid paths in $\mathcal{D}$ that are infeasible. The resulting problem is a Lagrangian sub-problem

$$\mathcal{L}(\boldsymbol{\lambda}) = \min\{\boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\lambda}(A\boldsymbol{x} - \boldsymbol{b}) : \ \boldsymbol{x} \in \text{Sol}(\mathcal{D})\},$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m_+$ are the Lagrangian penalties.

Since $\text{Sol}(\mathcal{D})$ can be reformulated as a network flow model $\text{NF}(\mathcal{D})$, all the theory of Lagrangian duality holds for $\mathcal{L}(\boldsymbol{\lambda})$ (Conforti et al., 2014; Fisher, 2004). In particular, $\mathcal{L}(\boldsymbol{\lambda})$ with $\boldsymbol{\lambda} \in \mathbb{R}^m_+$ is a valid dual bound for the original problem. Then, the Lagrangian dual problem seeks for $\boldsymbol{\lambda}$ that gives the tightest dual bounds. In our minimization example, the Lagrangian dual maximizes the sub-problem $\mathcal{L}(\boldsymbol{\lambda})$ and, thus, returns a bound that is equal to or stronger than the original problem:

$$\min\{\boldsymbol{c}^\top \boldsymbol{x} : \ \boldsymbol{x} \in \text{Sol}(\mathcal{D})\} \leq \max\{\mathcal{L}(\boldsymbol{\lambda}) : \ \boldsymbol{\lambda} \in \mathbb{R}^m\}.$$

Bergman et al. (2015b) first propose Lagrangian duality as a mechanism to enhanced DD relaxations. Their approach considers a DD that represents a subset of the constraints of a problem. The Lagrangian procedure introduces dual information for the remaining constraints as penalties in the objective function. The authors test their Lagrangian relaxation procedure over the TSP problem with encouraging results, where the DD-based Lagrangian relaxation returns significantly tighter bounds than the pure

DD relaxation. Hooker (2019) explores this idea for a family of sequencing problems and presents a detailed experimental evaluation. The empirical results show that the DD-based Lagrangian relaxation provides tight bounds for most instances and proves optimality of heuristic solutions in several open problems.

Bergman et al. (2015a) introduce an alternative procedure to employ Lagrangian penalties with DDs. Their technique uses Lagrangian decomposition to communicate information over multiple DDs that represent different constraints of an optimization problem. The Lagrangian decomposition scheme adds penalties to each DD to synchronize their solutions. The authors propose this technique in the CP literature to improve propagation across multiple DDs. Lagrangian decomposition can obtain tighter bounds than Lagrangian relaxation (Guignard and Kim, 1987) and, thus, can be an effective alternative for bound computation.

We note that despite the simplicity of this Lagrangian dual bounds, the works employing this procedure are very limited. We believe that this idea is a promising line of research. An advantage of Lagrangian procedures is that they avoid constructing huge relaxed DDs that encode the complete problem to create tight bounds. Alternatively, we can use an exact or relaxed DD to represent the set of constraints that are hard to encode using linear inequalities and introduce the remaining constraints as dual penalties.

Primal Bounds

In contrast to relaxed DDs, restricted DDs compute primal bounds and extract feasible solutions for a problem. Bergman et al. (2014d) first introduced this graphical structure as a general procedure to heuristically generate solutions for discrete optimization problems. Their procedure computes primal bounds by optimizing the restricted DD using the shortest path procedure for minimization problems (see Section 2.2.4). Since all the paths represent feasible solutions, an optimal path corresponds to the tightest primal bound given by the restricted DD.

The main advantage of restricted DDs is that they encode a set of feasible solutions, so they can potentially compute stronger primal bounds than other methodologies. For example, Bergman et al. (2014d) show that DD primal bounds are competitive to the ones provided by IP solvers for set covering and set packing problems. Moreover, we can introduce restricted DDs into search algorithms to obtain stronger primal bounds. For instance, ONeil and Hoffman (2019) create restricted DDs to solve a TSP problem with pickups and deliveries in an online setting. The authors use small-width restricted DDs within a branch-and-bound search to find high-quality solutions in a few seconds.

We note that relaxed DDs can also provide primal bounds. For example, Horn and Raidl (2019) used a limited discrepancy search procedure guided by relaxed DD dual bounds to find feasible solutions for a prize-collecting job sequencing problem. Alternatively, we can extract feasible solutions from a relaxed DD using some heuristic methods that traverse the DD from root to terminal node. However, the procedure might be unsuccessful if we select a partial path that leads to infeasible solutions.

Lastly, the primal bound of a restricted DD can also certify the feasibility of a problem. For instance, Kell and Van Hoeve (2013) use restricted DDs to show the feasibility of multidimensional bin packing problems. If the restricted DD has at least one path, then the problem is feasible. This simple procedure proved feasibility for several instances that IP and CP technologies could not in a given time limit.

Branch-and-Bound Procedures

One of the main advantages of relaxed DDs is that they can provide stronger dual bounds than an LP relaxation (Bergman et al., 2014a). Thus, replacing the LP relaxation with a DD relaxation in a branch-and-bound procedure can be very advantageous.

Bergman et al. (2016b) propose a general branch-and-bound scheme where relaxed and restricted DDs provide dual and primal bounds, respectively. The main difference with standard LP-based branch-and-bound is that their procedure branches over nodes of a relaxed DD instead of variable-value assignments. Thus, the DD-based branch-and-bound can potentially generate fewer sub-problems since each branching decision fixes the values of multiple variables at a time. Bergman et al. (2014b) extend this procedure for parallel computing, where every core is responsible for a DD sub-problem. Their procedure takes advantage of the flexibility of DDs to efficiently process the sub-problems and communicate bounds between each other.

González et al. (2020) propose a mechanism that integrates IP into the DD-based branch-and-bound. The authors modify the DD-based branch-and-bound of Bergman et al. (2016b) so that relaxed nodes can be either solved by an IP solver or follow the original DD branching mechanism. To identify which node should be solved by an IP solver, the authors implement a supervised learning technique that chooses nodes during search. This novel DD-based branch-and-bound procedure can be applied to any combinatorial problem and shows promising results in the maximum independent set problem.

We note that DD bounds can be integrated to other search procedures, such as a standard branch-and-bound (Cire and van Hoeve, 2013; Kinable et al., 2017). The main disadvantage is that the search algorithm might not leverage the DD structure as the specialized DD-based branch-and-bound does. To take advantage of the DD structure it is important to branch on variables following the ordering in the DD. Some authors also notice that a depth-first search strategy is preferable for DDs since the branching updates can be done more efficiently by removing arcs in the last branched layer (Cire and van Hoeve, 2013).

3.3.2 Propagation

As reviewed in Section 3.2.1, several researchers in the CP community encode global constraints using exact DDs. However, the size of an exact DD grows exponentially with the number of variables, so they become impractical for large combinatorial structures. Andersen et al. (2007) propose to represent global constraints with relaxed DD instead to avoid the exponential size of exact DDs.

While relaxed DDs are more flexible than exact DDs in terms of memory usage, relaxed DDs do not achieve generalized arc consistency (GAC) in polynomial time as in the case of exact DD. Since some paths in a relaxed DD are infeasible, checking if there exists a feasible solution for a specific variable-value assignment is not a trivial task. Andersen et al. (2007) propose a new consistency measure to evaluate the propagation capabilities of a relaxed DD. We say that a relaxed DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ achieves DD consistency for a global constraint $\mathcal{C}$ if for each arc $a \in \mathcal{A}$ there exists a path p that traverses a and is feasible with respect to $\mathcal{C}$. Intuitively, DD consistency asks for a relaxed DD with no infeasible arcs. Also, DD consistency is equivalent to GAC for width-one DDs (Andersen et al., 2007) but it is a stronger condition for larger DDs.

Researchers have proposed sophisticated propagation mechanisms for different global constraints to achieve DD consistency in polynomial time. A propagation procedure for a relaxed DD is defined by

the set of relaxed states, the procedure to update the states in each node, and the set of filtering rules (see Section 2.2). Andersen et al. (2007) proposed the first relaxed DD propagators for `Linear` and `All-different` constraints. Later on, Hadzic et al. (2008b) extended the propagator from linear inequalities to separable inequalities and showed that it achieves DD consistency in polynomial time.

Since the work of Andersen et al. (2007), several papers have introduced DD propagators for other well-known global constraints. Hoda et al. (2010) explore DD propagators for the `Among` and `Element` constraints. The authors also propose new filtering rules of the `All-different` propagator, which improve its propagation capabilities. Bergman et al. (2014c) present a DD propagator for the `Sequence` constraint and show that DD consistency is NP-hard for this constraint. More recently, Perez and Régin (2017a) create an DD encoding for the `Dispersion` constraint where the DD enforces the mean value constraint and uses a cost-based propagation for the variance restriction.

Besides the encoding of existing global constraints, relaxed DDs are also a building block to create new global constraints. Cire and van Hoeve (2012) introduce a global constraint for disjunctive scheduling based on a relaxed DD. Their DD represents the set of job sequences and considers release times, deadlines, precedence relations, and sequence-dependent set-up times.

Two recent works develop new probabilistic global constraints using DDs. Perez and Régin (2017a) introduce a Probability Mass Function (PMF) constraint where a DD encodes the linear inequality restricting the mean value of the variables and a cost-based propagator to ensure that the probability of every feasible assignment is inside a pre-defined range. The authors extend this constraint to consider probability distributions given by a Markov chain process (Perez and Régin, 2017b).

Lastly, DDs can also improve the propagation capabilities of a CP solver by sharing information. Hadzic et al. (2009) first study this idea using compatibility labels between multiple DDs. Their procedure constructs an exact or relaxed DD for each constraint of the problem following the same variable ordering. It then traverses the nodes of each DD to identify compatibility with nodes in different DDs. The authors show that nodes that do not have any compatibility label can be removed since the solutions traversing that node are infeasible in all other DDs. Bergman et al. (2015a) introduce a different procedure to communicate information between DDs based on Lagrangian decomposition, which we discussed in Section 3.3.1.

While relaxed DDs can model a wide variety of global constraints, there might be alternative procedures that are more suitable for some constraints. Specifically, Andersen et al. (2007) note that there might be a polynomial-time algorithm to enforce GAC for some constraints but it might be NP-hard for polynomial-size DDs to do so. A simple example is the `All-different` constraint that achieves GAC in polynomial time by representing the constraint as a matching problem in a bipartite graph. However, GAC is NP-hard in a relaxed DD because the GAC problem reduces to a Hamiltonian path problem.

3.3.3 Constructing DD Relaxations

While several works have shown the advantage of using DD relaxations (Bergman et al., 2016a), the question of how to construct a relaxed DD that provides tight bounds remains open. The size of the DD plays an important role in the quality of relaxation, i.e., bigger diagrams are expected to produce tighter bounds. Nonetheless, two DDs with equal size can provide significantly different bounds. Thus, researchers have studied different procedures to construct relaxed DDs of fixed size that provide strong dual bounds.

The two most common techniques to construct relaxed DDs are the top-down and the iterative

refinement procedures (see Chapter 2). The top-down procedure (Andersen et al., 2007) starts at the root node, creates one layer at a time, and merges nodes when a layer exceeds the width limit. Conversely, the iterative refinement algorithm (Hadzic et al., 2008a) starts with a width-one DD and iteratively increases the DD width by splitting nodes until the width limit is reached. While there are implementation advantages for both techniques, the question of which procedure provides better bounds depends on their corresponding merging and splitting algorithms.

The decision on how to merge or split nodes is usually made heuristically and depends on the application at hand. However, recent works have studied this problem and develop sophisticated techniques to obtain strong DD bounds for general problem structures. For instance, Bergman and Cire (2017) study the problem of finding the best DD relaxation for a given width limit. The authors present an IP model that partitions the nodes in each layer to obtain the strongest dual bound. Their experiments show that the resulting DD produces significantly stronger bounds than any other heuristic methods. However, the computational time to solve their IP model can be quite long and, thus, impractical for many applications.

On the heuristic side, there are several merging and splitting strategies for sequencing problems (Cire and van Hoeve, 2013) and other combinatorial structures (Bergman et al., 2016a). Bergman et al. (2011) propose a simple merging strategy that merges nodes with respect to their longest/shortest path from the root node. Thus, the heuristic avoids merging nodes that can potentially impact the DD bound. Frohner and Raidl (2019) improve this heuristic by including tie-breaking rules to merge nodes that are similar to each other with respect to their relaxed states. Also, Frohner and Raidl (2019) present a binary classifier procedure where the classifier chooses a merging heuristic in each DD layer. Their classifier-based heuristic outperforms the simple heuristics in terms of dual bounds but it can be harder to implement due to the additional time required to train the classifier.

A less commonly used technique to construct relaxed DDs is the separation procedure (Ciré and Hooker, 2014). This algorithm is a variant of the iterative refinement since it starts with a width-one DD and iteratively split nodes. However, the separation procedure split nodes systematically so all the paths in the DD satisfy a constraint or have a longest-path value larger than a certain threshold (Bergman et al., 2011). This technique can generate highly-accurate relaxations if, for instance, we separate the constraint that is currently violated by the longest path in a maximization problem (Bergman and Cire, 2016c). Recent work by van Hoeve (2020) shows that separation procedure are quite effective for the graph coloring problem where the DD relaxation yield bounds competitive to state-of-the-art methodologies. The author identifies conflicts (i.e., violated constraints) in an infeasible path and separates the conflicts along that path.

Besides these three construction techniques, two recent works have developed new procedures to construct relaxed DDs. Römer et al. (2018) propose a local search framework with operations that can merge and split nodes or redirect arcs. Their procedure generalizes the top-down and iterative refinement algorithms and consistently yields better bounds than the simpler alternatives. Horn et al. (2018) introduce an A^*-inspired algorithm to construct relaxed DDs. Similarly to the top-down compilation, their procedure starts at the root node but keeps a priority list of nodes to create next. Moreover, their algorithm can create and merge nodes in different layers.

Independent of the chosen technique, one of the main challenges when constructing a relaxed DD is the variable ordering, i.e., the assignment of variables to layers. The size of a DD depends on the variable ordering and the optimal ordering can lead to significantly smaller DDs. The problem of finding

the optimal variable ordering is NP-hard (Bollig and Wegener, 1996) and has been extensively studied for exact DDs. Several authors propose heuristic variable orderings for different applications, including the knapsack (Behle, 2008) and the maximum independent set problem (Bergman et al., 2011). A more general ordering procedure was recently introduced by Cappart et al. (2019) based on reinforcement learning (RL). While their technique can be applied to any optimization problem, it is only compatible with the top-down construction procedure: the RL agent chooses during construction the variable that will be assigned to the next layer. Their procedure shows encouraging performance but, as most RL-based techniques, the RL agent needs to be trained over several random instances before deployment.

There is no clear hierarchy on which construction procedure is better but there are some general guidelines on which algorithm to choose depending on the application at hand. Top-down procedures are considerably faster to implement and deploy than iterative refinement procedure, so a top-down construction is usually preferable for prototyping new ideas or when the user needs to construct multiple DDs. In contrast, iterative refinement procedures can lead to stronger dual bounds if we use a splitting heuristics that focuses on separating infeasible solutions that contribute to the bound quality. Lastly, the variable ordering can significantly affect the DD bounds, so it is important to test several ordering strategies.

3.4 Conclusions

This chapter has reviewed recent DD works that model and solve discrete optimization problems. We describe a wide range of procedures that benefit from a graphical representation of the solution set and show how these techniques can be integrated into state-of-the-art solvers. In particular, we distinguish six different ways to use DDs: modeling, solution extraction, feasibility checking, computing primal and dual bounds, inference and propagation, and solution-space analysis.

Most of these procedures are associated with a specific type of DD but there are certain exceptions. For example, we can use either exact or relaxed DDs to create global constraints and their propagation procedures. The main difference between the two structures is that exact DDs have stronger propagation guarantees and that relaxed DDs need to be updated during search to further remove infeasible assignments. Another notable example is cutting plane procedures: relaxed DDs provide valid cuts but will not separate all infeasible points unless they are updated during search.

We notice that there are two key advantages of DDs that explain the success in many of the applications discussed in this chapter. First, we can efficiently optimize the DD using different linear objective functions. This property is crucial for cutting planes and decomposition techniques since DDs are iteratively used, for example, to solve the separation problem for each new fractional point. Similarly, DDs are computationally efficient when we need to extract information from the diagram multiple times, as in the case of the no-good inference procedures.

The second key advantage of DDs is that they can be used in multiple ways to solve the same problem. For example, we can construct a relaxed DD to obtain dual bound and have a heuristic procedure that extracts feasible solutions from the DD (i.e., obtain primal bounds). Similarly, we can apply the same DD to generate valid inequalities, prune sub-optimal solutions, and infer no-good assignments, all to solve the same application. This property makes DDs a strong and flexible optimization tool that can benefit from techniques developed in different fields.

Bridging Two Worlds

While the area of DDs for optimization problems is still fairly new, there is a vast literature on how to create and manipulate DDs in the computer science (CS) and verification communities (Wegener, 2000). The operation research (OR) community adopted some of the ideas and procedures to manipulate DDs from the CS literature (e.g., the reduction algorithm), however, these two communities consider different DD variants and construction procedures due to the problem characteristics of each field. Nonetheless, there is an opportunity to improve our DD algorithms with the techniques used in other fields.

As we mention in this dissertation, we require efficient algorithms in order to create large and informative (relaxed) DDs. Thus, exploring new ways to construct DDs could benefit the field of discrete optimization. The top-down and iterative refinement procedures are the most commonly used by researchers in OR because they are better suited to create DD relaxations. However, the verification community has developed different methodologies to create exact DDs for functions and circuits.

The most well-known approach to create DDs in the verification community is the *Apply* algorithm (Bryant, 1992). This procedure relies on merging two or more DDs using binary operators (e.g., union) to avoid the exponential explosion when creating exact DDs. The procedure starts with smaller DDs that represent portions of the original function and iteratively combines them to create an exact DD that is considerably smaller than the one obtain, for example, by a top-down algorithm on the entire function. The *Apply* procedure has been extensively applied in the verification community (Wegener, 2000), but, to the best of our knowledge, has never been used to create relaxed DDs. Therefore, one alternative to develop efficient DD construction procedures would be to adapt the *Apply* algorithm to relaxed DDs.

Another important difference between the verification and OR communities is the DD variants that the former considers. As mentioned in Chapter 2, the OR community usually avoid long arcs since they aggregate multiple variable assignments, while these arcs are fairly common in verification applications (Wegener, 2000). Zero-suppressed decision diagrams (ZDDs) is another DD variant that few OR researchers have employed (Morrison et al., 2016), even though it is known to be an efficient DD variant to represent sets (Minato, 1993). Thus, it would be interesting to analyze the advantages of using ZDDs to solve combinatorial problems and possibly extend some of the techniques develop for DDs to this more compact graphical structure.

Research Directions

While several recent works have shown the benefits of solving optimization problems using DDs, there are still several avenues to explore. For example, modeling is one of the most popular applications of DDs, in particular when the DD is reformulated as a network flow model. However, the DD formulation can be too large to include in an IP model and few works have considered using cutting plane procedures or decomposition schemes to overcome this limitation. Also, most cutting-plane procedures are proposed for general structures but none have focused on sequencing problems, which is one of the most popular applications of both relaxed and exact DDs.

Another notable research direction is generalizing DD procedures in a specific optimization community. For example, the CP community has developed several sophisticated algorithms to construct, manipulate, and propagate constraints over DDs, which can be beneficial in other areas. Some of the most popular procedures to construct relaxed DDs were first proposed in the CP literature. Following

this trend, a possible research direction is to study semi-DDs. This DD variant can represent the set of solutions using exponentially fewer nodes than a regular DD and, thus, has the potential of becoming a better graphical structure than current DDs. However, the limitations of this new DD variant are still unclear and it is not known if the DD procedures described in this literature review can be also adapted to semi-DDs.

Lastly, the works in the dissertation can be classified into one or more of the categories introduced in this survey (see Tables 3.1 and 3.2). Our work on a pickup-and-delivery problem (see Chapter 4) employs DD-based Lagrangian bounds, thus, it can be classified in the Lagrangian bounds class. Conversely, the delete-free AI planning work (see Chapter 5) uses relaxed DDs to compute primal and dual bounds, so it can be considered in these two categories. This work also propose inference mechanisms to extract information of a planning task (i.e., landmarks and redundant actions), so it can be considered as part of the inference class too. The cut generation and lifting procedures introduced in Chapter 6 can be categorized under the cut generation class. Also, our second-order cone encoding and novel network flow model are valuable contributions to the modeling category.

Chapter 4

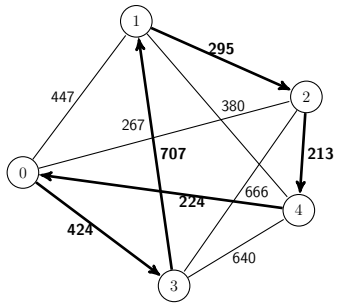
Multi-Commodity Pickup-and-Delivery

Pickup-and-delivery refers to a large class of optimization problems that is primarily concerned with the transportation of persons or commodities between locations. Such problems represent core routing decisions in a wide range of practical applications. Examples include parcel delivery (Holland et al., 2017), dial-a-ride problems (Cordeau and Laporte, 2007; Liu et al., 2018), home healthcare (Liu et al., 2013), robotics (Booth et al., 2016), and emergency dispatch (Cordeau et al., 2007), to name a few. The area is associated with a pervasive and rich literature in optimization and scheduling; see, e.g., Savelsbergh and Sol (1995); Parragh et al. (2008).

This work investigates a new exact approach for the *one-to-one multi-commodity pickup and delivery traveling salesman problem* (m-PDTSP), a variant of the classical traveling salesman problem (TSP) that incorporates the delivery of a fixed set of commodities by a capacitated vehicle. Specifically, the problem is defined over a directed graph G, where nodes represent locations and arcs are associated with non-negative travel costs. We are also given a set of commodities, each having a weight, a pickup location, and a delivery location. A solution to the m-PDTSP is a minimum-cost Hamiltonian tour on G where each commodity's pickup location must be visited prior to its corresponding delivery location, and the total weight carried by the vehicle never exceeds its capacity. Figure 4.1 depicts an example with 5 locations, two commodities and a vehicle with capacity 5, where an optimal tour starting and finishing at the depot (i.e., node 0) is presented in bold.

The m-PDTSP was introduced by Hernández-Pérez and Salazar-González (2009) and can be viewed as an important subproblem in vehicle routing applications, for instance when routes must be optimized for freight delivery (Holland et al., 2017). In contrast to classical pickup-and-delivery problems (Parragh et al., 2008), the locations in the m-PDTSP can be a source and a destination for multiple commodities at the same time. The problem thereby generalizes several related single-vehicle variants, such as the *pickup-and-delivery TSP* (Dumitrescu et al., 2008), where each location is either the source or destination of at most one commodity; the *sequential ordering problem* (Ascheuer et al., 2000), where the vehicle is uncapacitated; and the *one-commodity pickup-and-delivery TSP* (1-PDTSP) (Hernández-Pérez and Salazar-González, 2004), where all commodities are equivalent.

We propose a novel exact approach for the m-PDTSP that applies Lagrangian duality to combine a linear and a discrete relaxation of the problem. In particular, the discrete relaxation is encoded



Capacity (C): 5

Commodity	Weight (q_k)	Pickup Location (p_k)	Delivery Location (d_k)
1	3	1	4
2	2	3	2

Figure 4.1: Example of an m-PDTSP instance. Bold arcs represent the optimal tour.

as a relaxed multivalued decision diagram (MDD). In this work, we leverage the underlying network representation of an MDD to better exploit the combinatorial structure of the m-PDTSP while also incorporating dual information from a linear programming relaxation of the problem.

The resulting approach provides a flexible relaxation that yields bounds on the optimal solution value of the m-PDTSP and can be embedded, e.g., in any branching search. Computational experiments using a constraint programming solver indicate that the resulting MDD relaxation can enhance solution times by orders of magnitude in a number of instances, in particular when capacities are small. We also find provably optimal solutions to 33 instances in the literature for the first time, 27 of those with our best parametrization.

Main contributions. Our first contribution is to introduce an MDD-based discrete relaxation for the m-PDTSP. Its main purpose is to provide valid bounds on the optimal solution of the problem. Our data structure, inspired by a mixed-integer linear programming (MILP) formulation introduced by Gouveia and Ruthmair (2015), compactly encodes all feasible solutions as paths in a directed acyclic graph where edges represent positions in a tour. We present structural results and strategies for constructing relaxed MDDs that take into account both tour constraints and vehicle capacities, extending previous work on MDDs for sequencing problems (Cire and van Hoeve, 2013). Specifically, our capacity-based construction guarantees the satisfaction of the capacity constraint for all solutions represented in the relaxed MDD.

Our second key contribution is a Lagrangian technique that significantly strengthens the existing bounds for the m-PDTSP based on the concepts introduced by Bergman et al. (2015b). Namely, we incorporate Lagrange multipliers that penalize solutions of the MDD which do not represent valid Hamiltonian tours or violate precedence and capacity constraints. The key advantage of this framework is that it exploits the discrete representation of the m-PDTSP while still taking advantage of linear programming (LP) relaxation information, thereby benefiting from both MDD and LP methodologies.

Finally, we present an extensive numerical study that evaluates our MDD construction strategies and the performance of distinct MDD-based Lagrangian relaxations. To this end, we incorporate our relaxation mechanism into a constraint programming model and evaluate the quality of our bounds and the solution performance with respect to state-of-the-art techniques. We show that our integrated methodology can provide significant improvements over the existing dataset, which is more pronounced when the instance is associated with a small vehicle capacity relative to the commodity weights.

This work was published in the *INFORMS Journal of Computing* (Castro et al., 2020a).

Outline. This chapter is organized as follows. Section 4.1 presents a review of the previous related works. We formalize the m-PDTSP and introduce notation in Section 4.2. The section also describes a dynamic programming (DP) model and a mathematical formulation of the m-PDTSP that are used in our MDD-based Lagrangian technique. Section 4.3 introduces our MDD encoding for the m-PDTSP and the MDD construction produce based on the DP formulation. Section 4.4 presents our MDD-based Lagrangian relaxation and the procedure to solve the Lagragian dual problem. The overall approach is described in Section 4.5 and a constraint programming model for the m-PDTSP is introduced in Section 4.6. Numerical experiments are presented in Section 4.7, which includes a comparison with the state-of-the-art approaches. Finally, Section 4.8 provides a discussion on the work presented, the main results obtained, and final remarks.

4.1 Related Works

The m-PDTSP is a capacitated version of the sequential ordering problem (SOP), a variant of the asymmetric TSP where tours are subject to additional precedence constraints between locations. Several mathematical formulations and heuristic techniques have been previously investigated for the SOP; see e.g., Balas et al. (1995); Ascheuer et al. (2000); Hernádvölgyi (2004); Gouveia and Pesneau (2006). These works typically form the basis upon which the existing m-PDTSP mathematical models are built.

The m-PDTSP remains a challenging problem with few exact approaches in the literature. Existing solution methods are primarily based on MILPs that exploit polyhedral structure and decomposition strategies. The first models were investigated by Hernández-Pérez and Salazar-González (2009), who propose a multi-commodity flow and a path-based formulation. Both are solved using Benders decomposition strategies that iteratively add vehicle capacity constraints to a relaxed MILP model, solving instances with up to 47 locations in less than two hours. Nonetheless, these techniques are not able to solve smaller instances when tighter vehicle capacities were considered. Letchford and Salazar-González (2016) later extended this formulation by introducing valid inequalities for the original multi-commodity flow model. These inequalities significantly improve the LP relaxation bound at the root node, albeit negatively impacting their solution times due to computationally expensive separation routines.

The state-of-the-art solution methods for the m-PDTSP are branch-and-cut algorithms proposed by Gouveia and Ruthmair (2015). The authors show that the problem can be reduced to an 1-PDTSP by considering additional precedence constraints, and propose new MILP formulations based on *layered graph models*. These models formulate tours as paths in an expanded graph, where each layer corresponds to a position in the tour. The models either combine flow and capacity constraints or are restricted to enforcing precedence relations. The resulting MILPs are then solved by a branch-and-cut algorithm that combines preprocessing methods, primal heuristics based on a variable neighborhood descent, and separation routines. The authors significantly improve upon the existing run times, solving several open instances to optimality.

Despite the significant solution time improvements, the state-of-the-art method by Gouveia and Ruthmair (2015) reports instances with 20 locations and 10 commodities that are still left unsolved within a reasonable amount of time. Such instances typically involve small vehicle capacities relative to the commodity weights, leading to a combinatorial structure that is not well captured by current LP relaxations. Our model builds on the ideas by Gouveia and Ruthmair (2015), but our approach exploits a different type of approximation and focuses on operating directly on the MDD graphical structure.

As for heuristic methods, Rodríguez-Martín and Salazar-González (2012) propose two heuristics that combine a greedy randomized search with a variable neighborhood descent. Their best algorithm computed high-quality solutions for instances with up to 300 locations and 600 commodities, improving upon the best known solution for existing open instances.

Lagrangian duality is a bounding technique extensively investigated in mathematical programming; see, e.g., the review by Fisher (2004). The methodology, as described by Geoffrion (1974), consists of dropping constraints and penalizing their violation in the objective function, thereby leading to a relaxed problem that is easier and possibly decomposed. Such constraint violations, in particular, are weighted by *Lagrange multipliers*. The problem of finding the set of multipliers that provide the best possible bound defines a maximization problem with a piecewise linear concave objective known as the *Lagrangian dual*, initially tackled by Held and Karp (1971) using subgradient methods. This work considers the methodology by Frangioni (2002), which is a generalized version of the Bundle method introduced by Lemaréchal (1975).

The use of MDDs for routing problems was initially investigated by Cire and van Hoeve (2013) and Kinable et al. (2017) for other TSP variants, including the TSP with time windows and sequence-depend cost. We also note that the use of MDDs for Lagrangian duality was first proposed by Bergman et al. (2015b), who introduce the general concept and report preliminary results for the TSP with time windows. We refer the reader to Chapter 3 for a further discussion of these and other related works in the DD literature.

Our MDD approach also relates to the constrained shortest path problem (CSPP), in that finding the optimal solution to the m-PDTSP is equivalent to solving the CSPP over the relaxed MDD graph where the side constraints and costs are given by the m-PDTSP parameters (Section 4.4). Our Lagrangian approach and filtering procedure share the same underlying ideas as the techniques used to tackle the resource variant of the CSPP (Beasley and Christofides, 1989; Carlyle et al., 2008). Nonetheless, we generalize such techniques by leveraging the graphical structure of an MDD through refinement techniques that exploit the pickup-and-delivery constraint structure.

The proposed methodology can also be viewed as a type of state-space relaxation similar to what was considered by Baldacci et al. (2012) for the TSPTW. Nonetheless, while the latter work used dual information to strengthen a specific DP state-space relaxation (the *ng*-route), our diagram can be constructed to better exploit the m-PDTSP structure.

4.2 Problem Definition and Formulations

This section presents the m-PDTSP problem and the notation used throughout the chapter. We also introduce dynamic programming and mathematical programming formulations that will be combined in our MDD-based Lagrangian dual methodology.

The m-PDTSP is defined on a complete directed graph $G = (V, E)$, where $V = \{0, \dots, n\}$ is a set of $n + 1$ locations with 0 as the depot. Each arc $(i, j) \in E$ is associated with a non-negative travel cost $c_{i,j} \geq 0$, where potentially $c_{i,j} \neq c_{j,i}$ (i.e., costs may be asymmetric). We are also given a set of m commodities, $K = \{1, \dots, m\}$. A commodity $k \in K$ has an integral positive weight $q_k \in \mathbb{Z}_+$, a pickup location $p_k \in V \backslash \{0\}$, and a delivery location $d_k \in V \backslash \{0\}$. Finally, we assume a single vehicle with capacity limit $C \in \mathbb{Z}_+$.

A solution for the m-PDTSP is a minimum-cost Hamiltonian tour over G that observes the vehi-

cle capacity and the pickup-and-delivery order established for each commodity. This can be formally represented using the 1-PDTSP reduction by Gouveia and Ruthmair (2015). Namely, let

$$\Delta q_i = \sum_{k \in K, p_k = i} q_k - \sum_{k \in K, d_k = i} q_k,$$

be the net weight of pickup and deliveries at a location $i \in V$. A feasible tour to the m-PDTSP is a sequence of locations $\boldsymbol{x} := (x_0, x_1, \ldots, x_n, x_{n+1})$ which satisfies (i)-(iii):

(i) The sequence starts and ends at the depot, $x_0 = x_{n+1} = 0$, and $(x_1, \ldots, x_n)$ is a permutation of $V \setminus \{0\}$.

(ii) The accumulated net weight in every position of the sequence never exceeds the vehicle capacity, i.e., $\sum_{t=1}^{t'} \Delta q_{x_t} \in \{0, \ldots, C\}$ for all $t' \in \{1, \ldots, n\}$; and

(iii) For every commodity $k \in K$, its pickup location is visited prior to its delivery location, i.e., $x_t = p_k$ and $x_{t'} = d_k$ implies $t < t'$.

The cost $c(\boldsymbol{x})$ of a feasible tour $\boldsymbol{x}$ is the total travel cost starting at the depot, visiting each location in V in the order defined by $\boldsymbol{x}$, and returning to the depot. That is,

$$c(\boldsymbol{x}) = \sum_{t=0}^{n} c_{x_t, x_{t+1}}.$$

The m-PDTSP requires a feasible tour $\boldsymbol{x}$ that minimizes the tour cost $c(\boldsymbol{x})$. The optimal tour cost is denoted by ν^*.

Example 4.1 Figure 4.1 depicts an instance of the m-PDTSP used as a running example in the text. The underlying graph has five locations (with 0 as the depot), where travel costs are represented as arc labels. The vehicle capacity is $C = 5$ and two commodities must be considered, with weights and pickup-and-delivery locations described in the figure's table. Thus, the net weight of each location is given by $\Delta q_1 = 3$, $\Delta q_2 = -2$, $\Delta q_3 = 2$, and $\Delta q_4 = -3$.

The optimal tour is $\boldsymbol{x} = (0, 3, 1, 2, 4, 0)$ with cost $\nu^* = 1863$, as represented by the bold arcs in Figure 4.1. The tour picks up commodity 2 at location 3, then picks commodity 1 at location 1, and finally delivers commodities 2 and 1 at locations 2 and 4, respectively. The net weight in $\boldsymbol{x}$ is always between 0 and the vehicle capacity $C = 5$. Furthermore, the delivery location for each commodity succeeds its corresponding pickup location. □

4.2.1 Dynamic Programming Formulation

We now formalize the m-PDTSP in terms of a recursive model that we later use to construct our MDD relaxation (see in Section 4.3). Our recursive model is an extension of the classical TSP dynamic programming formulation (Held and Karp, 1962) with capacity and precedence constraints. We also simplify the state representation by omitting the last location visited. This state information is needed to compute the objective function as a shortest path over its corresponding state transition graph. However, as explain in Section 2.2.4, we can omit this information by storing the accumulated objective cost over the arcs instead.

Our recursive model $\mathcal{R}$ considers decision variables $\boldsymbol{x}$ as defined above and $n+3$ stages. For a given stage t, the state variables are defined as $\boldsymbol{S} = (L, Q)$ where L represents the set of locations visited and Q the load of the vehicle. Thus, the state-space at stage t is $\mathcal{S}_t = 2^V \times \{0, \dots, C\}$. We define the transition function as $\phi_t(\boldsymbol{S}, x) = (L \cup \{x\}, Q + \Delta q_x)$, i.e., each decision x at stage t updates the number of locations visited and the load of the vehicle, respectively. Consider set $P(i)$ as the locations that need to be visited before reaching location $i \in V$, i.e., $P(i) = \{j \in V : \exists k \in K \text{ such that } p_k = j \text{ and } d_k = i\}$. Then, the feasibility set $X_t(L, Q)$ for a state $\boldsymbol{S} = (L, Q) \in \mathcal{S}_t$ is given by

$$X_t(L,Q) = \{x \in V \setminus \{0\} : \;\; x \notin L, \;\; 0 \leq Q + \Delta q_x \leq C, \;\; P(x) \subseteq L\}, \qquad \forall t \in \{1, \dots, n\}, \tag{4.1}$$

and $X_0(L, Q) = X_{n+1}(L, Q) = \{0\}$ for the first and last stage, respectively. The immediate cost at stage $t \in \{1, \dots, n+1\}$ is $f_t(x) = c_{x_{t-1},x}$ and $f_0(x) = 0$ at the initial stage. The minimum accumulated cost $h_t(L, Q)$ for state $\boldsymbol{S} = (L, Q)$ at stage t is given by:

$$h_t(L,Q) = \min_{x \in X_{t+1}(L,Q)} \left\{ c_{x_{t-1},x} + h_t(L \cup \{x\},\; Q + \Delta q_x) \right\} \qquad t \in \{0, \dots, n+1\}, \tag{$\mathcal{R}$}$$

where the initial state is given by $\boldsymbol{S}_0 = (\emptyset, 0)$ and the accumulated cost at stage $n+2$ is $h_{n+2}(L, Q) = 0$.

Note that every solution of $\mathcal{R}$ is a feasible tour for the m-PDTSP. The feasibility sets at the first and last stage require the tour to start and end at the depot, i.e., $x_0 = x_{n+1} = 0$. Also, the feasibility set $X_t(L, Q)$ defined in (4.1) has three constraints imposing conditions (i), (ii), and (iii), respectively. Lastly, the optimal tour cost can be computed using the cumulative arc length procedure over the state-transition graph, as shown in Section 2.2.4.

4.2.2 Mathematical Programming Formulation

We now present an integer linear program (ILP) for the m-PDTSP that we later use to enhance our relaxed MDD bounds via Lagrangian Duality (see Section 4.4) The m-PDTSP has a large array of formulations proposed in the literature (Letchford and Salazar-González, 2016). We consider a well-known time-indexed formulation for the TSP (see, e.g., Dash et al. 2012) augmented with precedence constraints.

Let z_{ij}^t be a binary variable indicating if location j follows location i at position t in the tour, for $i, j \in V$ and $t \in \{0, \dots, n\}$. Also, let $y_{i,t} \in \{0, 1\}$ be a binary variable that indicates if location i is visited in position t of a feasible tour. Thus, the time-indexed formulation $\mathcal{T}$ for the m-PDTSP is as follows:

$$\min_{\boldsymbol{z},\boldsymbol{y}} \quad \sum_{t=0}^{n} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{i,j}\, z_{i,j}^t \tag{$\mathcal{T}$}$$

$$\text{s.t.} \quad \sum_{i=0}^{n} y_{i,t} = 1 \qquad t \in \{0, \dots, n+1\}, \tag{4.2a}$$

$$\sum_{t=1}^{n} y_{i,t} = 1 \qquad i \in V \setminus \{0\}, \tag{4.2b}$$

$$\sum_{t=1}^{t'} \sum_{i=1}^{n} y_{i,t} \cdot \Delta q_i \geq 0 \qquad t' \in \{1, \dots, n\}, \tag{4.2c}$$

$$\sum_{t=1}^{t'} \sum_{i=1}^{n} y_{i,t} \cdot \Delta q_i \leq C \qquad t' \in \{1, \dots, n\}, \tag{4.2d}$$

$$\sum_{t=1}^{n} t \cdot y_{p_k,t} - \sum_{t=1}^{n} t \cdot y_{d_k,t} \leq -1 \qquad k \in K, \tag{4.2e}$$

$$y_{0,0} = y_{0,n+1} = 1,\ y_{0,t} = 0 \qquad t \in \{1, \dots, n\}, \tag{4.2f}$$

$$y_{i,t} - \sum_{j=0}^{n} z_{i,j}^{t} = 0 \qquad i \in V,\ t \in \{0, \dots, n\}, \tag{4.2g}$$

$$y_{j,t} - \sum_{i=0}^{n} z_{i,j}^{t-1} = 0 \qquad j \in V,\ t \in \{1, \dots, n+1\}, \tag{4.2h}$$

$$y_{i,t} \in \{0,1\} \qquad i \in V,\ t \in \{0, \dots, n+1\},$$

$$z_{i,j}^{t} \in \{0,1\} \qquad i,j \in V,\ t \in \{0, \dots, n\}.$$

The objective function is a reformulation of the tour cost in terms of $\boldsymbol{z}$. Equalities (4.2a) and (4.2b) are matching constraints enforcing that tour positions are assigned to exactly one location and that each location must be visited exactly once, respectively. Inequalities (4.2c) and (4.2d) state the vehicle capacity limitation. Inequalities (4.2e) impose precedence constraints, i.e., each pickup location is visited prior to its delivery location. Such precedence inequalities are typical, e.g., in time-indexed formulations of resource-constrained project scheduling problems (Artigues, 2017). The equalities in (4.2f) indicate that a tour should start and end at the depot. Lastly, (4.2g) and (4.2h) establish the connection between variables $\boldsymbol{z}$ and $\boldsymbol{y}$.

A feasible solution $\boldsymbol{y}'$ to $\mathcal{T}$ defines a feasible tour $\boldsymbol{x}'$ such that $x'_t = \sum_{i=0}^{n} i\, y'_{i,t}$ for every $t \in \{0, \dots, n+1\}$. Every tour can be converted to a feasible solution to $\mathcal{T}$ analogously. Also, there is an one-to-one mapping between feasible binary vectors $\boldsymbol{y}$ and $\boldsymbol{z}$ based on (4.2g)-(4.2h).

While other m-PDTSP formulations are also applicable in our framework, model $\mathcal{T}$ has two advantages that we exploit. First, $\mathcal{T}$ has a polynomial number of linear inequalities, which when relaxed leads to a tractable number of Lagrange multipliers that can be efficiently optimized. Second, the time-indexed variables $\boldsymbol{y}$ have a direct translation to the layered network representation of a decision diagram, as described in Section 4.4.

4.3 Multivalued Decision Diagram Encoding

We now introduce a MDD encoding for the m-PDTSP based on recursive model $\mathcal{R}$ and the sequencing representation by Cire and van Hoeve (2013) and Kinable et al. (2017). The model is a graphical representation of the feasible tours set of an m-PDTSP instance, which can be limited in size to provide valid bounds for the m-PDTSP. In the following, we rely on the notation introduced in Chapter 2 to describe our MDD encoding.

An MDD for the m-PDTSP is a directed acyclic layered graph $\mathcal{M} = (\mathcal{N}, \mathcal{A})$, where the set of nodes $\mathcal{N}$ is partitioned into $n+3$ layers $\mathcal{N}_0, \dots, \mathcal{N}_{n+2}$. We associate a value $v_a \in V$ to each arc $a \in \mathcal{A}$ emanating from $\mathcal{N}_t$ that represents the location assigned to the t-th position of a tour; i.e., paths traversing arc a are such that $x_t = v_a$. Thus, the set of solutions encoded by the MDD, i.e., $\text{Sol}(\mathcal{M})$, corresponds to all possible variable assignments for $\boldsymbol{x}$.

We say that an MDD $\mathcal{M}$ exactly represents an instance of the m-PDTSP if there is an one-to-one

correspondence between Sol($\mathcal{M}$) and all the feasible tours. Alternatively, an MDD is relaxed if Sol($\mathcal{M}$) over-approximates the set of feasible tours, i.e., every feasible tour is encoded in some $\mathbf{r}-\mathbf{t}$ path of $\mathcal{M}$, but not all paths in Sol($\mathcal{M}$) are necessarily feasible tours. Specifically, infeasible tours in $\mathcal{M}$ may either represent invalid permutations, violate the vehicle capacity, or fail to observe pickup-and-delivery precedence constraints.

Example 4.2 Figure 4.2 depicts an exact and relaxed MDD for the instance in Example 4.1. Each $\mathbf{r}-\mathbf{t}$ path in the exact MDD (left) encodes a feasible tour and equivalently every tour is encoded by exactly one $\mathbf{r}-\mathbf{t}$ path. In particular, path $(\mathbf{r}, u_1, u_3, u_5, u_8, u_9, \mathbf{t})$, in bold, encodes the optimal tour $\boldsymbol{x} = (0, 3, 1, 2, 4, 0)$. Every path in the exact MDD has an associated path in the relaxed MDD (right). Nonetheless, the relaxed MDD contains the infeasible tour $(0, 1, 4, 1, 4, 0)$ given by the shaded path $(\mathbf{r}, u_1, u_2, u_5, u_6, u_8, \mathbf{t})$. □

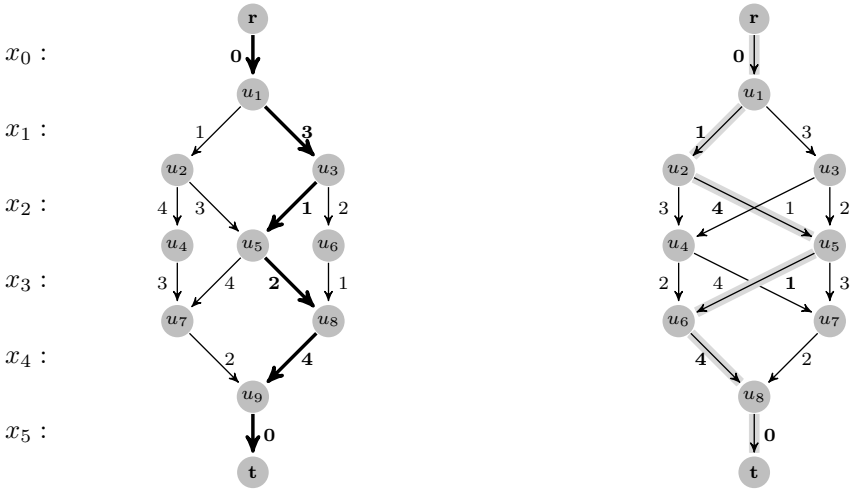


Figure 4.2: Two MDDs for Example 4.1. The left illustration corresponds to an exact MDD, while the right graphic is a relaxed MDD.

Our relaxed MDD construction procedure uses a variant of the incremental refinement framework described in Chapter 2, Algorithm 3. Specifically, we propose a construction mechanism that enforces each of the three constraint classes, i.e., conditions (i)-(iii), one at a time until either they are satisfied by all tours in the MDD, or the maximum width $\mathcal{W}$ is reached. To this end, we start with a small valid relaxed MDD and rule out the solutions violating a particular constraint by splitting nodes and removing arcs accordingly. Once a constraint type has been fully observed by the paths of the MDD, we repeat this procedure iteratively with the remaining constraints.

The splitting strategy is depicted in Algorithm 6. For each layer, we split nodes to first satisfy the vehicle capacity constraints (procedure SplitNodesCapacity), and second to satisfy the tour constraints and precedence relations (procedure SplitNodesTour), while ensuring that the width $w(\mathcal{M})$ of the network does not exceed $\mathcal{W} > 0$. We give priority to capacity constraints, which are well-known to be challenging in integer programming formulations for the m-PDTSP (Letchford and Salazar-González, 2016), while conversely MDD relaxations may be weak when enforcing tour constraints (Cire and van Hoeve, 2013) and are better represented by linear assignment constraints. However, the order of the splitting procedures can be reversed without loss of generality.

The following sections describe in detail the main components of the relaxed MDD construction

Algorithm 6 Relaxed MDD splitting for the m-PDTSP

```
1: procedure SplitDDNodes(𝒩_t, 𝒲)
2:     SplitNodesCapacity(𝒩_t, 𝒲)
3:     SplitNodesTour(𝒩_t, 𝒲)
```

procedure based on the framework introduced in Chapter 2. Section 4.3.1 presents the relaxed state and the filtering rules based on our recursive model $\mathcal{R}$. Sections 4.3.2 and 4.3.3 introduce our two node splitting schemes to satisfy the capacity and tour constraints, respectively. Lastly, Section 4.3.4 explains how to compute MDD-based lower bounds for the non-linear objective in $\mathcal{R}$ based on the shortest-path procedure presented by Kinable et al. (2017).

4.3.1 Relaxed States and Filtering

As introduced in Chapter 2, relaxed states are approximations of the state variables of $\mathcal{R}$ and are stored in each MDD node to identify infeasible arcs and split nodes during the MDD construction procedure. We now introduce the relaxed states for state variable $\boldsymbol{S} = (L, Q)$ and their corresponding filtering rules.

Load Relaxed States. We first introduce the relaxed state and filtering rule for the vehicle load state variables Q. These state variables are used to impose the capacity limit, which can be represented as two separable inequalities over Q:

$$0 \leq Q + \Delta q_x \quad \text{and} \quad Q + \Delta q_x \leq C, \qquad \forall \boldsymbol{S} = (L, Q) \in \mathcal{S}_t,\ x \in X_t(\boldsymbol{S}),\ t \in \{0, \dots, n\}. \tag{4.3}$$

Thus, we create relaxed states and filtering rules for state variable Q based on the separable inequalities case introduced in Section 2.2.2. For any node $u \in \mathcal{N}_t$, $Q_{\min}(u)$ and $Q_{\max}(u)$ correspond to the top-down relaxed states for Q that under and over approximate the value of Q, respectively, for all variable assignments associated to $\mathbf{r} - u$ paths in $\mathcal{M}$. Thus, their values are calculated via a top-down pass through $\mathcal{M}$ using the following recursions for all nodes $u \in \mathcal{N}$:

$$\begin{aligned} Q_{\min}(\mathbf{r}) &= Q_{\max}(\mathbf{r}) = 0, \\ Q_{\min}(u) &= \min_{a \in \mathcal{A}^{\text{in}}(u)} \{Q_{\min}(s(a)) + \Delta q_{v_a}\}, \quad \text{and} \\ Q_{\max}(u) &= \max_{a \in \mathcal{A}^{\text{in}}(u)} \{Q_{\max}(s(a)) + \Delta q_{v_a}\}. \end{aligned}$$

We used the above relaxed states to identify and remove infeasible arcs, i.e., whenever all $\mathbf{r} - \mathbf{t}$ paths traversing an arc $a \in \mathcal{A}$ violate (4.3). Specifically, we say that an arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_t$ can be removed from $\mathcal{M}$ if:

$$Q_{\min}(s(a)) + \Delta q_{v_a} > C \quad \text{or} \quad Q_{\max}(s(a)) + \Delta q_{v_a} < 0. \tag{CAP-R1}$$

The validity of the above filtering rules follow from the relaxed state definitions and inequality (4.3).

We also introduce bottom-up relaxed states $Q^{\uparrow}_{\min}(\cdot), Q^{\uparrow}_{\max}(\cdot) \in \mathbb{Z}$ that are symmetric versions of $Q_{\min}(\cdot)$ and $Q_{\max}(\cdot)$. Namely, the relaxed states $Q^{\uparrow}_{\min}(u)$ and $Q^{\uparrow}_{\max}(u)$ represent the minimum and maximum accumulated net weights, respectively, of all partial paths starting at node $u \in \mathcal{N}$ and ending at the terminal node $\mathbf{t}$. Thus, analogously to the previous case, we update the relaxed states for each

node $u \in \mathcal{N}$ using the following recursions:

$$
\begin{aligned}
&Q_{\min}(\mathbf{t}) = Q_{\max}(\mathbf{t}) = 0, \\
&Q^{\uparrow}_{\min}(u) = \min_{a \in \mathcal{A}^{\text{out}}(u)} \left\{ Q^{\uparrow}_{\min}(t(a)) + \Delta q_{v_a} \right\}, \quad \text{and} \\
&Q^{\uparrow}_{\max}(u) = \max_{a \in \mathcal{A}^{\text{out}}(u)} \left\{ Q^{\uparrow}_{\max}(t(a)) + \Delta q_{v_a} \right\}.
\end{aligned}
$$

We utilize these bottom-up states to create a new set of filtering rules based on the fact that the vehicle is empty at the end of any feasible tour. Specifically, we say that an arc a is infeasible if the net weights of all tours corresponding to $\mathbf{r}-\mathbf{t}$ paths traversing a are different to zero. Proposition 4.1 shows two filtering rules based on this condition and proves their validity.

Proposition 4.1. *An arc $a \in \mathcal{A}$ can be removed from $\mathcal{M}$ if:*

$$
Q_{\min}(s(a)) + \Delta q_{v_a} + Q^{\uparrow}_{\min}(t(a)) > 0, \quad \text{or} \tag{CAP-R2}
$$

$$
Q_{\max}(s(a)) + \Delta q_{v_a} + Q^{\uparrow}_{\max}(t(a)) < 0. \tag{CAP-R3}
$$

Proof. By the definition of each relaxed state, it follows that any tour $\boldsymbol{x} \in \text{Sol}(\mathcal{M})$ from a $\mathbf{r}-\mathbf{t}$ path which includes arc a satisfies

$$
Q_{\min}(s(a)) + \Delta q_{v_a} + Q^{\uparrow}_{\min}(t(a)) \leq \sum_{t=0}^{n+1} \Delta q_{x_t} \leq Q_{\max}(s(a)) + \Delta q_{v_a} + Q^{\uparrow}_{\max}(t(a)).
$$

Therefore, $\boldsymbol{x}$ is infeasible if either CAP-R2 or CAP-R3 are satisfied. ■

Location Relaxed States. The relaxed states for state variable L are based on the relaxed states for the all-different combinatorial structure (see Section 2.2.3). Condition (i) states that, except for the first and last stage, a feasible tour visits a different location at each stage. Thus, the relaxed states and filtering rules for the all-different structure are also valid for the m-PDTSP.

For a given node $u \in \mathcal{N}$, we consider two top-down relaxed states, $L_{\text{all}}(u)$ and $L_{\text{some}}(u)$, to under and over approximate the set of locations visited given by all $\mathbf{r}-u$ paths. Intuitively, $L_{\text{all}}(u)$ represents the set of locations visited by all $\mathbf{r}-u$ tours, while $L_{\text{some}}(u)$ corresponds to the locations visited by at least some $\mathbf{r}-u$ tour. These values are calculated during a top-down pass through $\mathcal{M}$ using recursions (2.6) and (2.7) as explain in Section 2.2.3.

Analogously, we define the bottom-up relaxed states $L^{\uparrow}_{\text{all}}(u)$ and $L^{\uparrow}_{\text{some}}(u)$ for each node $u \in \mathcal{N}$. As presented in Section 2.2.3, we use recursions (2.8) and (2.9) to calculate these relaxed states during the bottom-up pass through $\mathcal{M}$. Then, these top-down and bottom up relaxed states allow us to apply filtering rules AD-R1 to AD-R5 to remove infeasible arcs with respect to the tour condition (i). Since location 0 (i.e., the depot) appears twice in each feasible tour, we slightly modify these filtering rules to consider this corner case.

We also employ the top-down and bottom-up relaxed states to filter out arcs that violate the precedence condition (iii), i.e., deliver a commodity before picking it. We implement the filtering rules introduced by Cire and van Hoeve (2013) for precedence relationships. To do so, for each location $i \in V$, consider $P(i)$ as defined in Section 4.2 and $D(i)$ as the locations that need to be visited after reaching i, i.e., $D(i) = \{j \in V : \exists k \in K \text{ such that } d_k = j \text{ and } p_k = i\}$. Then, we say that an arc $a \in \mathcal{A}$ is infeasible

if:

$$P(v_a) \not\subseteq L_{\text{some}}(s(a)) \qquad \text{or} \qquad D(v_a) \not\subseteq L^{\uparrow}_{\text{some}}(t(a)). \tag{PRE-R1}$$

Thus, any arc a satisfying PRE-R1 would represent tours that deliver a commodity before picking it up (left-condition) or pickup a commodity without ever delivering it (right-condition).

4.3.2 Capacity Constraint-based Refinement

In this section, we develop the SplitNodesCapacity procedure for Algorithm 6. Its main purpose is to modify a given relaxed MDD so that the vehicle capacity constraints are satisfied by the paths in $\text{Sol}(\mathcal{M})$. Specifically, the load relaxed states, $Q_{\min}(\cdot)$ and $Q_{\max}(\cdot)$, provide a mechanism to measure the degree of infeasibility of $\mathcal{M}$ with respect to the capacity constraints. This is formalized in the proposition below.

Proposition 4.2. *For all $u \in \mathcal{N}$ and some $\epsilon \geq 0$, suppose that*

(i) $Q_{\min}(u) \leq C$, $Q_{\max}(u) \geq 0$; *and*

(ii) $Q_{\max}(u) - Q_{\min}(u) \leq \epsilon$.

Then, for all $\boldsymbol{x} \in \text{Sol}(\mathcal{M})$,

$$-\epsilon \leq \sum_{t=0}^{t'} \Delta q_{x_t} \leq C + \epsilon, \qquad t' \in \{0, \dots, n+1\}. \tag{4.4}$$

That is, tours in $\mathcal{M}$ violate the capacity constraints by at most ϵ.

Proof. Assume (i) and (ii) hold for some $\epsilon \geq 0$ as defined in the proposition statement. For any partial path $\boldsymbol{x}$ from the root $\mathbf{r}$ to a node $u \in \mathcal{N}_t$, $t \geq 1$,

$$\sum_{s=0}^{t-1} \Delta q_{x_s} \leq Q_{\max}(u) \leq Q_{\min}(u) + \epsilon \leq C + \epsilon.$$

The first inequality follows from the definition of $Q_{\max}(\cdot)$. The second follows from condition (ii), and the last is from (i). Equivalent reasoning can be used for $\sum_{t=1}^{t'} \Delta q_{x_t} \geq -\epsilon$. ■

We can assume condition (i) from Proposition 4.2 always holds, since otherwise we can simply remove the violating nodes from $\mathcal{M}$ as they only encode infeasible paths. This leads directly to the following corollary.

Corollary 4.1. *The tours in* $\text{Sol}(\mathcal{M})$ *satisfy the vehicle capacity constraints if*

$$0 \leq Q_{\min}(u) = Q_{\max}(u) \leq C, \qquad \forall\, u \in \mathcal{N}_t,\ t \in \{0, \dots, n+1\}.$$

We can now state our splitting procedure, which is formalized in Algorithm 7. Given the nodes $\mathcal{N}_t$ in a layer t, we first define the splitting set $S^Q(\mathcal{N}_t)$ of $\mathcal{N}_t$ as the set of nodes in $\mathcal{N}_t$ from which all parents satisfy the conditions of Corollary 4.1, i.e.,

$$S^Q(\mathcal{N}_t) := \left\{ u \in \mathcal{N}_t \,:\, Q_{\min}(s(a)) = Q_{\max}(s(a)) \ \text{ for all } a \in \mathcal{A}^{\text{in}}(u) \right\}.$$

Notice that, by definition of the relaxed states, a node $u' \notin S^Q(\mathcal{N}_t)$ can never be split to satisfy the corollary conditions, since the minimum net weight at u' will always be strictly lower than its maximum net weight regardless on how its incoming arcs are partitioned.

If all nodes $u \in S^Q(\mathcal{N}_t)$ are such that $Q_{\min}(u) = Q_{\max}(u)$, then, by Corollary 4.1, no more splitting is needed. Otherwise, consider $u \in S^Q(\mathcal{N}_t)$ such that $Q_{\min}(u) < Q_{\max}(u)$. Furthermore, define $\mathcal{A}^{\text{in}}_{\min}(u)$ as the set of incoming arcs at u that certify the label $Q_{\min}(u)$, i.e.,

$$\mathcal{A}^{\text{in}}_{\min}(u) := \left\{a \in \mathcal{A}^{\text{in}}(u) \,:\, Q_{\min}(s(a)) + \Delta q_{v_a} = Q_{\min}(u)\right\}. \tag{4.5}$$

If the width limit $\mathcal{W}$ is not met, we create a new node u' and redirect the arcs in $a \in \mathcal{A}^{\text{in}}_{\min}(u)$ to u'. This redirection will ensure, by construction, that $Q_{\min}(u') = Q_{\max}(u')$ and will increase $Q_{\min}(u)$ as shown in Proposition 4.3. We also copy the arcs emanating from u and assign them to emanate from u' to ensure that the paths originally crossing arcs in $\mathcal{A}^{\text{in}}_{\min}(u)$ are preserved. Finally, we update the labels $Q_{\min}(\cdot)$ and $Q_{\max}(\cdot)$ accordingly and repeat the procedure until either the maximum width is met or Corollary 4.1 is satisfied for the nodes in that layer. See Algorithm 7 for a detail explanation of the capacity-based splitting procedure.

Algorithm 7 Splitting nodes based on capacity.

1: **procedure** SplitNodesCapacity($\mathcal{N}_t$, $\mathcal{W}$)
2: **while** $|\mathcal{N}_t| < \mathcal{W}$ and $\exists u \in S^Q(\mathcal{N}_t)$ such that $Q_{\max}(u) - Q_{\min}(u) > 0$ **do**
3: Create a new node u', add it to $\mathcal{N}_t$.
4: Set $s(a^*) = u'$ for all $a^* \in \mathcal{A}^{\text{in}}_{\min}(u)$, as defined in (4.5).
5: For every arc $a \in \mathcal{A}^{\text{out}}(u)$, create a new arc $a' = (u', t(a))$.
6: Update labels $Q_{\min}(u), Q_{\max}(u), Q_{\min}(u'), Q_{\max}(u')$.

Proposition 4.3. *For a sufficiently large $\mathcal{W}$, the procedure SplitNodesCapacity ensures that, for every node $u \in \mathcal{N}_t$, $Q_{\min}(u) = Q_{\max}(u)$.*

Proof. It suffices to show that the procedure terminates for any arbitrarily large $\mathcal{W}$. Assume all previous layers satisfy the condition of the statement and consider an iteration that chooses a certain u such that $Q_{\min}(u) < Q_{\max}(u)$. The new node u' satisfies $Q_{\min}(u') = Q_{\max}(u')$ because of (4.5). Moreover, for all arcs $a' \in \mathcal{A}^{\text{in}}(u) \setminus \mathcal{A}^{\text{in}}_{\min}(u)$,

$$Q_{\min}(s(a')) + \Delta q_{v_{a'}} > Q_{\min}(u)$$

and therefore the updated relaxed states $Q_{\min}(u)$ strictly increases. Since $Q_{\max}(u)$ is finite and remains constant for the subsequent iterations that pick the same node u, the result follows. ■

Proposition 4.3 and Corollary 4.1 ensure that, for a sufficiently large width $\mathcal{W}$, all tours in $\mathcal{M}$ satisfy the capacity constraints. Note that, if the input node set has a cardinality of one (as in our width-one MDD case), then the maximum width required for Proposition 4.3 is $C + 1$, since no two nodes will have the same labels. Second, the choice of node u in line 2 of Algorithm 7 can be done in a systematic fashion. For instance, if we choose the node with maximum $\epsilon := Q_{\max}(u) - Q_{\min}(u)$, we move towards decreasing the total violation ϵ of paths, based on Proposition 4.2. In our numerical experiments, we choose $\mathcal{W}$ to be sufficiently large (i.e., at least $C + 1$) to ensure all paths satisfy vehicle capacity.

Example 4.3 Figure 4.3 shows three relaxed MDDs that illustrate the SplitNodesCapacity procedure for our running example. Notice that shaded arrows indicate infeasible arcs identified by our filtering

rules (Section 4.3.1). Starting with the width-one MDD (left diagram), the procedure splits layer $\mathcal{N}_2$ that has a single node u_2 with $Q_{\min}(u_2) = -3$ and $Q_{\max}(u_2) = 3$. The resulting MDD corresponds to the middle diagram in Figure 4.3 and has two nodes in layer $\mathcal{N}_2$ satisfying Corollary 4.1. Similarly, the right most diagram illustrates the SplitNodesCapacity procedure when it is applied to $\mathcal{N}_3$. □

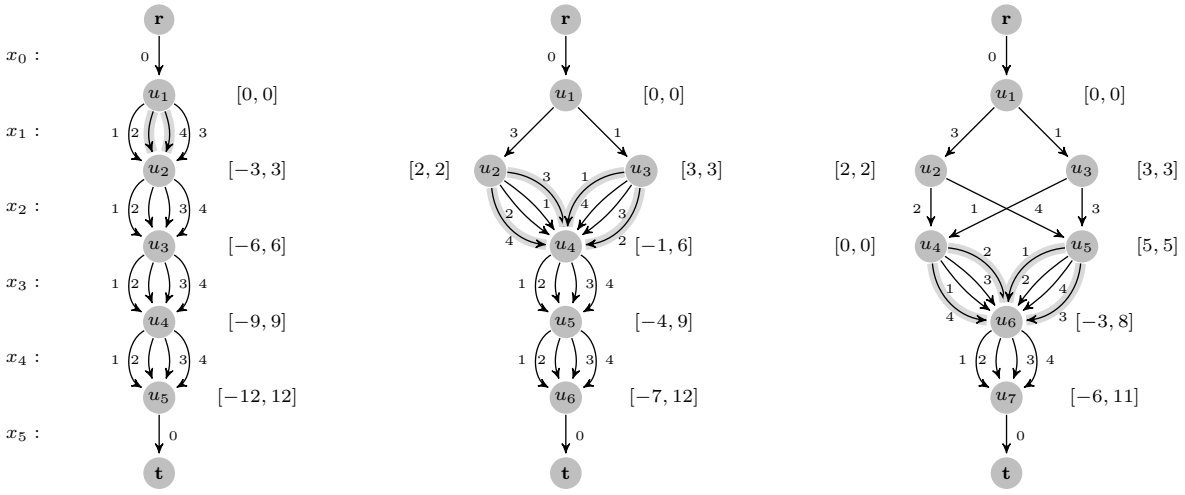


Figure 4.3: MDD construction algorithm. Depicts SplitNodesCapacity procedure and filtering rules.

4.3.3 Tour and Precedence Constraints-based Refinement

We now develop the SplitNodesTour procedure for Algorithm 6 to impose that tours in a relaxed MDD $\mathcal{M}$ do not violate precedence and tour constraints. To do so, we rely on the location relaxed states $L_{\text{all}}(\cdot)$ and $L_{\text{some}}(\cdot)$ to define sufficient conditions for which all tours in $\text{Sol}(\mathcal{M})$ satisfy the tour and precedence constraints, as shown in the following proposition.

Proposition 4.4. *For all $u \in \mathcal{N}_t$, $t \in \{0, \dots, n+1\}$, suppose that*

(i) $|L_{\text{all}}(u)| = t$*; and*

(ii) For all arcs $a \in \mathcal{A}^{\text{out}}(u)$ emanating from u, the deliveries in v_a are preceded by their associated pickups in $L_{\text{all}}(u)$, i.e., if $d_k = v_a$ for some commodity $k \in K$, then $p_k \in L_{\text{all}}(u)$.

If (i) and (ii) hold, the solutions $\boldsymbol{x} \in \text{Sol}(\mathcal{M})$ satisfy the tour and precedence constraints.

Proof. We show by induction on t that, for any node $u \in \mathcal{N}_t$, the tours associated with $\mathbf{r} - u$ paths satisfy the tour and precedence constraints. This is trivially valid for the base case $t = 0$. Assume now this statement holds for $t \in \{0, \dots, t'\}$ for some $t' \geq 1$.

Pick any node $u' \in \mathcal{N}_{t'+1}$. For an arc $a \in \mathcal{A}^{\text{in}}(u')$ and a tour $\boldsymbol{x}' = (x_0, x_1, \dots, x_{t'-1})$ encoded by an $\mathbf{r} - s(a)$ path, the relaxed state definition implies that $|L_{\text{all}}(u')| = t'$ holds only if $v_a \neq x_t$ for all $t \in \{0, \dots, t'-1\}$; i.e., the extended tour $\boldsymbol{x} := (\boldsymbol{x}', v_a)$ satisfies the tour constraints. Moreover, assumption (ii) directly implies that $\boldsymbol{x}$ also satisfies the precedence constraints. ■

Algorithm 8 states the tour splitting procedure based on the results in Proposition 4.4. We first define the splitting set $S^L(\mathcal{N}_t)$ of $\mathcal{N}_t$ as the set of nodes in $\mathcal{N}_t$ from which all parents satisfy the assumption

(i) of Proposition 4.4:

$$S^L(\mathcal{N}_t) := \left\{u \in \mathcal{N}_t \,:\, |L_{\text{all}}(s(a))| = t-1 \;\text{ for all } a \in \mathcal{A}^{\text{in}}(u)\right\}.$$

Notice that, by the relaxed state definition, a node $u' \notin S^L(\mathcal{N}_t)$ can never be split to satisfy the required assumption, since the cardinality of L_{all} can increase by at most one in each layer.

If all nodes $u \in S^L(\mathcal{N}_t)$ are such that $|L_{\text{all}}(u)| = t$, then, by Proposition 4.4, no more splitting is needed and we can stop. Otherwise, let $u \in S^L(\mathcal{N}_t)$ be a node such that $|L_{\text{all}}(u)| < t$. Furthermore, for any $a' \in \mathcal{A}^{\text{in}}(u)$, define $\mathcal{A}^{\text{in}}_{\text{all}}(u, a')$ as the set of incoming arcs at u that lead to the same label $L_{\text{all}}(\cdot)$ as when applying a', i.e.,

$$\mathcal{A}^{\text{in}}_{\text{all}}(u, a') := \left\{a \in \mathcal{A}^{\text{in}}(u) \,:\, L_{\text{all}}(s(a)) \cup \{v_a\} = L_{\text{all}}(s(a')) \cup \{v_{a'}\}\right\}. \tag{4.6}$$

If the width limit $\mathcal{W}$ is not met, we select a node $u \in S^L(\mathcal{N}_t)$ and any arc $a' \in \mathcal{A}^{\text{in}}(u)$. We then create a new node u' and redirect the arcs in $a \in \mathcal{A}^{\text{in}}_{\text{all}}(u, a')$ to u', which imposes assumption (i) from Proposition 4.4. We also copy the arcs emanating from u to emanate from u' to ensure that the paths originally crossing arcs in $\mathcal{A}^{\text{in}}_{\text{all}}(u, a')$ are preserved, update the relaxed states accordingly, and repeat the procedure. Note that the choice of a' and u in Algorithm 8 can also be done in a systematic way, as discussed by Cire and van Hoeve (2013).

Procedure SplitNodesTour ensures the fulfillment of conditions of Proposition 4.4 for a large enough width limit $\mathcal{W}$, as stated in Proposition 4.5. The proof of this result is analogous to Proposition 4.3. The minimum width required for Proposition 4.5 is $\mathcal{O}(2^n)$, since it requires enumeration of all subsets of $V \backslash \{0\}$. Notice that it may be significantly larger than the pseudo-polynomial size for the capacity constraints in the m-PDTSP case.

Proposition 4.5. *For a sufficiently large $\mathcal{W}$, the procedure SplitNodesTour ensures that, for every node $u \in \mathcal{N}_t$, assumptions (i) and (ii) of Proposition 4.4 are satisfied.*

4.3.4 Bound Computation

If $\mathcal{M}$ is an exact MDD encoding Hamiltonian paths in a graph, Kinable et al. (2017) show that the minimum total travel cost, i.e., $\min\{c(\boldsymbol{x}) : \boldsymbol{x} \in \text{Sol}(\mathcal{M})\}$, can be found in polynomial time in the size of $\mathcal{M}$. To this end, the authors equip $\mathcal{M}$ with a more general travel cost matrix ζ, where $\zeta^t_{i,j}$ represents the cost of traveling from location i to location j when i is assigned to the t-th position of the tour, for

Algorithm 8 Expanding nodes based on tour and precedence.

1: **procedure** ExpandNodesTour($\mathcal{N}_t$, $\mathcal{W}$)
2: **while** $|\mathcal{N}_t| < \mathcal{W}$ and $\exists u \in S^L(\mathcal{N}_t)$ such that $|L_{\text{all}}(u)| < t$ **do**
3: Create a new node u', add it to $\mathcal{N}_t$.
4: Select any arc $a' \in \mathcal{A}^{\text{in}}(u)$.
5: Set $s(a) = u'$ for all $a \in \mathcal{A}^{\text{in}}_{\text{all}}(u, a')$, as defined in (4.6).
6: For every arc $a \in \mathcal{A}^{\text{out}}(u)$, create a new arc $a' = (u', t(a))$.
7: Update relaxed states for nodes u and u'.

$i, j \in V$ and $t \in \{0, \dots, n\}$. With such a matrix ζ, the cost $c(\boldsymbol{x})$ of a tour $\boldsymbol{x}$ becomes

$$c(\boldsymbol{x}) = \sum_{t=0}^{n} \zeta^t_{x_t, x_{t+1}}. \tag{4.7}$$

Note that $\zeta^t_{i,j} = c_{i,j}$ for all $t \in \{0, \dots, n\}$ in any given m-PDTSP instance, i.e., we could drop the additional index t. Nonetheless, we maintain this general cost representation when optimizing over $\mathcal{M}$, as it will be later directly applied to our Lagrangian dual in Section 4.4.2, where travel costs are also position dependent.

Let ℓ_a be the minimum cost of all partial tours encoded by paths starting at the root $\mathbf{r}$ and ending at an arc $a \in \mathcal{A}$. Such values are obtained using the recurrence

$$\ell_a := \begin{cases} 0, & \text{if } s(a) = \mathbf{r}, \\ \min_{a' \in \mathcal{A}^{\text{in}}(s(a))} \left\{ \ell_{a'} + \zeta^{t-1}_{v_{a'}, v_a} \right\}, & \text{otherwise.} \end{cases} \tag{4.8}$$

for each arc a with source node in layer $\mathcal{N}_t$. That is, for an arc a emanating from layer $\mathcal{N}_t$ $(t > 1)$, ℓ_a is the minimum cost among its possible predecessor locations, i.e., all locations $v_{a'}$ such that $a' \in \mathcal{A}^{\text{in}}(s(a))$, plus the cost to travel from the predecessor to the arc's location v_a. The optimal tour cost ν^* is, by definition,

$$h(\mathcal{M}) := \min_{a \in \mathcal{A}: t(a) = \mathbf{t}} \ell_a. \tag{4.9}$$

A proof of the validity of (4.8) is presented by Kinable et al. (2017) in the context of time-dependent sequencing. If $\mathcal{M}$ has a width of $w(\mathcal{M})$, all such values can be computed with a breadth-first search traversal in $\mathcal{O}(n|\mathcal{A}|w(\mathcal{M}))$.

If $\mathcal{M}$ is a relaxed MDD of arbitrary size, the value obtained in (4.9) provides instead a lower bound to the optimal solution value of the m-PDTSP. This follows since $\text{Sol}(\mathcal{M})$ over-approximates the set of feasible tours.

Example 4.4 Consider the exact MDD presented in Figure 4.2 (right). We use (4.8) to compute the arc costs $\ell_{(\mathbf{r},u_1)} = 0$, $\ell_{(u_1,u_2)} = \ell_{(\mathbf{r},u_1)} + \zeta^0_{v_{(\mathbf{r},u_1)}, v_{(u_1,u_2)}} = c_{0,1} = 447$, and analogously for the remaining arcs. In particular, the cost of arc (u_5, u_8) is given by

$$\begin{aligned} \ell_{(u_5,u_8)} &= \min\{\ell_{(u_2,u_5)} + \zeta^1_{v_{(u_2,u_5)}, v_{(u_5,u_8)}},\ \ell_{(u_3,u_5)} + \zeta^1_{v_{(u_3,u_5)}, v_{(u_5,u_8)}}\} \\ &= \min\{\ell_{(u_2,u_5)} + c_{3,2},\ \ell_{(u_3,u_5)} + c_{1,2}\} \\ &= \min\{1154 + 666,\ 1131 + 295\} = 1426. \end{aligned}$$

Similarly, we apply recursion (4.8) to compute the arc costs of the relaxed MDD shown in Figure 4.2 (left). In this case, the optimal tour cost (4.9) is given by $h(\mathcal{M}) = \min\{\ell_{(u_8,\mathbf{t})}\} = 1811$, which encodes the infeasible tour $\boldsymbol{x} = (0, 1, 4, 1, 4, 0)$. □

Notice that this procedure is an extension of the MDD bound for the sum-of-setup-times objective presented in Section 2.2.4. As such, the cost-based filtering rule CB-R2 is also valid for the m-PDTSP and can remove sub-optimal tours from $\text{Sol}(\mathcal{M})$.

4.4 An MDD-based Lagrangian Dual for the m-PDTSP

This section introduces the Lagrangian dual for the m-PDTSP that combines the ILP and the MDD formulations described in Sections 4.2 and 4.3, respectively. Our main purpose is to provide a new mechanism to obtain valid lower bounds to the problem. Such bounds can be used, e.g., to certify the quality of a feasible solution or to enhance a branch-and-bound search.

Our approach assumes that we are given a relaxed MDD $\mathcal{M}$ for an m-PDTSP instance (e.g., the one proposed in Section 4.3). Because $\mathcal{M}$ and a linear relaxation of $\mathcal{T}$ may be complementary in terms of the combinatorial structure each encodes, we wish to combine them into a single model that leverages the strengths of both formulations. To this end, we propose a Lagrangian dual that incorporates information from the LP relaxation of $\mathcal{T}$ as costs into $\mathcal{M}$, building on earlier works integrating DP and MDDs with Lagrangian relaxation (Beasley and Cao, 1998; Bergman et al., 2015b; Hooker, 2019).

In the remainder of this section, we first describe a model that integrates both relaxations to enhance bounds. Next, we show how such model can be addressed by solving its Lagrangian dual, which yields a subproblem that is polynomially solvable in $\mathcal{M}$.

4.4.1 Hybrid ILP-MDD Relaxation for the m-PDTSP

For ease of notation, let $A \in \mathbb{R}^{r\times|V|\times(n+2)}$ and $\boldsymbol{b} \in \mathbb{R}^r$ represent the matrix coefficients and right-hand side vector of the inequalities in $\mathcal{T}$, respectively, that only involve $\boldsymbol{y}$. Specifically,

$$\left\{\boldsymbol{y} \in \mathbb{R}^{|V|\times(n+2)} \,:\, A\boldsymbol{y} \leq \boldsymbol{b}\right\} = \left\{\boldsymbol{y} \in \mathbb{R}^{|V|\times(n+2)} \,:\, \boldsymbol{y} \text{ satisfies (4.2a)-(4.2e)}\right\},$$

assuming an appropriate dimension $r > 0$ encoding the number of constraints. In particular, an element $a_{l,i,t}$ of A is the coefficient of variable $y_{i,t}$ in the l-th constraint of $\mathcal{T}$. Notice that $A\boldsymbol{y} \leq \boldsymbol{b}$ models the matching, capacity, and precedence constraints of $\mathcal{T}$.

Furthermore, for a given exact or relaxed MDD $\mathcal{M}$, let $\text{Sol}_\gamma(\mathcal{M})$ be the set of binary solutions $(\boldsymbol{z}, \boldsymbol{y})$ that can be mapped to a tour encoded by $\mathcal{M}$. That is,

$$\text{Sol}_\gamma(\mathcal{M}) := \Bigg\{(\boldsymbol{z}, \boldsymbol{y}) \text{ binaries} \,:\, \exists\, \boldsymbol{x} \in \text{Sol}(\mathcal{M}) \text{ s.t. } \sum_{i=0}^{n} i\, y_{i,t} - x_t = 0 \text{ for all } t \in \{0, \dots, n+1\},$$
$$\text{and } (\boldsymbol{z}, \boldsymbol{y}) \text{ satisfies (4.2g)-(4.2h)}\Bigg\},$$

where the dimensions of $(\boldsymbol{z}, \boldsymbol{y})$ are as in $\mathcal{T}$, omitted above for exposition. Note that there is a one-to-one mapping between a vector pair $(\boldsymbol{z}, \boldsymbol{y}) \in \text{Sol}_\gamma(\mathcal{M})$ and a tour $\boldsymbol{x} \in \text{Sol}(\mathcal{M})$.

Let $\mathcal{M}$ be a relaxed MDD and denote the convex hull of a set $\mathcal{X}$ by $\text{conv}(\mathcal{X})$. We propose a new bound for the m-PDTSP obtained by solving the following hybrid relaxation $\mathcal{H}$:

$$\begin{aligned} \nu_R := \min_{\boldsymbol{z},\boldsymbol{y}} \quad & \sum_{t=0}^{n}\sum_{i=0}^{n}\sum_{j=0}^{n} c_{i,j}\, z_{i,j}^t \\ \text{s.t.} \quad & A\boldsymbol{y} \leq \boldsymbol{b}, \\ & (\boldsymbol{z}, \boldsymbol{y}) \in \text{conv}(\text{Sol}_\gamma(\mathcal{M})). \end{aligned} \tag{$\mathcal{H}$}$$

The optimal solution value of $\mathcal{H}$ is a lower bound to the original problem, i.e., $\nu^* \geq \nu_R$, since $\mathcal{H}$ is

the intersection of two over-approximations of the feasible tour set.

The choice of formulation $\mathcal{H}$ follows from three key motivations. First, since a convex hull can be equivalently described by a set of linear inequalities, problem $\mathcal{H}$ is a well-defined linear program that captures both the linear relaxation of $\mathcal{T}$ and the relaxation structure of $\mathcal{M}$. Second, using Lagrangian duality, $\mathcal{H}$ can be solved by exploiting an efficient combinatorial algorithm over $\mathcal{M}$. Finally, the bound provided by $\mathcal{H}$ is never worse than the original MDD bound or the LP relaxation of $\mathcal{T}$ when each is considered separately. As indicated by our numerical study, such a bound is often stronger and leads to significant speed-ups in our branch-and-bound search.

4.4.2 Solving $\mathcal{H}$ by Lagrangian Duality

We address $\mathcal{H}$ by dropping inequalities $A\boldsymbol{y} \leq \boldsymbol{b}$ and penalizing their violation in the objective function with Lagrange multipliers $\boldsymbol{\lambda}$. This yields a Lagrangian dual that observes strong duality with respect to $\mathcal{H}$. Namely, by Conforti et al. (2014), Theorem 8.2, we have that

$$\nu_R = \max_{\lambda} \{\mathcal{L}(\boldsymbol{\lambda}) \,:\, \boldsymbol{\lambda} \geq 0\}, \tag{$\mathcal{D}$}$$

where $\mathcal{L}(\cdot)$ is the *Lagrangian subproblem* defined as

$$\mathcal{L}(\boldsymbol{\lambda}) := \min_{\boldsymbol{z},\boldsymbol{y}} \left\{ \sum_{t=0}^{n}\sum_{i=0}^{n}\sum_{j=0}^{n} c_{i,j} z_{i,j}^t + \boldsymbol{\lambda}^\top (A\boldsymbol{y} - \boldsymbol{b}) \,:\, (\boldsymbol{z},\boldsymbol{y}) \in \mathrm{Sol}_\gamma(\mathcal{M}) \right\}.$$

Notice that $\mathcal{L}(\cdot)$ optimizes a linear function over $\mathrm{Sol}_\gamma(\mathcal{M})$ as opposed to the convex hull of such set. We now show that $\mathcal{L}(\cdot)$ is also tractable in the size of $\mathcal{M}$.

Proposition 4.6. *For any $\boldsymbol{\lambda} \geq 0$, the Lagrangian subproblem $\mathcal{L}(\boldsymbol{\lambda})$ can be solved in polynomial time in the size of $\mathcal{M}$. Specifically,*

$$\mathcal{L}(\boldsymbol{\lambda}) = h(\mathcal{M}) + \boldsymbol{\lambda}^\top \boldsymbol{b},$$

where $h(\mathcal{M})$ is defined as in (4.9) and computed using the MDD cost structure

$$\zeta_{i,j}^t = c_{i,j} + \sum_{l=1}^{r} (\lambda_l \, a_{l,i,t})$$

for all $i, j \in V$ and $t \in \{0, 1, \ldots, n\}$.

Proof. By definition, any $(\boldsymbol{z},\boldsymbol{y}) \in \mathrm{Sol}_\gamma(\mathcal{M})$ maps to a (unique) $\boldsymbol{x} \in \mathrm{Sol}(\mathcal{M})$. For such a triple $(\boldsymbol{z},\boldsymbol{y},\boldsymbol{x})$, recall that $\sum_{t=0}^{n}\sum_{i=0}^{n}\sum_{j=0}^{n} c_{i,j} z_{i,j}^t = c(\boldsymbol{x}) = \sum_{t=0}^{n} c_{x_t,x_{t+1}}$. Thus,

$$\begin{aligned}
\sum_{t=0}^{n}\sum_{i=0}^{n}\sum_{j=0}^{n} c_{i,j} z_{i,j}^t + \boldsymbol{\lambda}^\top (A\boldsymbol{y} - \boldsymbol{b}) &= \sum_{t=0}^{n} c_{x_t,x_{t+1}} + \sum_{l=1}^{r} \lambda_l \left(\sum_{i=0}^{n}\sum_{t=0}^{n} a_{l,i,t} y_{i,t} - b_l \right) \\
&= \sum_{t=0}^{n} c_{x_t,x_{t+1}} + \sum_{t=0}^{n}\sum_{i=0}^{n} \left(\sum_{l=1}^{r} \lambda_l a_{l,i,t} \right) y_{i,t} - \boldsymbol{\lambda}^\top \boldsymbol{b} \\
&= \sum_{t=0}^{n} \left(c_{x_t\, x_{t+1}} + \sum_{l=1}^{r}\sum_{i=0}^{n} (\lambda_l a_{l,i,t}) y_{i,t} \right) - \boldsymbol{\lambda}^\top \boldsymbol{b}
\end{aligned}$$

$$= \sum_{t=0}^{n} \left(c_{x_t,x_{t+1}} + \sum_{l=1}^{r} (\lambda_l a_{l,x_t,t}) \right) - \boldsymbol{\lambda}^\top \boldsymbol{b}.$$

This implies that $\sum_{t=0}^{n}\sum_{i=0}^{n}\sum_{j=0}^{n} c_{i,j} z_{i,j}^t + \boldsymbol{\lambda}^\top A\boldsymbol{y} = \sum_{t=0}^{n} \zeta_{x_t,x_{t+1}}^t$ for costs $\zeta_{x_t,x_{t+1}}^t$ of the same form (4.7) as required for $h(\mathcal{M})$. ■

4.4.3 Incorporating the Lagrange Multipliers into $\mathcal{M}$

We now provide further details on how to incorporate the Lagrange multipliers associated with the linear system $A\boldsymbol{y} \leq \boldsymbol{b}$ into a relaxed MDD $\mathcal{M}$ by means of Proposition 4.6. Our implementation focuses on the Lagrange multipliers related to the tour equalities (4.2b), the capacity inequalities (4.2c)-(4.2d), and the precedence inequalities (4.2e). Notice that we are not required to consider inequalities (4.2a) and (4.2f) since they are enforced by construction of $\mathcal{M}$. Other valid linear inequalities to the m-PDTSP, however, can be incorporated analogously.

Consider the Lagrange multipliers $\boldsymbol{\lambda} = (\boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\sigma})$, where $\boldsymbol{\beta} \in \mathbb{R}^{|V|}$, $\boldsymbol{\mu} \in \mathbb{R}^{2n}$ ($\boldsymbol{\mu} \geq 0$), and $\boldsymbol{\sigma} \in \mathbb{R}^{|K|}$ ($\boldsymbol{\sigma} \geq 0$) are associated with constraints (4.2b), (4.2c)-(4.2d), and (4.2e), respectively. For this set of multipliers, the cost matrix ζ of Proposition 4.6 is given by

$$\zeta_{i,j}^t = c_{i,j} + \quad \beta_i \quad + \quad \Delta q_i \sum_{t'=t}^{n} (\mu_{n+t'} - \mu_{t'}) \quad + \quad t \left(\sum_{\{k \in K : p_k = i\}} \sigma_k - \sum_{\{k \in K : d_k = i\}} \sigma_k \right).$$

Notice that the constant $\boldsymbol{\lambda}^\top \boldsymbol{b}$ is

$$\boldsymbol{\lambda}^\top \boldsymbol{b} = -\sum_{i \in V} \beta_i - \sum_{t=n+1}^{2n} C\mu_t + \sum_{k \in K} \sigma_k.$$

The cost matrix ζ is used in recurrence (4.8)-(4.9) to compute a new lower bound over $\mathcal{M}$, as illustrated in the example below. Any valid set of multipliers suffice to obtain a valid lower bound for the m-PDTSP. In particular, the strongest bound is at least as strong as the one obtained from the original relaxed MDD $\mathcal{M}$ (Fisher, 2004).

Example 4.5 Consider our running example and the relaxed MDD $\mathcal{M}$ shown in Figure 4.2 (right). Suppose, for illustration purposes, that we incorporate only equalities (4.2b) and let $\boldsymbol{\beta} \in \mathbb{R}^{|V|}$ be the vector of Lagrange multipliers associated with (4.2b). If we set $\beta_1 = \beta_4 = 100$ and $\beta_i = 0$ for all $i \in V \setminus \{1, 4\}$, we obtain $\mathcal{L}(\boldsymbol{\lambda}) = h(\mathcal{M}) + \boldsymbol{\lambda}^\top \boldsymbol{b} = 1863$. This solution corresponds to the optimal tour $\boldsymbol{x} = (0, 3, 1, 2, 4, 0)$ and improves the original MDD bound of 1811. □

4.4.4 Solution Method for the Lagrangian Dual

Problem $\mathcal{D}$ is the *Lagrangian dual* problem, a maximization problem with a piecewise linear concave objective (Fisher, 2004). It can be solved iteratively by computing $\mathcal{L}(\boldsymbol{\lambda}^0)$ for some $\boldsymbol{\lambda}^0 \geq 0$, obtaining a new set of Lagrange multipliers $\boldsymbol{\lambda}^1$ based on the solution of $\mathcal{L}(\boldsymbol{\lambda}^0)$, and repeating until some termination criteria is reached. The function $\mathcal{L}(\cdot)$ can be computed efficiently in $\mathcal{O}(n|\mathcal{A}|w(\mathcal{M}))$ as described in Sections 4.4.2 and 4.4.3.

For the update of the Lagrange multipliers, we apply Bundle methods that have a relatively fast convergence rate in comparison to other methods; i.e., typically bounded by $\mathcal{O}(1/\epsilon^3)$ for a given precision

ϵ (Lemaréchal, 1975). A Bundle method is a variant of the cutting plane method, which adds a quadratic stabilizer to improve convergence. The method iteratively solves $\mathcal{L}(\boldsymbol{\lambda})$ until it converges to the optimal set of multipliers $\boldsymbol{\lambda}^*$. Each optimal solution $(\boldsymbol{z}', \boldsymbol{y}')$ of $\mathcal{L}(\boldsymbol{\lambda})$ has an associated subgradient, $A\boldsymbol{y}' - \boldsymbol{b}$, that is used to generate a cutting plane that is valid for the function $\mathcal{L}(\cdot)$.

For the specific case of the m-PDTSP, consider the k-th iteration of the procedure with $\boldsymbol{\lambda}^k$ as the current set of multipliers associated with the system $A\boldsymbol{y} \leq \boldsymbol{b}$. Our implementation solves $\mathcal{L}(\boldsymbol{\lambda}^k)$ using the recursive arc cost procedure (i.e., (4.8)-(4.9)) over $\mathcal{M}$ with the cost structure shown in Proposition 4.6. Solution $\boldsymbol{x}^k = \arg\max\{h(\mathcal{M})\}$ is mapped to $(\boldsymbol{z}^k, \boldsymbol{y}^k)$ for the subgradient computation, $A\boldsymbol{y}^k - \boldsymbol{b}$. The method then solves the following quadratic problem to generate a new set of multipliers $\boldsymbol{\lambda}^{k+1}$,

$$\boldsymbol{\lambda}^{k+1} = \arg\max_{w,\boldsymbol{\lambda}} \left\{ w + \frac{1}{2t}||\boldsymbol{\lambda} - \boldsymbol{\lambda}^k|| : w \leq \mathcal{L}(\boldsymbol{\lambda}^s) + (A\boldsymbol{y}^s - \boldsymbol{b})^\top(\boldsymbol{\lambda} - \boldsymbol{\lambda}^s),\ \forall s \in \{0, \ldots, k\}, w \in \mathbb{R},\ \boldsymbol{\lambda} \in \mathbb{R}^r_+ \right\}.$$

In the problem above, w is a variable that over approximates the Lagrangian dual bound, and $\boldsymbol{\lambda}$ corresponds to the set of Lagrange multipliers. The objective function includes a quadratic stabilizer $\frac{1}{2t}||\boldsymbol{\lambda} - \boldsymbol{\lambda}^k||$ $(t < 1)$ to improve convergence (Lemaréchal, 1975). In each iteration of the procedure, the set of constraints increases by one, where each new constraint is a cutting plane derived from the subgradients in the previous iterations.

4.5 Overall Solution Approach

Our complete solution approach to the m-PDTSP uses the Lagrangian dual $\mathcal{D}$ as a bounding mechanism in a branch-and-bound procedure. The approach exploits the graphical structure of $\mathcal{M}$ to branch in sequential order with respect to the layers in $\mathcal{M}$.

We first build a relaxed MDD of maximum width $\mathcal{W}$ using the construction procedure described in Section 4.3. The Lagrangian dual is then solved to optimality (Section 4.4). We then perform a depth-first search by branching on the $\boldsymbol{x}$ variables in this sequence. Each branching decision fixes to either $x_t = i$ or $x_t \neq i$, which is equivalent to a binary branching over $\boldsymbol{y}$. Given a variable x_t to branch on, we choose the location i that is part of a shortest $\mathbf{r} - \mathbf{t}$ path (4.8) (ties are broken according to a lexicographic order).

When fixing $x_t = i$, we update $\mathcal{M}$ by removing infeasible arcs, re-applying the expansion method, and recomputing the Lagrangian dual objective function using the optimal multipliers from the root node (i.e., we never resolve $\mathcal{D}$ to optimality). This provides us a new lower bound that is used to prune nodes according to the best feasible solution found during search. We do not consider any additional primal heuristics.

We implement two variants of this methodology. The first, denoted by $\mathcal{M}^{\mathcal{C}}$, builds a relaxed MDD using SplitNodesCapacity (Section 4.3.2) first. If the capacity constraints can be represented exactly with a smaller width (i.e., when $\mathcal{W} > C + 1$), we apply the SplitNodesTour until the maximum width is met. The second implementation, denoted by $\mathcal{M}^{\mathcal{T}}$, inverts the order of the expansion procedures, i.e., SplitNodesTour is performed before SplitNodesCapacity. We will investigate the impact of dualizing the different inequalities (4.2b)-(4.2e) over $\mathcal{M}^{\mathcal{T}}$ and $\mathcal{M}^{\mathcal{C}}$.

4.6 Constraint Programming Formulation

As a baseline to our approach, we formulate a constraint programming model for the m-PDTSP using IBM ILOG CP Optimizer 12.9 (IBM, 2019) notation. We model the m-PDTSP as a sequential problem where the variables represent the time in which each location is visited and the traveling cost represents the traveling time between locations.

Formally, the model considers a set of $n+2$ interval variables and a sequence variable. We consider an interval variable I_i for each location $i \in V \setminus \{0\}$ that represents the time window in which the vehicle visits location i, i.e., $\texttt{StartOf}(I_i)$ and $\texttt{EndOf}(I_i)$ correspond to the arrival and departure time to location i, respectively. We also include two interval variables to represent the start and end at the depot, I_0 and I_{n+1}, respectively. Since the problem does not consider the time spent at each location, the duration is fixed to one and that value is then discounted in the objective function. The sequence variable π is defined over the set of interval variables and represents the sequence of locations to visit. Our constraint programming model ($\mathcal{CP}$) is as follows:

$$
\begin{aligned}
\min \quad & \texttt{StartOf}(I_{n+1}) - (n+2) && && (\mathcal{CP})\\
\text{s.t.} \quad & \texttt{NoOverlap}(\pi, \{c_{i,j} : (i,j) \in E\}), && && (4.10a)\\
& \texttt{Before}(\pi, I_{p_k}, I_{d_k}), && \forall k \in K, && (4.10b)\\
& \sum_{i \in V} \texttt{StepAtStart}(I_i, \Delta q_i) \leq C, && && (4.10c)\\
& \texttt{First}(\pi, I_0), && && (4.10d)\\
& \texttt{Last}(\pi, I_{n+1}), && && (4.10e)\\
& I_i \ : \ \texttt{intervalVar}(0, UB), && i \in \{0, ..., n+1\}, && (4.10f)\\
& \pi \ : \ \texttt{sequenceVar}(\{I_0, \ldots, I_{n+1}\}). && && (4.10g)
\end{aligned}
$$

The objective function minimizes the time until the return to the depot, omitting the time spend in each location. Constraint (4.10a) enforces the traveling time between locations. Constraint (4.10b) imposes the precedence condition for each commodity. Constraint (4.10c) defines the load on the vehicle and enforces it to satisfy the capacity restriction. Conditions (4.10d) and (4.10e) impose the sequence to start and end at the depot, respectively. Lastly, (4.10f) and (4.10g) define the set of variables, where UB is an upper bound on the objective function (e.g., sum of all traveling distances).

Our CP model implementation considers a search phase over the sequential variable. Thus, we ask the search engine to first fix the order of the intervals inside the sequential variable π before branching on the start and end times of each interval variable. Preliminary experiments showed that this search strategy achieved better performance than the default CP Optimizer search.

4.7 Numerical Study

This section describes the experimental analysis for our proposed MDD-based Lagrangian approach. We use the benchmark of 1,178 instances developed by Hernández-Pérez and Salazar-González (2009), which is divided into three classes. Class 1 (248 instances) is a set of modified SOP problems introduced by Ascheuer et al. (2000), where each precedence relation in the original instance is associated with a commodity. The class is divided into two groups that differ on the commodity weights: *max1* for unitary

weights (i.e., $q_k = 1$ for all $k \in K$), and *max5* for discrete weights up to 5 units (i.e., $q_k \in \{1, \dots, 5\}$ for all $k \in K$). Class 2 (720 instances) and Class 3 (210 instances) are generated by placing locations on a grid uniformly at random and considering the Euclidean distance between locations as traveling costs. The weights in these two classes are also generated uniformly at random from the set $\{1, \dots, 5\}$. Instances of Class 2 have no restriction on the number of commodities that each location can supply or demand. Instances of Class 3 have $m = n/2$ commodities and each location is either a pickup or a delivery spot for exactly one commodity.

All experiments use a maximum width $\mathcal{W} = 1,024$ and solve the Lagrangian dual $\mathcal{D}$ using the proximal bundle method implemented by Frangioni (2002) in `C++`, kindly provided by the author, using a specialized single-thread quadratic programming solver. The MDD construction was implemented within the constraint solver ILOG CP Optimizer 12.9 (IBM, 2019), which was used only for the purpose of handling the depth-first search bookkeeping, i.e., we disabled all constraint propagation and additional features of the solver.

The experiments were run on an Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50GHz with 16 GB RAM considering a time limit of 2 hours (7,200 seconds) and a single core (parameter `Workers` = 1). The MDD-related times account for both the Lagrangian dual solution times and the actual search time.

The experimental evaluation is organized as follows. Section 4.7.1 presents a summary of the results for our MDD-based Lagrangian procedures and compares them to our CP model and state-of-the-art MILP techniques. Section 4.7.2 analyzes the quality of the MDD bounds enhanced with Lagrangian penalties. Section 4.7.3 presents a comparison between our two MDD compilations, $\mathcal{M}^{\mathcal{T}}$ and $\mathcal{M}^{\mathcal{C}}$, and draws conclusions on which procedure to used based on the problem characteristics. Lastly, Section 4.7.4 compares the MDD-based Lagrangian procedures to $\mathcal{CP}$ in terms of search effort.

4.7.1 Overall Comparison with State-of-the-Art Techniques

We now present the performance of our MDD-based Lagrangian procedure over the 1,178 instances in the literature (Hernández-Pérez and Salazar-González, 2009). We consider the two best-performing MDD variants: the capacity-based MDD $\mathcal{M}^{\mathcal{C}}_{\beta}$ and the tour-based MDD $\mathcal{M}^{\mathcal{T}}_{\beta}$, both techniques strengthened with the tour inequalities (4.2b) within our Lagrangian dual framework. To evaluate the robustness of the methodologies, we compare these techniques with $\mathcal{M}^{\mathcal{T}}$ and $\mathcal{M}^{\mathcal{C}}$ (i.e., both MDD construction procedures without Lagrange multipliers) and with our CP model $\mathcal{CP}$. Detailed results for all instances and techniques can be found in Appendix A.

Table 4.1 presents a summary of the number of instances solved to optimality and proved infeasible for each class. The table shows that $\mathcal{M}^{\mathcal{T}}_{\beta}$ and $\mathcal{M}^{\mathcal{C}}_{\beta}$ outperform $\mathcal{CP}$ in all instances classes, both when proving optimality and infeasibility. We also see that the MDD-based Lagrangian techniques outperform the pure discrete optimization alternative, e.g., $\mathcal{M}^{\mathcal{T}}_{\beta}$ solves 97 more instances to optimality than $\mathcal{M}^{\mathcal{T}}$. Lastly, we highlight the performance difference between our two MDD compilations, which we analyze in Section 4.7.3.

We also compare our methodologies with state-of-the-art techniques in the literature. We consider the Benders decomposition $\mathcal{BE}$ by Hernández-Pérez and Salazar-González (2009), and the branch-and-cut algorithm by Gouveia and Ruthmair (2015), $CUTR^*$, which we will refer to as $\mathcal{CU}$. Due to the lack of results presented in previous papers, we restrict our comparison to a subset of 527 feasible instances. Notice that the results for $\mathcal{BE}$ were obtained using CPLEX 10.2 and a personal computer with Intel Pentium 3.0 Ghz, while the $\mathcal{CU}$ results used CPLEX 12.6 with an Intel Xeon E5540 machine with 2.53

Table 4.1: Number of instances solved for our techniques over the complete dataset.

Status	Class	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$
Optimality	Class 1	110	120	120	**210**	150
	Class 2	611	**634**	**634**	**634**	**634**
	Class 3	201	201	195	202	**208**
	Total	922	955	949	**1046**	992
Infeasibility	Class 1	6	**20**	**20**	**20**	**20**
	Class 2	73	**86**	**86**	**86**	**86**
	Class 3	0	0	0	0	0
	Total	79	**106**	**106**	**106**	**106**

GHz.

Table 4.2: Total number of instances solved to optimality per class.

	# instances	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$ & $\mathcal{M}^{\mathcal{C}}_{\beta}$
Class 1	36	24	26	13	22	22	**35**	30	**35**
Class 2	341	314	330	320	**341**	**341**	**341**	**341**	**341**
Class 3	150	134	136	141	141	136	142	**148**	**149**
Total	527	472	492	474	504	499	518	**519**	**525**

Table 4.2 presents the number of instances solved to optimality for this reduced dataset. Both MDD-based Lagrangian techniques solve the same instances as the MILP-based approaches, in addition to several open instances in the literature. $\mathcal{M}^{\mathcal{C}}_{\beta}$ closes 27 open instances, while $\mathcal{M}^{\mathcal{T}}_{\beta}$ closes 26. If we consider the total number of instances solved by $\mathcal{M}^{\mathcal{C}}_{\beta}$ and $\mathcal{M}^{\mathcal{T}}_{\beta}$ together, we were able to prove optimality for 33 open instances for the first time. We also highlight that $\mathcal{CP}$ has a competitive performance when compared to the MILP methodologies despite the fact that it is an "out-of-the-box" model without the substantial reformulation and algorithmic effort that has gone into the other models.

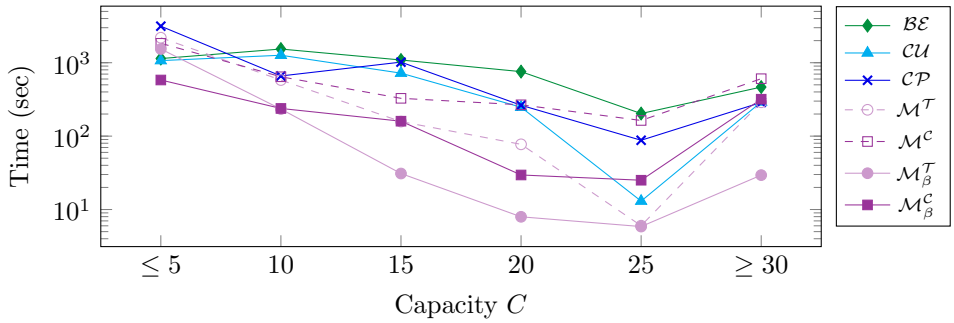


Figure 4.4: Average run time comparison.

Figure 4.4 compares the average run times for all techniques. The instances are divided according to the capacity restriction C as follows: $\{C \leq 5, C = 10, C = 15, C = 20, C = 25, C \geq 30\}$. The plot shows that our MDD-based Lagrangian approaches outperform $\mathcal{CP}$ and the MILP techniques for the different capacity limits. The figure illustrates that the pure discrete relaxation techniques, $\mathcal{M}^{\mathcal{C}}$ and $\mathcal{M}^{\mathcal{T}}$, are slower on average than their Lagrangian dual counterparts. We also see that $\mathcal{M}^{\mathcal{C}}_{\beta}$ has the lowest average

run time when the capacity is small (i.e., $C \leq 10$), while $\mathcal{M}^{\mathcal{T}}_{\beta}$ is the fastest technique on instances with looser capacity restrictions (i.e., $C \geq 15$).

4.7.2 MDD Relaxation Analysis

We now investigate incorporating different subsets of inequalities from $\mathcal{T}$ into our MDD as Lagrangian penalties. As introduced in Section 4.4.3, we use $\boldsymbol{\beta}$, $\boldsymbol{\mu}$, and $\boldsymbol{\sigma}$ to represent the Lagrange multipliers related to the tour (4.2b), capacity (4.2c)-(4.2d), and precedence (4.2e) constraints, respectively; e.g., $\mathcal{M}_{\beta}$ corresponds to relaxing inequality (4.2b). We note that, in all cases, solving the Lagrangian dual takes less than 2 minutes. The experiments here only consider the tour-based MDD, $\mathcal{M}^{\mathcal{T}}$, as similar results were obtained with the capacity-based MDD.

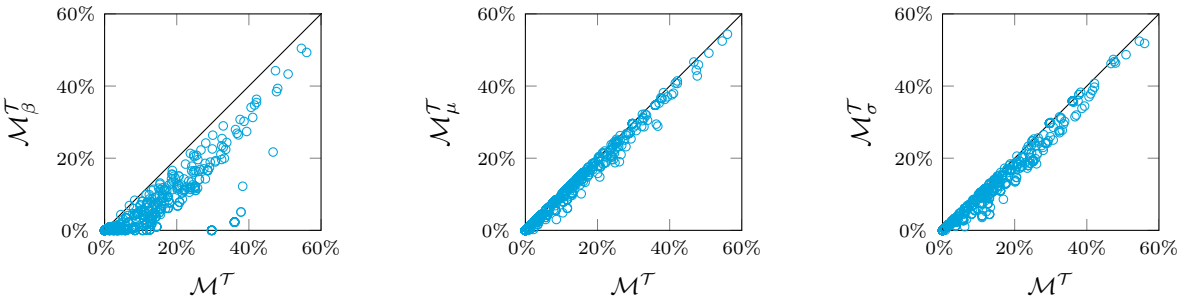


Figure 4.5: Optimality gap comparison at the root node.

We start by analyzing the quality of the bound computed at the root node, i.e., the optimal solution to the Lagrangian dual problem. To this end, we compute the optimality gap for each relaxation as $gap = (opt - LB)/opt$, where LB is the lower bound and opt the optimal value. We then compare the gap obtained by each Lagrangian relaxation with the gap produced by $\mathcal{M}^{\mathcal{T}}$ with no Lagrange multipliers.

Figures 4.5 depicts the gap improvement when the tour constraints (4.2b) (left plot), capacity constraints (4.2c)-(4.2d) (middle plot), and precedence constraints (4.2e) (right plot) are considered in the Lagrangian dual problem. In each plot, a point represents an instance, its x-coordinate the gap computed by $\mathcal{M}^{\mathcal{T}}$, and the y-coordinate the gap obtained by solving the Lagrangian dual problem. Points below the diagonal are instances where the Lagrangian gap is smaller. Figure 4.5 shows that relaxations based on the tour constraints obtain the greatest optimality gap reduction (left plot). In contrast, $\mathcal{M}^{\mathcal{T}}_{\mu}$ and $\mathcal{M}^{\mathcal{T}}_{\sigma}$ slightly improve the bound quality when compare to $\mathcal{M}^{\mathcal{T}}$.

Table 4.3: Number of instances solved for different Lagrangian relaxations.

Status	Instances	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\mu}$	$\mathcal{M}^{\mathcal{T}}_{\sigma}$	$\mathcal{M}^{\mathcal{T}}_{\beta+\mu}$	$\mathcal{M}^{\mathcal{T}}_{\beta+\sigma}$
Optimality	Class 1	120	**210**	120	120	**210**	**210**
	Class 2	**634**	**634**	**634**	**634**	**634**	**634**
	Class 3	201	**202**	**202**	201	**202**	**202**
	Total	955	**1046**	956	955	**1046**	**1046**
Infeasibility	Class 1	20	20	20	20	20	20
	Class 2	86	86	86	86	86	86
	Class 3	0	0	0	0	0	0
	Total	106	106	106	106	106	106

These results directly correlate with the number of instances solved, as shown in Table 4.3. The table presents the number of instances solved to optimality and the number of instances proven to be infeasible for different MDD-based Lagrangian alternatives. Here we can see that $\mathcal{M}^{\mathcal{T}}_{\beta}$ solves 93 more instances to optimality than $\mathcal{M}^{\mathcal{T}}_{\mu}$ and $\mathcal{M}^{\mathcal{T}}_{\sigma}$, while these last two techniques solve no more than a couple of additional instances when compare to $\mathcal{M}^{\mathcal{T}}$. We also tested other combinations of multipliers with no substantial improvements. For instance, relaxations including tour-based multipliers with any of the two others multipliers (i.e., $\mathcal{M}^{\mathcal{T}}_{\beta+\mu}$ and $\mathcal{M}^{\mathcal{T}}_{\beta+\sigma}$) had only a small optimality gap reduction and no difference in terms of instances solved when compare to $\mathcal{M}^{\mathcal{T}}_{\beta}$ (see Table 4.3).

4.7.3 MDD Construction Analysis

This section analyzes our two MDD construction procedures (i.e., $\mathcal{M}^{\mathcal{T}}$ and $\mathcal{M}^{\mathcal{C}}$) for different problem characteristics. As previously mentioned, the capacity restriction is one of the main characteristics for problem difficulty for the m-PDTSP (Letchford and Salazar-González, 2016). As such, the aim of this section is to verify that our capacity based compilation $\mathcal{M}^{\mathcal{C}}$ is more suitable for solving instances with tight vehicle capacity than the tour based construction $\mathcal{M}^{\mathcal{T}}$. Our analysis classifies each instance into three categories based on its vehicle capacity: small C (i.e., $C \leq 10$), medium C (i.e., $10 < C \leq 20$), and large C (i.e., $C > 20$).

Figure 4.6 illustrates two plots comparing the gap for our two MDD compilations, considering the pure discrete relaxation (left plot) and the best Lagrangian dual alternative (right plot). Each (x, y) point in the plots corresponds to an instance from the dataset, where the x and y values are the gaps for the $\mathcal{M}^{\mathcal{T}}$ and $\mathcal{M}^{\mathcal{C}}$ MDD relaxations, respectively. Both plots show that the majority of the small C instances are below the diagonal and that a large portion of the instances with medium or large C are above the diagonal. This result illustrates that the our capacity-based compilation usually achieves better bounds for problems with tight vehicle capacity. We also highlight that this difference is more predominant when our MDDs include the Lagrange penalties (right plot).

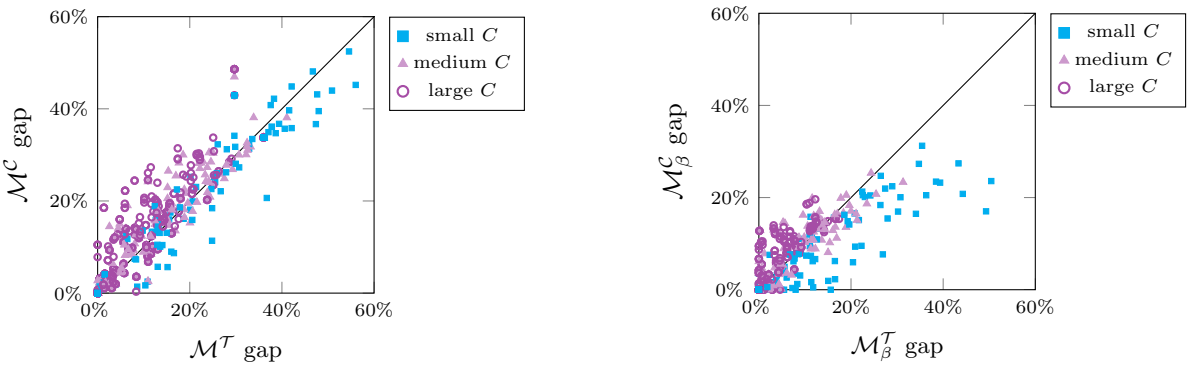


Figure 4.6: Optimality gap comparison for MDD construction procedures.

We observed a similar behavior when we compare the solving time between our two MDD construction procedures. Figure 4.7 shows two plots comparing the solving time between the MDD construction procedures, once again considering the pure discrete relaxation (left plot) and the Lagrangian dual (right plot). Each point is a single instance and the x and y values correspond to the run time for each technique on their respective axes. The capacity based construction tends to have smaller solving times

over instances with small C. Also, notice the vertical line of small C points in the top right corner of each plot, which represents instances that the $\mathcal{M}^{\mathcal{C}}$ compilation solves and $\mathcal{M}^{\mathcal{T}}$ times out.

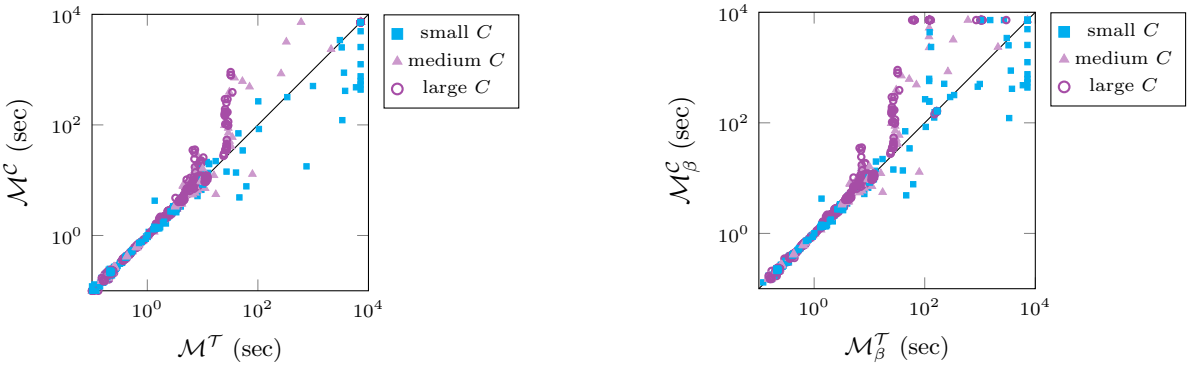


Figure 4.7: Time comparison for MDD construction procedures. Plots in logarithmic scale.

4.7.4 MDD and CP Search Effort

We end the numerical study of this chapter by comparing the search effort of our best MDD-based Lagrangian technique $\mathcal{M}^{\mathcal{T}}_{\beta}$ and our CP model $\mathcal{CP}$. Recall that our MDD-based approaches are implemented inside the IBM ILOG CP Optimizer solver, thus, comparing our MDD techniques to the CP Optimizer global constraints used in $\mathcal{CP}$ gives us an idea of the strength of the MDD inference.

Figure 4.8 compares the search effort of these two techniques in terms of run time (left plot) and number of branches (right plot) during search. Each point corresponds to a specific instance and, as in Section 4.7.3, we classify the problems in terms of their vehicle capacity. The left plot shows that $\mathcal{M}^{\mathcal{T}}_{\beta}$ and $\mathcal{CP}$ have competitive solving time for the easier instances (i.e., less than 100 seconds to be solved) but the difference in solving time increases for the harder instances. In particular, we see that $\mathcal{CP}$ struggles to solve problems with a small capacity limit. In these instances, $\mathcal{CP}$ has a hard time finding feasible solutions and, thus, the propagation of its constraints is quite limited.

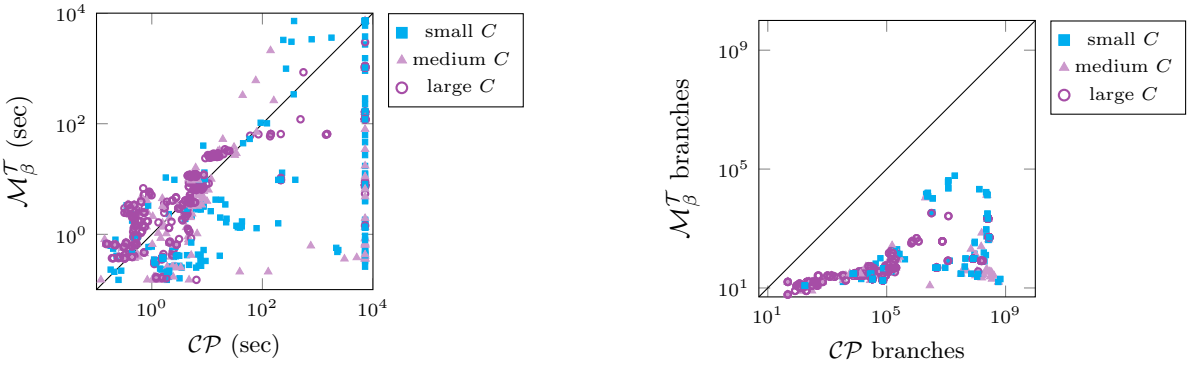


Figure 4.8: Time (left) and branches (right) comparison between $\mathcal{M}^{\mathcal{T}}_{\beta}$ and $\mathcal{CP}$. Plots in logarithmic scale.

The right plot in Figure 4.8 shows the number of branches explored for each technique during search. We can see that all the points are below the diagonal, which illustrates the small search effort of $\mathcal{M}^{\mathcal{T}}_{\beta}$.

Thus, the inference and bounding capabilities of $\mathcal{M}_\beta^\mathcal{T}$ are superior to the ones for the CP Optimizer global constraints. However, the propagation of $\mathcal{M}_\beta^\mathcal{T}$ inside the CP solver is quite slow, limiting the number of branches that this technique can explore before the time limit. This result explains why $\mathcal{CP}$ has a similar solving time to $\mathcal{M}_\beta^\mathcal{T}$ is several instances, while exploring orders of magnitude more branches than $\mathcal{M}_\beta^\mathcal{T}$. Thus, improving the construction and propagation algorithms of $\mathcal{M}_\beta^\mathcal{T}$ could have a positive impact when tackling the most challenging instances of the problem.

4.8 Conclusions

We presented a novel approach to tackle the m-PDTSP, a challenging problem from the vehicle routing literature. The approach considers a discrete relaxation, encoded as a relaxed MDD, to better represent the combinatorial structure of the problem. We used Lagrangian duality to combine the discrete relaxation with a linear representation of the problem. Overall, the technique closes 33 instances in the literature, whereas our best implementation closes 27 of those instances.

The Lagrangian dual plays a key role on improving the bound quality and the coverage of our technique. In addition, it provides valuable insight on the quality of our discrete relaxation. Namely, the fact that relaxing the tour inequalities considerably reduces the optimality gap indicates that these constraints are often violated in the relaxed MDD paths. Therefore, efforts to generate stronger relaxed MDDs for sequence problems should emphasize this restriction.

Nevertheless, the tightness of the capacity constraint is still a key component of the m-PDTSP that can considerably increase the search effort. Our results show that this can be partially addressed using a construction based on the capacity constraint. In fact, our capacity-based MDD construction solves some of the most challenging capacity restricted problems. In contrast, the tour construction is stronger for problems with a higher number of locations and a larger capacity.

This work emphasizes the value of exploiting a discrete relaxation for problems with a complex combinatorial structure, such as the m-PDTSP, alongside valid linear relaxations. This extends the use of MDDs to solve sequencing problems with capacity restrictions by presenting new construction and filtering strategies. Possible extensions of this work include single-commodity and time windows pickup-and-delivery problems, which can be naturally incorporated into this framework.

Chapter 5

Delete-Free AI Planning

Automated planning is an area of Artificial Intelligence (AI) that looks for a sequence of operations that an autonomous agent needs to follow to achieve a set of goals (Ghallab et al., 2004). Among its many sub-areas, cost-optimal classical AI planning is one of the most well-studied in the field (Helmert and Domshlak, 2009; Pommerening et al., 2014). A classical AI planning task seeks a minimum-cost sequence of operators that lead from an initial state to a state where all the goals are satisfied. In this context, states are represented by propositional facts and operators move the agent from one state to another by changing the facts values to either true (i.e., additive effects) or false (i.e., delete effects). This problem is usually solved using heuristic search algorithms, such as A^* (Hart et al., 1968), where the search procedure is guided by an admissible heuristic (i.e., a valid lower bound). Building admissible heuristics from relaxations of the original problem is a common approach in the literature (Haslum et al., 2005; Helmert et al., 2008).

This chapter focuses on *cost-optimal delete-free planning (DFP)*, a variant of classical AI planning that ignores the delete effects of operators. This is an NP-hard problem (Bylander, 1994) that has been extensively investigated by the planning community as the basis for efficient methodologies to address classical AI planning problems (Betz and Helmert, 2009; Helmert and Domshlak, 2009). Moreover, some challenging planning problems can be formulated as delete-free tasks, such as the minimal seed set problem (Gefen and Brafman, 2011).

We explore the effectiveness of relaxed decision diagrams (DDs) for solving DFP tasks. We introduce a new family of admissible heuristics based on relaxed DDs and, thus, propose a new path for heuristic development in the field. As a first step, this work shows the potential of relaxed DD heuristics over DFP tasks. Nonetheless, these heuristics can be used in more challenging planning variants with minor modifications to the proposed implementation, e.g., inside cost-optimal classical AI planners. We also relate the DD heuristics to well-known heuristics in the planning community (i.e., critical path and disjunctive landmarks), opening new research avenues using relaxed DDs to combine and create new admissible heuristics. Moreover, we introduce a flexible DD construction procedure based on node information that can be extended to consider, e.g., numerical variables.

Main contributions. We present a multivalued decision diagram (MDD) encoding of a DFP task and a binary decision diagram (BDD) representation of the DFP sequential relaxation. We investigate the structural properties of each graphical model and present a numerical comparison with the DFP linear

programming (LP) relaxation (Imai and Fukunaga, 2014, 2015). Specifically, we propose construction procedures for each graphical structure that guarantee the admissibility and consistency of their resulting heuristics. We explore the theoretical properties of our relaxed DDs and relate their heuristics to existing techniques in the literature. Furthermore, we show how to leverage these graphical structures to identify landmarks and redundant operators, and to extract delete-free plans.

This chapter includes an extensive empirical analysis that highlights the advantages and disadvantages of using relaxed DD-based heuristics in place of the LP relaxation. We show that, even with small relaxed DDs, our heuristics have competitive performance on DFP tasks. Our relaxed MDD and BDD approaches outperform the mixed-integer linear programming (MILP) model (Imai and Fukunaga, 2015) in four delete-free IPC domains. However, the MILP model remains the state-of-the-art for the majority if DFP tasks.

This work led to a conference paper at *The International Conference on Automated Planning and Scheduling* (Castro et al., 2019) and a journal publication in the *Journal of Artificial Intelligence Research* (Castro et al., 2020c).

Outline. The rest of the chapter is as follows. Section 5.1 describes DFP and its sequential relaxation. Section 5.2 presents a brief literature review on DFP tasks. Section 5.3 introduces our relaxed MDD encoding and construction procedure, while Section 5.4 presents the relaxed MDD heuristic, its theoretical properties and relationship to existing techniques. Similarly, Section 5.5 describes the relaxed BDD encoding for the sequential relaxation and Section 5.6 discusses the theoretical properties of the heuristic. Section 5.7 highlights the main differences between the graphical structures and provides guidance on when each approach is more appropriate. Section 5.8 presents alternative uses of relaxed DDs beyond heuristic computation. Section 5.9 explains our planner implementation and Section 5.10 presents an empirical analysis of our proposed methods. The chapter ends with concluding remarks and potential research directions.

5.1 Problem Definition

This work considers cost-optimal DFP using the STRIPS formalism (Fikes and Nilsson, 1971) restricted to tasks with no negative preconditions and no conditional effects. A DFP task is given by a tuple $\Pi^+ = \langle F, \mathsf{s}_I, G, O \rangle$ where F corresponds to the set of facts, s_I is the initial state, $G \subseteq F$ is the set of goals, and O is the set of operators. Each fact $f \in F$ can be either true or false. We define a state s by its set of true facts, i.e., we say that f is true in s if $f \in \mathsf{s}$ and false otherwise.

An operator $o \in O$ is a tuple $\langle \mathsf{pre}(o), \mathsf{add}(o), \mathsf{cost}(o) \rangle$, where $\mathsf{pre}(o) \subseteq F$ is the set of preconditions, $\mathsf{add}(o) \subseteq F$ is the set of additive effects, and $\mathsf{cost}(o) \geq 0$ corresponds to the operator cost. We assume, without loss of generality, that $\mathsf{add}(o) \cap \mathsf{pre}(o) = \emptyset$ for all operators $o \in O$. We say that an operator o is applicable to a state s if its preconditions are true in s, i.e., $\mathsf{pre}(o) \subseteq \mathsf{s}$. Given a state s and an applicable operator o, the successor state s' is given by $\mathsf{s}' = \mathsf{succ}(\mathsf{s}, o) = \mathsf{s} \cup \mathsf{add}(o)$. In general, we say that a sequence of operators $(o_1, \ldots, o_k)$ is applicable to a state s if o_1 is applicable in s, o_2 is applicable in $\mathsf{s}_1 = \mathsf{succ}(\mathsf{s}, o_1)$, and o_i is applicable in $\mathsf{s}_{i-1} = \mathsf{succ}(\mathsf{s}_{i-2}, o_{i-1})$ for all $i \in \{3, \ldots, k\}$. Extending the notation, given a state s and a sequence of applicable operators $(o_1, \ldots, o_k)$, $\mathsf{succ}(\mathsf{s}, (o_1, \ldots, o_k))$ represents the resulting state after applying the operator sequence.

Given a DFP task Π^+, we define a valid plan $\pi = (o_1, ..., o_k)$ as a sequence of applicable operators

from the initial state s_I such that the resulting state $\mathsf{s}_G = \mathsf{succ}(\mathsf{s}_I, \pi)$ satisfies all goal facts, i.e., $G \subseteq \mathsf{s}_G$. We use notation $\pi \in \Pi^+$ to indicate that the sequence of operators π is a plan for the DFP task Π^+. Similarly, we say that π is an *invalid* plan if it is not a plan, i.e., $\pi \notin \Pi^+$. The cost of a plan is given by the total cost of its operators, i.e., $\mathsf{cost}(\pi) = \sum_{o \in \pi} \mathsf{cost}(o)$ where $o \in \pi$ represents that operator o is part of plan π. In particular, a cost-optimal plan π^* for task Π^+ is a plan with minimal cost, i.e., $\mathsf{cost}(\pi^*) \leq \mathsf{cost}(\pi)$ for all plans $\pi \in \Pi^+$.

Throughout this work we assume that the DFP task Π^+ is solvable, i.e., there exists a valid plan $\pi \in \Pi^+$. Since it is possible to check if Π^+ is solvable in polynomial time (Blum and Furst, 1997), this assumption does not limit the applicability of our approach.

We say that $f \in F$ is a *fact landmark* if f is true at some point in all valid plans (Porteous et al., 2001). In the DFP case, $f \in F$ is a fact landmark if $f \in \mathsf{succ}(\mathsf{s}_I, \pi)$ for all plans $\pi \in \Pi^+$. We use the symbol L^F to represent the set of all fact landmarks. Notice that, by definition, all goals are fact landmarks, i.e., $G \subseteq L^F$. Similarly, we say that an operator $o \in O$ is an *operator landmark* if $o \in \pi$ for all plans $\pi \in \Pi^+$.

Recall that it is possible to extract all fact landmarks in polynomial time for any DFP task, e.g., using the relaxed planning graph (Porteous et al., 2001). However, finding all fact landmarks in a classical AI planning task is as hard as solving the planning task itself, i.e., PSPACE-hard (Bylander, 1994).

5.1.1 Sequential Relaxation

This work considers a relaxation of the DFP task to compute admissible heuristics based on BDDs. The sequential relaxation of a DFP task, also known as a temporal relaxation, ignores the order in which the operators are applied (Imai and Fukunaga, 2015).

Definition 5.1. *Given a DFP task Π^+, a valid sequence-relaxed plan for Π^+, sr-plan, is a set of operators $\pi_{sr} = \{o_1, \ldots, o_k\}$ such that:*

(i) For every operator $o \in \pi_{sr}$, each fact $f \in \mathsf{pre}(o)$ is true in s_I or is added by some operator $o' \in \pi_{sr}$, $o \neq o'$.

(ii) Each goal $f \in G$ is true in s_I or is added by some operator $o \in \pi_{sr}$.

Similar to the cost of a plan, the cost of an sr-plan is given by the sum of the cost of its operators, i.e., $\mathsf{cost}(\pi_{\mathsf{sr}}) = \sum_{o \in \pi_{\mathsf{sr}}} \mathsf{cost}(o)$. The sequential relaxation task asks for a minimum cost sr-plan. Since every plan for Π^+ is an sr-plan, it follows that any cost-optimal plan π^* has a cost greater or equal to any cost-optimal sr-plan π^*_{sr}, i.e., $\mathsf{cost}(\pi^*_{\mathsf{sr}}) \leq \mathsf{cost}(\pi^*)$. Similar to DFP, finding an optimal sr-plan can be shown to be NP-hard by a reduction from set covering (Bylander, 1994).

Example 5.1 Consider the following DFP task Π^+_4 of the visit-all domain (García-Olaya et al., 2011). The set of facts is given by $F = \{i_1, r_1, i_2, r_2, i_3, r_3, i_4, r_4\}$, where facts i_k and r_k represent that the agent *is currently at* and *has visited* room $k \in \{1, 2, 3, 4\}$, respectively. The set of operators $O = \{o_{1,2}, o_{2,1}, o_{2,3}, o_{3,2}, o_{3,4}, o_{4,3}, o_{1,4}, o_{4,1}\}$ is such that $o_{k,j} \in O$ represents the movement from room k to j with $\mathsf{pre}(o_{k,j}) = \{i_k\}$, $\mathsf{add}(o_{k,j}) = \{i_j, r_j\}$ and $\mathsf{cost}(o_{k,j}) = 1$. Figure 5.1 depicts the initial state (left image) and goal conditions (right image), where the human symbol (🚹) represents the position of the agent and the square (■) a visited room.

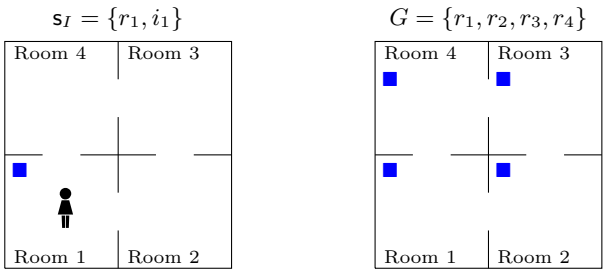


Figure 5.1: Visit-all domain with 4 rooms. Initial state in the right and goal facts in the left.

A cost-optimal plan for task Π_4^+ is $\pi = (o_{1,2}, o_{2,3}, o_{3,4})$ with $\mathsf{cost}(\pi) = 3$. A cost-optimal sr-plan is $\pi_{\mathsf{sr}} = \{o_{2,3}, o_{3,2}, o_{3,4}\}$ with $\mathsf{cost}(\pi_{\mathsf{sr}}) = 3$. Notice that while π is also a cost-optimal sr-plan, π_{sr} is an invalid plan since no operator $o \in \pi_{\mathsf{sr}}$ is applicable in s_I. □

5.2 Related Works

Cost-optimal DFP is a well-studied problem in the planning community and has inspired several state-of-the-art heuristics (Betz and Helmert, 2009). For example, critical path heuristics (Bonet and Geffner, 1999) (e.g., h^{max}) are obtained from graphical structures build over DFP tasks. While these heuristics can be used on their own, they are also employed as sub-routines in more complex state-of-the-art admissible heuristic procedures, e.g., in the LM-Cut heuristic (Helmert and Domshlak, 2009). Similarly, disjunctive landmarks are usually extracted from DFP tasks (Porteous et al., 2001) and are key components of many state-of-the-art heuristics in classical AI planning, such as the operator counting heuristic (Pommerening et al., 2014).

Several researchers have also proposed procedures to tackle cost-optimal DFP on its own. In particular, Bonet and Helmert (2010) show that a DFP task can be reformulated as a hitting set problem with exponentially many subsets, each encoding a separate disjunctive landmark. This result has been extended to derive the necessary set of disjunctive landmarks (Bonet and Castillo, 2011) and the set-inclusion minimal disjunctive landmarks (Haslum et al., 2012) required to solve a DFP task. Moreover, Pommerening and Helmert (2012) build on the hitting set representation to design an incremental disjunctive landmark heuristic for DFP.

Imai and Fukunaga (2014) presented a MILP formulation for a DFP task with polynomially many constraints. This formulation is currently regarded as the state-of-the-art solution approach for DFP tasks. Moreover, its LP relaxation coupled with operator counting constraints (Pommerening et al., 2014) defines an admissible heuristic that achieved competitive performance with respect to state-of-the-art classical AI planning heuristics (Imai and Fukunaga, 2015).

Recent works have shown an interest on relaxed DDs to compute admissible heuristics for planning tasks. Castro et al. (2018) use MDDs to create a relaxed representation of the state-transition graph for classical AI planning. The approach relates to several well-known techniques, such as critical path heuristics and abstractions. The authors report preliminary results that indicate the potential of the technique when it is used to extract plans.

The techniques proposed in this work are related to the work by Corrêa et al. (2018), who apply relaxed DDs to approximate the state-space of a DFP task. Our methodology, however, differs in three

main ways. First, the authors use a BDD encoding with a dynamic operator branching per node, while we propose an MDD encoding that focuses on the sequencing aspect of the problem. Second, our BDD models the sequential relaxation of a DFP task instead of the full DFP task. Lastly, Corrêa et al. (2018) implement a top-down DD construction, while we propose an iterative refinement procedure for our two DD approaches. While the top-down construction is faster, the iterative refinement leverages the encoded information to create a stronger relaxation (see Chapter 2).

There are other applications of decision diagrams in planning that are not closely related to our approach but that we note here for completeness. BDDs have been used in planning to succinctly represent sets of states (symbolic states). Using this representation, a symbolic version of the A^* search algorithm achieved state-of-the-art performance in cost-optimal classical AI planning (Torralba et al., 2016). Several admissible heuristics have been proposed to guide the search over the symbolic state-space, e.g., abstraction-based heuristics (Edelkamp et al., 2012; Torralba et al., 2013). Lastly, the planning literature has utilized arc-value multivalued decision diagrams to represent cost functions of planning problems with state-dependent operators costs (Keller et al., 2016; Geißer et al., 2016).

5.3 MDD Encoding for Delete-Free AI Planning

We now introduce our MDD encoding for a DFP task Π^+ and describe the relaxed MDD construction procedure. In the following, we rely on the notation and concepts introduced in Chapter 2 to explain the MDD encoding and construction procedure.

Our MDD encoding represents the sequential aspect of the problem by compactly encoding all cost-optimal plans. To do so, we start by presenting a recursive formulation for a DFP task. Since the number of operators in any cost-optimal plan is unknown, our encoding considers a dummy operator $o_{\texttt{noop}}$ that represents the execution of no further operators: $\mathsf{pre}(o_{\texttt{noop}}) = G$, $\mathsf{add}(o_{\texttt{noop}}) = \emptyset$, and $\mathsf{cost}(o_{\texttt{noop}}) = 0$. We use notation $O_0 = O \cup \{o_{\texttt{noop}}\}$ to refer to the set of operators including $o_{\texttt{noop}}$. In the following, we assume that we have an upper bound on the maximum number of operators needed by any cost-optimal plan $n \leq |O|$ (see Section 5.3.3 on how to compute n). Since an operator needs to be applied at most once in each cost-optimal plan of Π^+, the total number of operators $|O|$ is a trivial upper bound on the maximum number of operators needed for a solvable DFP task.

Our recursive model R-DFP for task Π^+ is defined over $n+1$ stages. The model considers a n-dimensional decision variable $\boldsymbol{x}$ where each x_i represents the i-th operator in a plan, i.e., $x_i \in O_0$. The states of the system corresponds to the states of a DFP task, thus, state $\boldsymbol{S} = A$ represents the set of facts that are achieved (i.e., facts with true value) in its respective planning state. The initial state of the system is given by $\boldsymbol{S}_1 = \mathsf{s}_I$ and $\mathcal{S}_i$ represent the set of reachable states after applying $i-1$ operators. For each state $A \in \mathcal{S}_i$ and stage $i \in \{1, ..., n\}$, the feasibility set is given by the set of applicable operators and we impose the terminal states to be goal states, i.e.,

$$X_i(A) = \{x \in O_0 : \mathsf{pre}(x) \subseteq A\}, \qquad \forall\, i \in \{1, ..., n-1\},$$
$$X_n(A) = \{x \in O_0 : \mathsf{pre}(x) \subseteq A \text{ and } G \subseteq A \cup \mathsf{add}(x)\}.$$

The transition function is defined as the successor of a state, i.e., $\phi_i(A, x) = A \cup \mathsf{add}(x)$ for any $A \in \mathcal{S}_i$, $x \in X_i(A)$, and $i \in \{1, ..., n\}$.

The model considers a state-independent immediate cost function given by the operator cost, i.e.,

$f_i(A, x) = \mathsf{cost}(x)$ for all $i \in \{1, ..., n\}$. Then, our recursive model is given by

$$h_i(A) = \min_{x \in X_i(A)} \{\mathsf{cost}(x) + h_{i+1}(A \cup \mathsf{add}(x))\}, \qquad \forall i \in \{1, \dots, n\}, \tag{R-DFP}$$

where $h_i(A)$ is the accumulated cost of state A at stage i, and the accumulated cost at stage $n+1$ is $h_{n+1}(A) = 0$.

The above recursive model allow us to define an MDD $\mathcal{M} = (\mathcal{N}, \mathcal{A})$ for a DFP task Π^+ as follows. The set of nodes $\mathcal{N}$ is partitioned into $n+1$ layers $\mathcal{N}_1, \dots, \mathcal{N}_{n+1}$ where arcs emanating from layer $\mathcal{N}_i$ point to nodes in layer $\mathcal{N}_{i+1}$. We associate a value $v_a \in O_0$ to each arc $a \in \mathcal{A}$ emanating from $\mathcal{N}_i$ that represents the operator assigned to the i-th position of a plan, i.e., paths traversing arc a have $x_i = v_a$.

We say that an MDD $\mathcal{M}$ exactly represents a DFP task Π^+ if there is a one-to-one correspondence between $\mathrm{Sol}(\mathcal{M})$ and all plans of Π^+. Intuitively, an MDD that represents the state-space transition graph of task Π^+ is an exact MDD. However, building such MDD is intractable since, in the worst case, its size is exponential with respect to the number of facts in F. Thus, we consider a relaxed MDD for Π^+, i.e., every cost-optimal plan is encoded in some $\mathbf{r} - \mathbf{t}$ path of $\mathcal{M}$, but some paths in $\mathrm{Sol}(\mathcal{M})$ might represent invalid plans. Specifically, invalid plans in $\mathcal{M}$ may consider operators that are not applicable or might fail to achieve all the goals.

Example 5.2 Consider the DFP task Π_4^+ described in Example 5.1. Figure 5.2 illustrates an exact and a relaxed MDD for this domain with $n = 3$. The operator associated with each arc is shown next to the arc. When arcs are too close to each other, we use a set notation to represent the operators associated with a set of arcs.

Notice that every $\mathbf{r} - \mathbf{t}$ path in the exact MDD (left diagram) corresponds to a cost-optimal plan for Π_4^+ and every cost-optimal plan for Π_4^+ has a corresponding $\mathbf{r} - \mathbf{t}$ path. In contrast, the relaxed MDD (right diagram) has a path for each cost-optimal plan of Π_4^+ but there exist some paths that are invalid plans. For example, path $p = (o_{1,4}, o_{1,2}, o_{\mathtt{noop}})$ is an invalid plan because operator $o_{\mathtt{noop}}$ is not applicable in state $\mathsf{s} = \mathsf{succ}(\mathsf{s}_I, (o_{1,4}, o_{1,2}))$. □

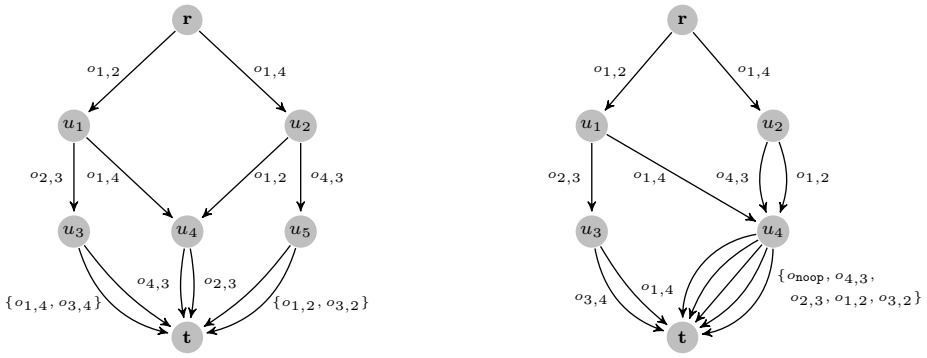


Figure 5.2: An exact MDD (left) and a relaxed MDD (right) for the 4-room visit-all DFP task.

We utilize the iterative refinement procedure described in Chapter 2 to construct our relaxed MDD (see Algorithm 3). The following sections explain the main components to build our relaxed MDD and our critical path algorithm over the resulting MDD to compute an admissible heuristic.

5.3.1 MDD Relaxed States and Filtering Rules

We now define the relaxed states and filtering rules employed in our relaxed MDD construction procedure. As explained in Chapter 2, relaxed states are approximations of the states of our recursive model R-DFP. In particular, we define relaxed states to approximate the set of true facts in an MDD node, i.e., the achieved set A. We also include relaxed states that approximate the set of needed facts to construct a plan achieved (i.e., goals and preconditions) which we employ in our filtering rules.

Achieved Relaxed States. We first present the relaxed states for our achieved facts state variable A. State variable A is used to ensure that all goals are achieved and that operators are applicable. Thus, we create two relaxed state variables that over and under approximate the set of achieved facts in every MDD node. For each node $u \in \mathcal{N}$, consider the top-down relaxed states $A_{\text{all}}(u)$ and $A_{\text{some}}(u)$ as the set of facts that are achieved by *all* and at least *some* $\mathbf{r}-u$ path in $\mathcal{M}$, respectively. Starting with the root node $\mathbf{r}$, we initialize the sets with the facts in the initial state, $A_{\text{all}}(\mathbf{r}) = A_{\text{some}}(\mathbf{r}) = \mathsf{s}_I$, and updates each node $u \in \mathcal{N} \setminus \{\mathbf{r}\}$ as:

$$A_{\text{all}}(u) = \bigcap_{a \in \mathcal{A}^{\text{in}}(u)} \left(A_{\text{all}}(s(a)) \cup \mathsf{add}(v_a)\right),$$
$$A_{\text{some}}(u) = \bigcup_{a \in \mathcal{A}^{\text{in}}(u)} \left(A_{\text{some}}(s(a)) \cup \mathsf{add}(v_a)\right).$$

Similarly, we define a bottom-up relaxed state $A^{\uparrow}_{\text{some}}(u)$ for node $u \in \mathcal{N}$ as the set of facts added by some operator in a $u-\mathbf{t}$ path. Starting with the terminal node $\mathbf{t}$, the procedure start with an empty set of facts $A^{\uparrow}_{\text{some}}(\mathbf{t}) = \emptyset$ and updates each node $u \in \mathcal{N} \setminus \{\mathbf{t}\}$ as:

$$A^{\uparrow}_{\text{some}}(u) = \bigcup_{a \in \mathcal{A}^{\text{out}}(u)} \left(A^{\uparrow}_{\text{some}}(t(a)) \cup \mathsf{add}(v_a)\right).$$

We employ the above relaxed states to define two filtering rules for $\mathcal{M}$, as shown in Proposition 5.1. The first filtering rule DFP-R1 forces each arc to represent applicable operators. The second rule DFP-R2 removes paths corresponding to plans that fail to achieve some fact landmark. The latter rule can be define over the set of goals G. However, we can potentially remove more arcs if we used the set of fact landmarks since every goal is a fact landmark, $G \subseteq L^F$.

Proposition 5.1. *An arc $a \in \mathcal{A}$ can be removed from $\mathcal{M}$ if either of the following conditions hold:*

$$\mathsf{pre}(v_a) \not\subseteq A_{\text{some}}(s(a)), \tag{DFP-R1}$$
$$L^F \not\subseteq A_{\text{some}}(s(a)) \cup \mathsf{add}(v_a) \cup A^{\uparrow}_{\text{some}}(t(a)). \tag{DFP-R2}$$

Proof. Consider any arbitrary $\mathbf{r}-\mathbf{t}$ path $p = (a_1, ..., a_n) \in \mathcal{P}$ that corresponds to a valid plan $\pi \in \Pi^+$. We show that none of arcs $a \in p$ satisfy conditions DFP-R1 or DFP-R2. Therefore, no arc associated to a valid plan is removed with the above filtering rules.

For each arc $a_i \in p$, consider its associated operator $o_i = v_{a_i}$ and state $\mathsf{s}_{i-1} = \mathsf{succ}(\mathsf{s}_I, (o_1, ..., o_{i-1}))$. By definition, we have that $\mathsf{s}_{i-1} \subseteq A_{\text{some}}(s(a_i))$ for any $i \in \{1, ..., n\}$. Since p is a valid plan, the relation $\mathsf{pre}(o_i) \subseteq \mathsf{s}_{i-1} \subseteq A_{\text{some}}(a_i)$ holds and, thus, DFP-R1 is violated. Similarly, due to our relaxed state definitions, we have that $L^F \subseteq \mathsf{succ}(\mathsf{s}_I, \pi) \subseteq A_{\text{some}}(s(a_i)) \cup \mathsf{add}(o_i) \cup A^{\uparrow}_{\text{some}}(t(a_i))$ for all $i \in \{1, ..., n\}$,

so condition DFP-R2 is violated. ■

Needed Relaxed States. With the aim to create additional filtering rules, we introduce an additional bottom-up relaxed state. Intuitively, this relaxed state represents the set of facts that are preconditions to some operator or a goal fact. Formally, relaxed state $N^{\uparrow}_{\text{some}}(u)$ for a node $u \in \mathcal{N}$ is the set of facts that need to be achieved by at least some operator in a $u - \mathbf{t}$ path. Starting with the terminal node $\mathbf{t}$, we initialize the relaxed state with all the fact landmarks $N^{\uparrow}_{\text{some}}(\mathbf{t}) = L^F$ and update each $u \in \mathcal{N}$ as

$$N^{\uparrow}_{\text{some}}(u) = \bigcup_{a \in \mathcal{A}^{\text{out}}(u)} \left(N^{\uparrow}_{\text{some}}(t(a)) \cup \mathsf{pre}(v_a) \right).$$

Notice that initializing the needed relaxed state at the terminal node with the goal set (i.e., $N^{\uparrow}_{\text{some}}(\mathbf{t}) = G$) is also valid but might lead to weaker filtering rules.

The purpose of this relaxed state is to identify paths that include operators that achieve unnecessary facts, i.e., remove redundant operators. We note that removing redundant operators can potentially prune cost-optimal plans from tasks with zero-cost operators, since applicable zero-cost operators can be added to any cost-optimal plan even though they are redundant. Nonetheless, the correctness of our admissible heuristic computation holds if we maintain all *minimal* cost-optimal plans (i.e., plans without redundant operators). Proposition 5.2 presents filtering rule DFP-R3 and shows its validity.

Proposition 5.2. *An arc $a \in \mathcal{A}$ is associated to a redundant operator if*

$$\mathsf{add}(v_a) \subseteq A_{\text{all}}(s(a)) \qquad \text{or} \qquad \forall f \in \mathsf{add}(v_a) \setminus A_{\text{all}}(s(a)),\ f \notin N^{\uparrow}_{\text{some}}(t(a)). \tag{DFP-R3}$$

Thus, arc a can be removed from $\mathcal{M}$.

Proof. Consider any $\mathbf{r} - \mathbf{t}$ path $p = (a_1, ..., a_n) \in \mathcal{M}$ that corresponds to a minimal cost-optimal plan $\pi = (o_1, ..., o_n) \in \Pi^+$ with $o_i = v_{a_i}$ and $\mathsf{s}_i = \mathsf{succ}(\mathsf{s}_I, (o_1, ..., o_i))$. Consider any arc $a_i \in p$ and its operator o_i. Since π is a minimal plan, operator o_i adds at least one new fact $f \in \mathsf{add}(o_i) \setminus \mathsf{s}_{i-1}$ that is a precondition for some operator o_j $(j > i)$ or is a landmark $f \in L^F$. Then, since $A_{\text{all}}(s(a_i)) \subseteq \mathsf{s}_{i-1}$, condition DFP-R3 is violated. ■

5.3.2 MDD Splitting Algorithm

We now describe the node splitting procedure inside the relaxed MDD construction algorithm. As explained in Chapter 2, the SplitDDNodes procedure increases the size of $\mathcal{M}$ to strengthen the relaxation. Our procedure attempts to split nodes so each node represents exactly one state in the search-space. Given a node $u \in \mathcal{N}$, we say that u is *exact* if $A_{\text{all}}(u) = A_{\text{some}}(u)$. As shown in Proposition 5.3, we can eliminate all invalid plans from an MDD if all its nodes are exact.

Proposition 5.3. *Consider a relaxed MDD $\mathcal{M} = (\mathcal{N}, \mathcal{A})$ where each node $u \in \mathcal{N} \setminus \{\mathbf{t}\}$ is exact, i.e., $A_{\text{all}}(u) = A_{\text{some}}(u)$. Then, filtering rules DFP-R1 and DFP-R2 are sufficient to identify and remove all infeasible paths in $\mathcal{M}$.*

Proof. Consider any $\mathbf{r} - \mathbf{t}$ path $p = (a_1, ..., a_n) \in \mathcal{M}$ with associated operators $o_i = v_{a_i}$ that forms an invalid plan. Assume that $(o_1, ..., o_{i-1})$ is applicable to the initial state, i.e., $\mathsf{s}_{i-1} = \mathsf{succ}(\mathsf{s}_I, (o_1, ..., o_{i-1}))$, but o_i is not applicable to s_{i-1}. We have that $A_{\text{all}}(s(a_i)) = \mathsf{s}_{i-1} = A_{\text{some}}(s(a_i))$ by definition of A_{all} and

A_{some}, and the fact that node $s(a_i)$ is exact. Then, rule DFP-R1 removes arc a_i since $\mathsf{pre}(o_i) \not\subseteq \mathsf{s}_{i-1} = A_{\text{some}}(s(a_i))$.

Now assume that plan $\pi = (o_1, ..., o_n)$ is applicable to the initial state but the resulting state omits at least one goal, i.e., $G \not\subseteq \mathsf{s}_n = \mathsf{succ}(\mathsf{s}_I, \pi)$. Then, rule DFP-R2 eliminates arc $a_n = (u, \mathbf{t})$ since $G \not\subseteq \mathsf{s}_n = \mathsf{s}_{n-1} \cup \mathsf{add}(o_n) \cup \emptyset = A_{\text{some}}(u) \cup \mathsf{add}(o_n) \cup A^{\uparrow}_{\text{some}}(\mathbf{t})$. ■

Our SplitDDNodes procedure (Algorithm 9) iterates over the set of nodes in a layer and splits inexact nodes such that the resulting nodes represent fewer aggregated states. To do so, the procedure sorts the facts that are not in the initial state, $F_{\neg \mathsf{s}_I} = F \setminus \mathsf{s}_I$, in a priority queue Q and iterates over them (lines 2-3). For each fact $f \in Q$, the algorithm looks for nodes $u \in \mathcal{N}_i$ where fact f is achieved by some $\mathbf{r} - u$ paths but not all (lines 4-6). We then identify the set of incoming arcs where f is always achieved (line 7) and split the node into two: node u' where f is achieved by all $\mathbf{r} - u'$ paths, and node u where f is not achieved (lines 9-10). Lastly, the algorithm duplicates the outgoing arcs from the original node u to the resulting node u'. Notice that it might be impossible to split u with respect to fact f: $\delta_{\mathtt{all}}$ can be empty if the nodes in the previous layer correspond to the aggregation of distinct states (line 8).

Algorithm 9 Relaxed MDD Split Nodes Procedure

```
1:  procedure SplitDDNodes(𝒩_i, 𝒲)
2:      Initialize Q with all the facts in F_{¬s_I}
3:      while |Q| > 0 and |𝒩_i| < 𝒲 do
4:          Remove first fact f in Q
5:          for u ∈ 𝒩_i do
6:              if f ∈ A_some(u) and f ∉ A_all(u) then
7:                  δ_all = {a ∈ 𝒜^in(u) : f ∈ add(v_a) or f ∈ A_all(s(a))}
8:                  if δ_all ≠ ∅ then
9:                      Create new node u'. 𝒩_i = 𝒩_i ∪ {u'}
10:                     Redirect arcs: 𝒜^in(u') = δ_all and 𝒜^in(u) = 𝒜^in(u) \ δ_all
11:                     if 𝒜^in(u') ≠ ∅ then Duplicate outgoing arcs from u to u'
```

Our algorithm uses a priority queue Q over the facts in $F_{\neg \mathsf{s}_I}$ to split the nodes using the same fact order. The queue is divided into three priority levels where goals ($f \in G$) have the highest priority, followed by fact landmarks ($f \in L^F \setminus G$) and lastly the remaining facts ($f \in F_{\neg \mathsf{s}_I} \setminus L^F$).

This splitting procedure needs at most $2^{|F_{\neg \mathsf{s}_I}|}$ nodes in each layer to create an exact MDD. Moreover, if $\mathcal{W} = 2^{|F_{\neg \mathsf{s}_I}|}$, nodes in layer $\mathcal{N}_i$ represent all the states that can be reach after applying i operators.

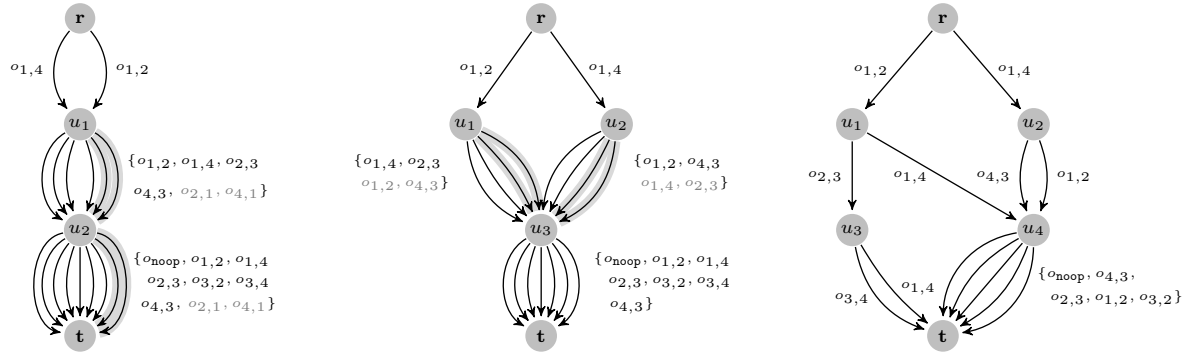


Figure 5.3: MDD construction example with $\mathcal{W} = 2$. The left diagram is a width-one MDD, the middle MDD illustrates the splitting algorithm, and the right diagram is the resulting MDD.

Example 5.3 Consider the DFP task Π_4^+ described in Example 5.1. Figure 5.3 illustrates some of the steps of Algorithm 3 for a relaxed MDD with $n = 3$ and $\mathcal{W} = 2$. The left diagram corresponds to the initial width-one MDD where gray arcs will be removed by filtering rule DFP-R3. The middle diagram illustrates the resulting MDD after applying the SplitDDNodes procedure in the second layer. Lastly, the left most relaxed MDD is the one returned by the construction procedure. □

5.3.3 Estimating the Number of Layers

Given a DFP task Π^+, the number of operators in a cost-optimal plan n is unknown but it is possible to create valid upper bounds. We propose two simple procedures to over-estimate n using the cost of any valid plan $\pi' \in \Pi^+$. Notice that we can generate a valid plan for a DFP task Π^+ in polynomial time using, for instance, the FF heuristic (Hoffmann and Nebel, 2001).

Algorithm 10 Maximum number of layers estimation

```
1:  procedure MaxLayerOperators(O, π')
2:      Q_O = (o_1, ...., o_|O|), list of operators ordered in increasing value of cost(o)
3:      C = 0, n = 0
4:      while |Q_O| > 0 and C ≤ cost(π') do
5:          Remove the first operator o in Q_O
6:          C = C + cost(o), n = n + 1
7:      return n

8:  procedure MaxLayerFacts(O, F_¬s_I, π')
9:      C = 0, n = 0
10:     for f ∈ F_¬s_I do
11:         cost(f) = min{cost(o) : f ∈ add(o)}
12:     Q_F = (p_1, ...., p_|F_¬s_I|), list of facts ordered in increasing value of cost(f)
13:     while |Q_F| > 0 and C ≤ cost(π') do
14:         Remove first fact f in Q_F
15:         C = C + cost(p), n = n + 1
16:     return n
```

Algorithm 10 shows our two estimation procedures, MaxLayerOperators and MaxLayerFacts. Procedure MaxLayerOperators uses the fact that each operator is applied at most once. The procedure sorts the operators in increasing order of cost (line 2) and sums up the cost of each operator in the queue until the total cost C is greater than the cost of our known plan π' (lines 3-6). Then, the total number of operators used to compute C is an upper bound on the maximum number of operators in any cost-optimal plan.

The second procedure, MaxLayerFacts, is valid only for minimal cost-optimal plans (i.e., plans without redundant operators). The procedure estimates the maximum number of operators in any minimal cost-optimal plan by assuming that each operator in the plan adds at least one fact that is not present in the previous state. The procedure first calculates the minimum cost of adding each fact $f \in F_{\neg s_I}$ by taking the minimum cost operator that adds f (lines 10-11). Then, we order the facts in increasing order of cost (line 12) and sum up the cost of each fact until the total cost C is greater than $\mathsf{cost}(\pi')$ (lines 13-15). Lastly, the total number of facts in the sum corresponds to the maximum number of operators in any minimal cost-optimal plan.

Our implementation considers the minimum between the two estimations. We notice that MaxLayerFacts tends to give a tighter bound when Π^+ has zero-cost operators, while MaxLayerOperators is more

accurate for tasks with operator costs greater or equal to one. Both procedures compute the same value of n when all operators have the same costs.

5.3.4 MDD Bound Computation

This section describes how to obtain lower bounds from a relaxed MDD, which we later use as our relaxed MDD heuristic. Our recursive model R-DFP has a state-independent cost function, thus, we can apply the shortest path procedure in Section 2.2.4 to get a valid lower bound. However, this procedure can lead to poor bounds (e.g., a zero-cost heuristic value) in tasks where there exist zero-cost operators. To compute more informative heuristics, we consider instead a critical path approach over the MDD (Castro et al., 2018). The idea is to keep track of the cost to achieve each fact in a node, and use the cost of the most expensive precondition to compute the length of an arc.

Formally, consider $c(u, f)$ to be the cost to achieve fact $f \in F$ in node $u \in \mathcal{N}$. The cost of each fact $f \in F$ at the root node $\mathbf{r}$ is zero, i.e., $c(\mathbf{r}, f) = 0$. Using this information, we compute the length ℓ_a of arc $a \in \mathcal{A}$ as its operator cost plus the most expensive precondition in its source node, i.e.,

$$\ell_a = \mathsf{cost}(v_a) + \max\{c(s(a), f) : \ f \in \mathsf{pre}(v_a)\}.$$

Thus, the length of an arc represents the accumulated cost of applying its corresponding operator at its source node. To have a better estimation, we actually compute the length of an arc for each fact in its resulting state. Let ℓ_a^f be the cost of an arc $a \in \mathcal{A}$ with respect to a fact $f \in F$:

$$\ell_a^f = \begin{cases} \ell_a, & f \in \mathsf{add}(v_a), \\ \max\{\ell_a,\ \mathsf{cost}(v_a) + c(s(a), f)\}, & f \notin \mathsf{add}(v_a),\ f \in A_{\text{some}}(s(a)), \\ \infty, & \text{otherwise.} \end{cases}$$

The above estimation distinguishes between facts that are added by the operator and facts that are present in the source state. In the first case, ℓ_a^f corresponds to the critical path cost ℓ_a. For the second case, we can under-approximate ℓ_a^f either using the critical path formula ℓ_a or summing up the operator and source node cost for that fact. Since there is no dominance between these two values, we choose the maximum.

Then, we use the above arc length values ℓ_a^f to compute the cost to achieve each fact in its target state. For each node $u \in \mathcal{N} \setminus \{\mathbf{r}\}$ and fact $f \in F$, the cost is given by:

$$c(u, f) = \begin{cases} \min\{\ell_a^f \ : \ a \in \mathcal{A}^{\text{in}}(u)\}, & f \in A_{\text{some}}(u), \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the cost of fact f at node u is the minimum cost over all incoming arcs if f is achieved by at least one $\mathbf{r} - u$ path. Otherwise, we set the cost to zero to guarantee the admissibility of our heuristic. Proposition 5.4 shows that this cost computation under-approximates the cost of each plan represented as a path in the MDD.

Proposition 5.4. *Consider any path $p = (a_1, ..., a_n) \in \mathcal{M}$ that corresponds to a valid plan $\pi = (o_1, ..., o_n) \in \Pi^+$, with $o_i = v_{a_i}$. The cost of any partial plan $\pi_i = (o_1, ..., o_i)$ is an upper bound to*

the cost of each fact $f \in \mathsf{s}_{i+1} = \mathsf{succ}(\mathsf{s}_I, \pi_i)$ *in the corresponding MDD node* $u_{i+1} = t(a_i) \in \mathcal{N}_{i+1}$*, i.e.,*

$$c(u_{i+1}, f) \leq \mathsf{cost}(\pi_i), \qquad \forall\, f \in \mathsf{s}_{i+1},\ i \in \{1, ..., n\}. \tag{5.1}$$

Proof. We prove (5.1) by induction. In the initial case $i = 1$ we have that $c(u_2, f) \leq 0 + \mathsf{cost}(o_1) = \mathsf{cost}(\pi_1)$ for all $f \in \mathsf{s}_I \cup \mathsf{add}(o_1)$. Next, consider that (5.1) holds for some $i \geq 1$ and we want to prove that it holds for $i + 1$. From (5.1) we have that $\max_{q \in \mathsf{s}_{i+1}} \{c(u_{i+1}, q)\} \leq \mathsf{cost}(\pi_i)$. Finally, for any $f \in \mathsf{s}_{i+2}$ we have that

$$c(u_{i+2}, f) \leq \ell^f_{a_{i+1}} \leq \max_{q \in \mathsf{s}_{i+1}} \{c(u_{i+1}, q)\} + \mathsf{cost}(o_{i+1}) \leq \mathsf{cost}(\pi_i) + \mathsf{cost}(o_{i+1}) = \mathsf{cost}(\pi_{i+1}).$$

■

Our relaxed MDD also includes a bottom-up cost estimation to create a cost-based filtering rule. In this case, we use a shortest path procedure where the cost of an arc is simply the cost of its associated operator. Let $c^{\uparrow}(u)$ be the bottom-up cost of a node $u \in \mathcal{N}$. Then, we assign $c^{\uparrow}(\mathbf{t}) = 0$ and, for each node $u \in \mathcal{N} \setminus \{\mathbf{t}\}$,

$$c^{\uparrow}(u) = \min_{a \in \mathcal{A}^{\text{out}}(u)} \left\{ c^{\uparrow}(t(a)) + \mathsf{cost}(v_a) \right\}.$$

The cost-based filtering rule removes a sub-optimal arc $a \in \mathcal{A}$ if the minimum cost traversing arc a is larger than the values of a known valid plan π'. Formally, we can remove an arc $a \in \mathcal{A}$ from $\mathcal{M}$ if:

$$\ell_a + c^{\uparrow}(t(a)) > \mathsf{cost}(\pi'). \tag{DFP-R4}$$

The validity of this rules follows from Proposition 5.4 and the bottom-up shortest path procedure.

5.4 Relaxed MDD-based Heuristic

We now present our relaxed MDD heuristic and prove its admissibility and consistency. Given any state s, we can create a relaxed MDD $\mathcal{M}$ for s updating the initial state $\mathsf{s}_I = \mathsf{s}$ and using a proper bound on the maximum number of layers n. Then, the relaxed MDD heuristic value for state s, $h^{\mathcal{M}}(\mathsf{s})$, is given by

$$h^{\mathcal{M}}(\mathsf{s}) = \max\{c(\mathbf{t}, f) : f \in L^F\}. \tag{5.2}$$

Theorem 5.1. *Consider a DFP task, a reachable state* s*, and a relaxed MDD* $\mathcal{M} = (\mathcal{N}, \mathcal{A})$ *for* s *with* $\mathcal{W} \geq 1$ *constructed using Algorithm 3. Then,* $h^{\mathcal{M}}(\mathsf{s}) = \max\{c(\mathbf{t}, f) : f \in L^F\}$ *is an admissible heuristic.*

Proof. Proposition 5.1 guarantees that all minimal cost-optimal plans are represented by $\mathcal{M}$. Then, by Proposition 5.4, $h^{\mathcal{M}}(\mathsf{s}) \leq \mathsf{cost}(\pi)$ for any minimal cost-optimal plan represented by a path in $\mathcal{M}$. ■

To avoid constructing a new MDD for each state in the search-space, we instead construct a relaxed MDD for the initial state s_I and update it during search. Given a state s and a sequence of operators that reach s from s_I, $\pi_{\mathsf{s}} = (o_1, ..., o_k)$, we update the relaxed MDD $\mathcal{M}$ by removing any arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$ ($i \in \{1, ..., k\}$) with $v_a \neq o_i$, i.e., the first k layers correspond to single path representing π_{s}. We then iteratively apply the top-down and bottom-up procedures (Algorithm 3) to

update $\mathcal{M}$. In this case, the heuristic is

$$h^{\mathcal{M}}(\mathsf{s}) = \begin{cases} \max\{c(\mathbf{t}, f) : f \in L^F\} - \mathsf{cost}(\pi_{\mathsf{s}}), & \pi_{\mathsf{s}} \in \mathcal{M}, \\ \infty, & \text{otherwise}, \end{cases} \tag{5.3}$$

where $\pi_{\mathsf{s}} \in \mathcal{M}$ implies that there exists a path in the MDD where the k initial operators are given by π_{s}. Notice that since all minimal cost-optimal plans are encoded by $\mathcal{M}$, the condition $h^{\mathcal{M}}(\mathsf{s}) = \infty$ holds only when there is no minimal cost-optimal plan from s_I that starts with the sequence of operators π_{s}. Thus, while the heuristic is inadmissible in these cases, it only gives value ∞ to sub-optimal states. In other words, $h^{\mathcal{M}}(\mathsf{s})$ given by (5.3) is a *global admissible* heuristic, as defined by Karpas and Domshlak (2012).

We choose this implementation because it speeds up the heuristic computation and it is compatible with our search procedure (see Section 5.9). In addition, the heuristic is consistent if the order of the facts in Q remains the same when updating $\mathcal{M}$ for state s (Theorem 5.2).

Theorem 5.2. *Consider a DFP task Π^+, a state s, and a relaxed MDD $\mathcal{M}$ with $\mathcal{W} \geq 1$ constructed using Algorithm 3. Then, $h^{\mathcal{M}}(\mathsf{s})$ given by (5.3) is consistent.*

Proof. Consider a state s, an applicable operator o, and its successor state $\mathsf{s}' = \mathsf{succ}(\mathsf{s}, o)$. Given the relaxed MDD $\mathcal{M}_{\mathsf{s}_I}$ for s_I, let $\mathcal{M}_{\mathsf{s}}$ and $\mathcal{M}_{\mathsf{s}'}$ be the updated MDDs for state s and s', respectively. Since each MDD is updated from $\mathcal{M}_{\mathsf{s}_I}$ without changing the fact order, every path $p \in \mathcal{M}_{\mathsf{s}'}$ is also in $\mathcal{M}_{\mathsf{s}}$. Then, we have $c(\mathbf{t}_{\mathsf{s}}, f) \leq c(\mathbf{t}_{\mathsf{s}'}, f)$ for all $f \in L^F$.

We know that

$$h^{\mathcal{M}}(\mathsf{s}) = \max_{f \in L^F}\{c(\mathbf{t}_s, f)\} - \mathsf{cost}(\pi_{\mathsf{s}}) \quad \text{and} \quad h^{\mathcal{M}}(\mathsf{s}') = \max_{f \in L^F}\{c(\mathbf{t}_{\mathsf{s}'}, f)\} - \mathsf{cost}(\pi_{\mathsf{s}}) - \mathsf{cost}(o).$$

Then,

$$h^{\mathcal{M}}(\mathsf{s}) - \mathsf{cost}(o) - h^{\mathcal{M}}(\mathsf{s}') = \max_{f \in L^F}\{c(\mathbf{t}_{\mathsf{s}}, f)\} - \max_{f \in L^F}\{c(\mathbf{t}_{\mathsf{s}'}, f)\} \leq 0,$$

and therefore $h^{\mathcal{M}}(\mathsf{s}) \leq \mathsf{cost}(o) + h^{\mathcal{M}}(\mathsf{s}')$. ■

Notice that $h^{\mathcal{M}}$ is consistent in this case because the set of paths encoded in the successor state MDD, $\mathcal{M}_{\mathsf{s}'}$, is a subset of the paths encoded by the current state MDD, $\mathcal{M}_{\mathsf{s}}$. Thus, any other implementation that guarantees this condition will result in a consistent heuristic.

5.4.1 Relationship to Critical Path Heuristics

We now compare our heuristic with the critical path heuristic h^{max} (Bonet and Geffner, 1999), and relate the MDD graphical structure to the planning graph (Blum and Furst, 1997). The h^{max} heuristic computes the minimum cost to reach each fact from the initial state. Specifically, consider $h(f)$ as the minimum cost to reach fact $f \in F$ and $h(o)$ as the minimum cost to use operator $o \in O$. These values can be computed recursively using the formula below and by setting $h(f) = 0$ for all $f \in \mathsf{s}_I$, $h(f) = \infty$ for any $f \notin \mathsf{s}_I$, and $h(o) = \infty$.

$$\begin{aligned} h(f) &= \min\{h(f), \min\{h(o) : f \in \mathsf{add}(o),\ o \in O\}\} && \forall f \in F, \\ h(o) &= \mathsf{cost}(o) + \max\{h(q) : q \in \mathsf{pre}(o)\} && \forall o \in O. \end{aligned}$$

The heuristic is defined as $h^{max} = \max\{h(f) : f \in G\}$. Notice that our MDD top-down node cost and arc length computations (Section 5.3.4) resemble the $h(f)$ and $h(o)$ recursions, respectively. Lemma 5.1 and Theorem 5.3 show that in fact our MDD heuristic dominates the h^{max} heuristic for a maximum width $\mathcal{W} \geq 1$.

Lemma 5.1. *Consider a DFP task Π^+, a reachable state s, and a relaxed MDD $\mathcal{M} = (\mathcal{N}, \mathcal{A})$ for s with $\mathcal{W} \geq 1$. Then,*

$$h(f) \leq c(u, f) \qquad \forall u \in \mathcal{N},\ f \in A_{\text{some}}(u). \tag{5.4}$$

Proof. We prove (5.4) by induction over the layers of $\mathcal{M}$. By construction, (5.4) holds for $\mathcal{N}_1 = \{\mathbf{r}\}$. Now consider that (5.4) is true for all nodes $u \in \mathcal{N}_i$ and $f \in A_{\text{some}}(u)$ $(i \geq 1)$. Consider any node $u' \in \mathcal{N}_{i+1}$ and a fact $f \in A_{\text{some}}(u')$. By construction, there exists an arc $a \in \mathcal{A}^{\text{in}}(u')$ such that $\ell_a^f = c(u', f)$. Consider operator $o = v_a$ and node $u = s(a) \in \mathcal{N}_i$. There are two cases, either $f \in \mathsf{add}(o)$ or not. If $f \in \mathsf{add}(o)$, then

$$c(u', f) = \ell_a^f = \mathsf{cost}(o) + \max\{c(u, q) : q \in \mathsf{pre}(o)\} \geq \mathsf{cost}(o) + \max\{h(q) : q \in \mathsf{pre}(o)\} \geq h(f).$$

If $f \notin \mathsf{add}(o)$, we necessarily have that $f \in A_{\text{some}}(u)$. Since $u \in \mathcal{N}_i$, we have $h(f) \leq c(u, f)$. Then it follows that $h(f) + \mathsf{cost}(o) \leq c(u, f) + \mathsf{cost}(o) \leq \ell_a^f = c(u', f)$. ■

Theorem 5.3. *Consider a DFP task Π^+, a reachable state s, and a relaxed MDD $\mathcal{M} = (\mathcal{N}, \mathcal{A})$ for s with $\mathcal{W} \geq 1$. Then, $h^{\mathcal{M}}(\mathsf{s}) \geq h^{max}(\mathsf{s})$.*

Proof. From Lemma 5.1 we have that $h(f) \leq c(\mathbf{t}, f)$ for any fact $f \in F$. Then, $h^{max}(\mathsf{s}) = \max\{h(f) : f \in G\} \leq \max\{c(\mathbf{t}, f) : f \in G\} = h^{\mathcal{M}}(\mathsf{s})$. ■

We can interpret our relaxed MDD heuristic as a type of critical path heuristic over the relaxed MDD graph instead of the planning graph (Blum and Furst, 1997). In fact, the planning graph can be seen as a relaxed MDD with $\mathcal{W} = 1$ where each node corresponds to a fact layer in the planning graph and each arc layer to an operator layer. As such, a relaxed MDD is a generalization of the planning graph since a relaxed MDD with $\mathcal{W} > 1$ considers more than one node per layer.

5.5 BDD Encoding for the Sequential Relaxation

We now investigate an alternative approach to compute admissible heuristics based on DDs: we use a binary decision diagram (BDD) to encode the sequential relaxation of a DFP task. Previous work has empirically shown that the sequential relaxation is an accurate approximation to a DFP task in most IPC domains (Imai and Fukunaga, 2015). Thus, a BDD encoding of the sequential relaxation can potentially compute informative heuristics using a smaller graphical structure than our relaxed MDD (see Section 5.7).

Our BDD encoding is based on a recursive model for the sequential relaxation, R-SR. The model considers one binary decision variable x_i for each operator $o_i \in O$ that represents whether the operator is part of an sr-plan or not. Therefore, the model has $m + 1$ stages where $m = |O|$ is the number of operators. In this case, the states of the system are given by $\boldsymbol{S} = (A, N)$ where A represents the set of facts that are achieved (i.e., facts in the initial state or added) and N is the set of needed facts (i.e., preconditions and goals). Thus, the first state in the system is given by the facts at the initial state

and the goal conditions, i.e., $\boldsymbol{S}_1 = (\mathsf{s}_I, G)$. There are no restrictions in the feasibility sets of the first $m-1$ stages, i.e., $X_i(A,N) = \{0,1\}$ for all $i \in \{1,...,m-1\}$, and the m-th feasibility set enforces the last-stage states $\boldsymbol{S}_{m+1} = (A_{m+1}, N_{m+1})$ to achieve all the needed facts, i.e., $N_{m+1} \subseteq A_{m+1}$.

The transition function of R-SR updates the set of achieved and needed facts only if the operator is selected, i.e.,

$$\phi_i((A,N),x) = \begin{cases} (A,N), & \text{if } x = 0, \\ (A \cup \mathsf{add}(o_i), N \cup \mathsf{pre}(o_i)), & \text{if } x = 1. \end{cases}$$

Lastly, the model considers a state-independent immediate cost function given by the operator cost if applied, i.e., $f_i((A,N),x) = x \cdot \mathsf{cost}(o_i)$ for all $i \in \{1,...,m\}$. Therefore, our recursive model is given by

$$h_i(A,N) = \min_{x \in X_{i+1}(A,N)} \{x \cdot \mathsf{cost}(o_i) + h_i(\phi_i((A,N),x)\}, \qquad \forall i \in \{1,\ldots,m\}, \tag{R-SR}$$

where $h_i(A,N)$ is the accumulated cost of state (A,N) at stage i, and the accumulated cost at stage $m+1$ is $h_{m+1}(A,N) = 0$.

We now define a BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ for the sequential relaxation of Π^+ using recursive model R-SR. The set of nodes is partitioned into $m+1$ layers $\mathcal{N} = (\mathcal{N}_1, \ldots, \mathcal{N}_{m+1})$ and we associate an operator o_i to each layer $\mathcal{N}_i$. Every arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$ has (i) a value $v_a \in \{0,1\}$ that represents if its associated operator $\mathsf{o}(a) = o_i$ is selected to be in an sr-plan or not, and (ii) a length ℓ_a derived from including (if $v_a = 1$) or excluding (if $v_a = 0$) the operator to the sr-plan, i.e., $\ell_a = v_a \cdot \mathsf{cost}(\mathsf{o}(a))$. A node $u \in \mathcal{N}$ has at most two arcs emanating from it, each with a distinct value. Thus, an $\mathbf{r}-\mathbf{t}$ path $p = (a_1, ..., a_m) \in \mathcal{B}$ has exactly one arc from each layer, and the values associated with its arcs correspond to the operators that will be included in and excluded from the sr-plan.

The BDD $\mathcal{B}$ is exact for the sequential relaxation of task Π^+ if there is a one-to-one correspondence between its solutions $\mathrm{Sol}(\mathcal{B})$ and all valid sr-plans of Π^+. As for the MDD case, we consider a relaxed BDD due to the exponentially many sr-plans. Therefore, every sr-plan is encoded in some $\mathbf{r}-\mathbf{t}$ path of $\mathcal{B}$, but some paths in $\mathrm{Sol}(\mathcal{B})$ might represent invalid sr-plans. Specifically, a invalid sr-plan in $\mathcal{B}$ fails to achieve all the needed facts.

Example 5.4 From now on we will consider a smaller task of the visit-all domain with three rooms, Π_3^+, since it results in a shorter BDD. The set of operators is $O = \{o_{1,2}, o_{2,1}, o_{2,3}, o_{3,2}\}$ and the set of facts is $F = \{i_1, r_1, i_2, r_2, i_3, r_3\}$. The meaning of each fact and operator is the same as in Example 5.1. The left illustration in Figure 5.4 depicts the initial state and goal conditions.

Figure 5.4 also illustrates an exact BDD $\mathcal{B}$ for Π_3^+ (left diagram). The dashed arrows represent zero-arcs (i.e., $v_a = 0$) and the solid arrows one-arcs (i.e., $v_a = 1$). On the left-hand-side are the operators associated with each layer. Notice that there is a one-to-one association between sr-plans and $\mathbf{r}-\mathbf{t}$ paths. □

Similarly to the DFP task case, we employ the iterative refinement procedure to construct our relaxed BDD (see Algorithm 3). The following sections explain the main components to build our relaxed BDD and also discuss the importance of the operator ordering to obtained smaller exact BDDs.

5.5.1 BDD Relaxed States, Filtering, and Bound Computation

We now define the relaxed states and filtering rules for our relaxed BDD construction procedure. We also present the bound computation over the diagram, which in this case corresponds to a simple shortest-

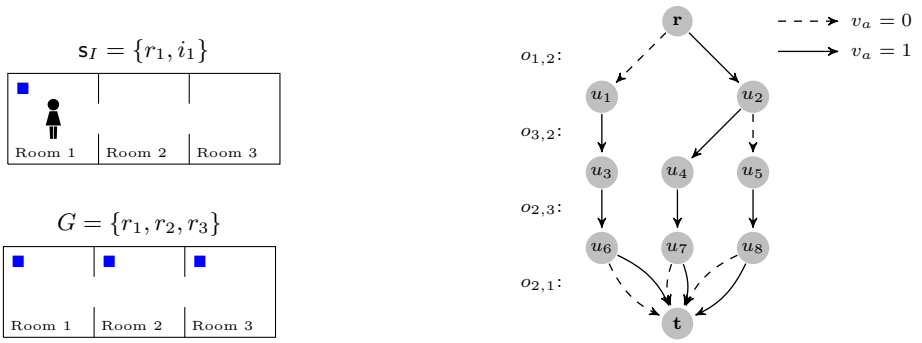


Figure 5.4: The three-room running example and its corresponding exact BDD.

path algorithm.

Achieved and Needed Relaxed States. We now introduce the relaxed states of our BDD encoding based on the achieved and needed states of R-DFP. To simplify the exposition, we will consider that all zero-arcs ($v_a = 0$) are associated to a dummy operator $\mathsf{o}(a) = o_{\mathsf{noop}}$ with empty add effects and preconditions, $\mathsf{add}(o_{\mathsf{noop}}) = \mathsf{pre}(o_{\mathsf{noop}}) = \emptyset$.

For each node $u \in \mathcal{N}$, we store the facts that are achieved by all $\mathbf{r} - u$ paths, $A_{\text{all}}(u)$, and the facts achieved by some $\mathbf{r} - u$ path, $A_{\text{some}}(u)$. These sets have the same meaning as in the MDD case and are computed in the same fashion (see Section 5.3.1). In addition, each node $u \in \mathcal{N}$ stores the set of facts that are needed by all and at least one $\mathbf{r} - u$ paths, $N_{\text{all}}(u)$ and $N_{\text{some}}(u)$, respectively. At the root node $N_{\text{all}}(\mathbf{r}) = N_{\text{some}}(\mathbf{r}) = L^F$, and for any other node $u \in \mathcal{N}$ we have

$$N_{\text{all}}(u) = \bigcap_{a \in \mathcal{A}^{\text{in}}(u)} (N_{\text{all}}(s(a)) \cup \mathsf{pre}(\mathsf{o}(a))) \quad \text{and} \quad N_{\text{some}}(u) = \bigcup_{a \in \mathcal{A}^{\text{in}}(u)} (N_{\text{some}}(s(a)) \cup \mathsf{pre}(\mathsf{o}(a))) \,.$$

For the bottom-up information of node $u \in \mathcal{N}$, sets $A^{\uparrow}_{\text{all}}(u)$ and $N^{\uparrow}_{\text{all}}(u)$ correspond to the facts achieved and needed by all $u - \mathbf{t}$ paths, respectively. Similarly, sets $A^{\uparrow}_{\text{some}}(u)$ and $N^{\uparrow}_{\text{some}}(u)$ represent the facts achieved and needed by some $u - \mathbf{t}$ path, respectively. Starting at $\mathbf{t}$ with $A^{\uparrow}_{\text{all}}(\mathbf{t}) = A^{\uparrow}_{\text{some}}(\mathbf{t}) = N^{\uparrow}_{\text{all}}(\mathbf{t}) = N^{\uparrow}_{\text{some}}(\mathbf{t}) = \emptyset$, we instantiate $A^{\uparrow}_{\text{some}}(u)$ and $N^{\uparrow}_{\text{some}}(u)$ for each node $u \in \mathcal{N}$ as described in Section 5.3.1, and set $A^{\uparrow}_{\text{all}}(u)$ and $N^{\uparrow}_{\text{all}}(u)$ as

$$A^{\uparrow}_{\text{all}}(u) = \bigcap_{a \in \mathcal{A}^{\text{out}}(u)} \left(A^{\uparrow}_{\text{all}}(t(a)) \cup \mathsf{add}(\mathsf{o}(a))\right) \quad \text{and} \quad N^{\uparrow}_{\text{all}}(u) = \bigcap_{a \in \mathcal{A}^{\text{out}}(u)} \left(N^{\uparrow}_{\text{all}}(t(a)) \cup \mathsf{pre}(\mathsf{o}(a))\right) .$$

Notice that the BDD keeps track of more fact sets than the MDD. The latter can avoid storing top-down information of the needed facts because it represents the sequencing aspect of the problem. In contrast, as the BDD only encodes sr-plans, we require the needed facts in both directions to identify invalid paths. Proposition 5.5 shows two filtering rules to eliminate arcs where the needed facts are not achieved. This two rules are similar to the MDD rules DFP-R1 and DFP-R2.

Proposition 5.5. *An arc $a \in \mathcal{A}$ can be removed from $\mathcal{B}$ if either of the following conditions hold:*

$$\mathit{pre}(\mathsf{o}(a)) \not\subseteq A_{\text{some}}(s(a)) \cup A^{\uparrow}_{\text{some}}(t(a)), \tag{SR-R1}$$

$$N_{\text{all}}(s(a)) \cup N^{\uparrow}_{\text{all}}(t(a)) \subseteq A_{\text{some}}(s(a)) \cup \mathit{add}(\mathsf{o}(a)) \cup A^{\uparrow}_{\text{some}}(t(a)). \tag{SR-R2}$$

Proof. Consider a path $p = (a_1, \ldots, a_m) \in \mathcal{B}$ that corresponds to an sr-plan and any arc $a \in p$. From the relaxed states definitions, we know that $\bigcup_{a' \in p \setminus \{a\}} \mathsf{add}(\mathsf{o}(a')) \subseteq A_{\text{some}}(s(a)) \cup A^{\uparrow}_{\text{some}}(t(a))$, and $N_{\text{all}}(s(a)) \cup N^{\uparrow}_{\text{all}}(t(a)) \subseteq \bigcup_{a' \in p \setminus \{a\}} \mathsf{pre}(\mathsf{o}(a')) \cup L^F$. Since $\mathsf{pre}(o) \cap \mathsf{add}(o) = \emptyset$ for all operators $o \in O$, rules SR-R1 and SR-R2 do not eliminate arc $a \in p$. ■

We develop an additional filtering rules to avoid redundant plans, i.e., we eliminate arcs that corresponds to operators that achieve no new fact. Thus, an arc $a \in \mathcal{A}$ can be eliminated from $\mathcal{B}$ if the following condition holds

$$(\mathsf{add}(\mathsf{o}(a)) \setminus \mathsf{s}_I) \cap \left(N_{\text{some}}(s(a)) \cup N^{\uparrow}_{\text{some}}(t(a))\right) \neq \emptyset. \tag{SR-R3}$$

The validity of the above rule follows from the relaxed states definitions. Notice that rule SR-R3 can be strengthened to remove all sr-plans with redundant operators as shown in SR-R3A. However, this rule can potentially remove some minimal cost-optimal plans. We decided to use SR-R3 in our implementation to guarantee that all minimal cost-optimal plans are represented in the BDD.

$$\left(\mathsf{add}(\mathsf{o}(a)) \setminus (A_{\text{all}}(s(a)) \cup A^{\uparrow}_{\text{all}}(t(a)))\right) \cap \left(N_{\text{some}}(s(a)) \cup N^{\uparrow}_{\text{some}}(t(a))\right) \neq \emptyset. \tag{SR-R3A}$$

Bound Computation. Recall that the immediate cost function of R-SR is state-independent, thus, we can utilize the shortest path procedure introduced in Section 2.2.4 to compute a lower bound. Each node $u \in \mathcal{N}$ stores the cost of the shortest $\mathbf{r} - u$ and $u - \mathbf{t}$ path, $c(u)$ and $c^{\uparrow}(u)$, respectively. Starting with $c(\mathbf{r}) = c^{\uparrow}(\mathbf{t}) = 0$, the top-down and bottom-up costs of $u \in \mathcal{N}$ are

$$c(u) = \min_{a \in \mathcal{A}^{\text{in}}(u)} \{c(s(a)) + \ell_a\} \qquad \text{and} \qquad c^{\uparrow}(u) = \min_{a \in \mathcal{A}^{\text{out}}(u)} \left\{c^{\uparrow}(t(a)) + \ell_a\right\}.$$

The shortest path information is used both for the BDD heuristic computation (see Section 5.6) and to identify and eliminate sub-optimal plans. Given a valid plan π' for task Π^+, we can remove arc $a \in \mathcal{A}$ from $\mathcal{B}$ if

$$c(s(a)) + \ell_a + c^{\uparrow}(t(a)) > \mathsf{cost}(\pi'). \tag{SR-R4}$$

Notice that the above condition is equivalent to the cost-based filtering rule CB-R1 from in Chapter 2.

5.5.2 BDD Splitting Algorithm

We now present our splitting procedure that aims to split nodes so the resulting BDD is exact (i.e., all paths are sr-plans) if the maximum width $\mathcal{W}$ is large enough. Our approach takes advantage of the DFP characteristics to create relaxed BDDs with tight worst cases on the maximum width needed per layer. We start by defining the exact information of a node.

Definition 5.2. *Consider a node $u \in \mathcal{N}$ and a fact $f \in F$.*

(i) Fact $f \in F$ is A-exact in node u if all $\mathbf{r}-u$ paths add f or none do, i.e., $f \in A_{\text{all}}(u)$ or $f \notin A_{\text{some}}(u)$, respectively.

(ii) Fact $f \in F$ is N-exact in node u if either all $\mathbf{r} - u$ paths require f or none do, i.e., $f \in N_{\text{all}}(u)$ or $f \notin N_{\text{some}}(u)$, respectively.

Algorithm 11 Relaxed BDD Split Nodes Procedures

1: **procedure** SplitDDNodes($\mathcal{N}_i$, $\mathcal{W}$)
2: Initialize Q with all the facts in $F_{\neg \mathtt{s}_I}$
3: **while** $|Q| > 0$ **and** $|\mathcal{N}_i| < \mathcal{W}$ **do**
4: Remove first fact f in Q
5: **if** $i \leq \gamma(f) + 1$ **then**
6: **for** $u \in \mathcal{N}_i$ **do**
7: **if** $f \in A_{\text{some}}(u)$ **and** $f \notin A_{\text{all}}(u)$ **then**
8: SplitBDDNodeAchieved(u, f)
9: **if** $|\mathcal{N}_i| = \mathcal{W}$ **then return**
10: **for** $u \in \mathcal{N}_i$ **do**
11: **if** $f \in N_{\text{some}}(u)$ **and** $f \notin N_{\text{all}}(u)$ **and** $f \notin A_{\text{all}}(u)$ **then**
12: SplitBDDNodeNeeded(u, f)
13: **if** $|\mathcal{N}_i| = \mathcal{W}$ **then return**

Algorithm 11 illustrates the SplitDDNodes procedure. The algorithm receives a node layer and the width limit, $\mathcal{W}$. The procedure iterates over a priority queue of facts Q and splits nodes such that for each $f \in Q$, all nodes are A-exact and N-exact or the width limit is reached. As in the MDD case (see Section 5.3.2), goals have higher priority, followed by fact landmarks, and then the remaining facts. Algorithm 11 avoids splitting nodes with respect to fact f after layer index $\gamma(f) + 1$ (line 5), a valid condition explained in Proposition 5.7.

Algorithm 12 shows how to split any node $u \in \mathcal{N}$ with respect to a fact $f \in F$. The procedure iterates over the incoming arcs of u and redirects the arcs to a new node u' accordingly. If f is A-exact (respectively, N-exact) in all nodes in the previous layer, our splitting procedure guarantees that f will be A-exact in u (respectively, N-exact).

Algorithm 12 Relaxed BDD Split Single Node Procedures

procedure SplitBDDNodeAchieved(u, f)
Create a new node u' and update $\mathcal{N}_i = \mathcal{N}_i \cup \{u'\}$
$\delta_{\texttt{achieved}} = \{a \in \mathcal{A}^{\text{in}}(u) : \ f \in A_{\text{all}}(s(a))$ **or** $f \in \mathsf{add}(\mathsf{o}(a))\}$
Redirect arcs: $\mathcal{A}^{\text{in}}(u') = \delta_{\texttt{achieved}}$ and $\mathcal{A}^{\text{in}}(u) = \mathcal{A}^{\text{in}}(u) \setminus \delta_{\texttt{achieved}}$
if $\mathcal{A}^{\text{in}}(u') \neq \emptyset$ **then** Duplicate outgoing arcs from u to u'

procedure SplitBDDNodeNeeded(u, f)
Create a new node u' and update $\mathcal{N}_i = \mathcal{N}_i \cup \{u'\}$
$\delta_{\texttt{needed}} = \{a \in \mathcal{A}^{\text{in}}(u) : \ f \in N_{\text{all}}(s(a))$ **or** $f \in \mathsf{pre}(\mathsf{o}(a))\}$
Redirect arcs: $\mathcal{A}^{\text{in}}(u') = \delta_{\texttt{needed}}$ and $\mathcal{A}^{\text{in}}(u) = \mathcal{A}^{\text{in}}(u) \setminus \delta_{\texttt{needed}}$
if $\mathcal{A}^{\text{in}}(u') \neq \emptyset$ **then** Duplicate outgoing arcs from u to u'

Proposition 5.6. *Consider a BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ such that for each $f \in F$ and node $u \in \mathcal{N}$, f is A-exact in u and f is N-exact in u when $f \notin A_{\text{all}}(u)$. Then, rules SR-R1 and SR-R2 are sufficient to remove all invalid sr-plan paths in $\mathcal{B}$.*

Proof. Consider a path $p \in \mathcal{B}$ and a fact $f \in F_{\neg \mathtt{s}_I}$ such that either (i) there exists an operator $o \in p$ with $f \in \mathsf{pre}(o)$ or (ii) $f \in L^F$ but for all $a' \in p$, $f \notin \mathsf{add}(a')$. Take the last arc $a \in p$, i.e., $s(a) = u \in \mathcal{N}_n$ and $t(a) = \mathbf{t}$. Since p is an invalid sr-plan, $f \notin A_{\text{all}}(u)$ and $f \in N_{\text{all}}(u) \cup \mathsf{pre}(\mathsf{o}(a))$. Moreover, $f \notin A_{\text{some}}(u)$ since f is A-exact in u. Since $A^{\uparrow}_{\text{some}}(\mathbf{t}) = \emptyset$, either rule SR-R1 or SR-R2 will eliminate arc a and, therefore, remove p from $\mathcal{B}$. ■

Proposition 5.6 implies that for each fact $f \in F$ we need at most three nodes in each layer $\mathcal{N}_i$, i.e., a node $u \in \mathcal{N}_i$ where $f \in A_{\text{all}}(u)$, a node $u' \in \mathcal{N}_i$ where $f \notin A_{\text{all}}(u')$ and $f \in N_{\text{all}}(u')$, and a node $u'' \in \mathcal{N}_i$ where $f \notin A_{\text{all}}(u'')$ and $f \notin N_{\text{all}}(u'')$. Since for all $f \in \mathsf{s}_I$ and $u \in \mathcal{N}$, $f \in A_{\text{all}}(u)$, there is no need to split nodes with respect to facts in the initial state. Similarly, for all facts $f \in L^F$ and nodes $u \in \mathcal{N}$, $f \in N_{\text{all}}(u)$, so each fact landmark needs at most two nodes in each layer. Then, a conservative estimate on the maximum width needed to construct an exact BDD is $O(3^{|F_{\neg \mathsf{s}_I, \neg L^F}|} \cdot 2^{|L^F{}_{\neg \mathsf{s}_I}|})$, where $L^F{}_{\neg \mathsf{s}_I} = L^F \setminus \mathsf{s}_I$ (i.e., all fact landmarks omitted in the initial state) and $F_{\neg \mathsf{s}_I, \neg L^F} = (F \setminus L^F) \setminus \mathsf{s}_I$ (i.e., all facts omitted in the initial state that are not landmarks).

Example 5.5 Consider our running example task Π^+_3 with three rooms. Figure 5.5 illustrates some of the steps of the BDD construction procedure. The right diagram corresponds to a the width-one BDD. The middle diagram illustrates the SplitDDNodes procedure over $\mathcal{N}_2$ and the filtering procedure by a gray arc (eliminated by Rule SR-R2). Lastly, the right diagram is the resulting BDD. □

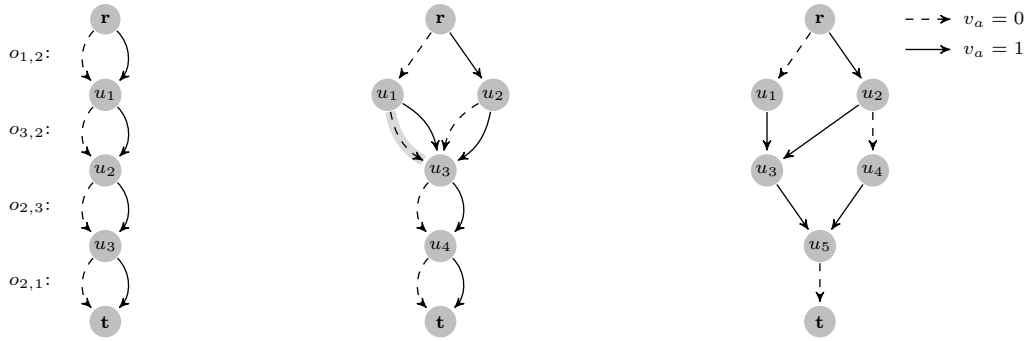


Figure 5.5: BDD construction procedure with $\mathcal{W} = 2$.

Even though the proposed splitting approach is valid, we prove that it is possible to create an exact BDD where not all nodes are A-exact or N-exact. Given a BDD $\mathcal{B}$ and a fact $f \in F$, we define the last layer of f, $\gamma(f)$, as the maximum arc layer index at which an operator either adds or requires f, i.e., for $i = \gamma(f)$, $f \in \mathsf{add}(o_i) \cup \mathsf{pre}(o_i)$ and for all $j > \gamma(f)$, $f \notin \mathsf{add}(o_j) \cup \mathsf{pre}(o_j)$.

Proposition 5.7. *Consider a BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ such that for each $f \in F$ and node $u \in \mathcal{N}_i$, with $i \leq \gamma(f) + 1$, f is A-exact in u and f is N-exact in u when $f \notin A_{\text{all}}(u)$. Then, rules SR-R1 and SR-R2 are sufficient to remove all invalid sr-plan paths in $\mathcal{B}$.*

Proof. Consider a path $p \in \mathcal{B}$ with a fact $f \in F_{\neg \mathsf{s}_I}$ such that $f \in L^F$ or for some operator $o \in p$, $f \in \mathsf{pre}(o)$ but f is not added by any operator in p. Now take arc $a \in p$ in the last layer of f, i.e., $s(a) \in \mathcal{N}_{\gamma(f)}$ and $t(a) \in \mathcal{N}_{\gamma(f)+1}$. By definition of last layer, $f \notin A^{\uparrow}_{\text{some}}(t(a))$ and $f \notin N^{\uparrow}_{\text{some}}(t(a))$. By hypothesis over p, $f \notin A_{\text{all}}(s(a))$ (and, $f \notin A_{\text{some}}(s(a))$) and $f \in N_{\text{all}}(s(a)) \cup \mathsf{pre}(\mathsf{o}(a))$. Then, either rule SR-R1 or SR-R2 will remove arc a and, hence, the invalid sr-plan path p. ■

Algorithm 11 uses Proposition 5.7 to avoid splitting nodes with respect to a fact $f \in Q$ when the current layer is greater than $\gamma(f)$ (line 5), and avoids splitting nodes $u \in \mathcal{N}$ if $f \in A_{\text{all}}(u)$ (line 11).

5.5.3 Maximum BDD Width and Operator Ordering

This section presents an upper bound for the maximum width needed in each layer of an exact BDD. We show how these bounds depend on the operator-layer assignment and develop a simple heuristic

procedure to create good operator-layer orderings.

For a given $f \in F_{\neg \mathsf{s}_I}$, consider the first layer where f is either added or needed, i.e., $\phi(f) = \min\{i \in \{1, ..., n\} : f \in \mathsf{add}(o_i) \cup \mathsf{pre}(o_i)\}$. We define $\psi(i)$ as the set of facts that need to be considered for splitting in layer $\mathcal{N}_i$, i.e., $\psi(i) = \{f \in F_{\neg \mathsf{s}_I} : \phi(f) + 1 \leq i \leq \gamma(f) + 1\}$. In particular, let $\psi_{L^F}(i) = \psi(i) \cap L^F$ be the set of fact landmarks that need splitting in layer $\mathcal{N}_i$, and $\psi_F(i) = \psi(i) \setminus L^F$ be the set of non-fact landmarks that need splitting in layer $\mathcal{N}_i$.

Corollary 5.1. *Consider a DFP task Π^+ and an exact BDD $\mathcal{B}$ constructed as described above. The maximum width of layer $\mathcal{N}_i$ is $O(2^{|\psi_{L^F}(i)|} \cdot 3^{|\psi_F(i)|})$. Then, an upper bound on the maximum width for $\mathcal{B}$ is given by $O(\max_{i \in \{1,...,n\}}\{2^{|\psi_{L^F}(i)|} \cdot 3^{|\psi_F(i)|}\})$.*

Proof. Follows directly from Proposition 5.7. ■

Notice that $\psi(i)$ depends on how we order the operators when we assign them to layers. In particular, we would like to minimize the number of facts that need to be split in every layer, i.e., find an ordering of facts Λ such that $\max_{i \in \{1,...,n\}}\{2^{|\psi_{L^F}(i)|} \cdot 3^{|\psi_F(i)|}\}$ is minimized. This NP-hard problem has been studied for knapsack constraints (Behle, 2008) and for the set covering and independent set problems (Bergman et al., 2011, 2012). We develop a simple operator ordering heuristic that takes advantage of the following proposition.

Proposition 5.8. *Consider a DFP task Π^+ and a relaxed BDD $\mathcal{B}$ constructed as described above with operator ordering Λ. Assume that for a given $f \in F_{\neg \mathsf{s}_I, \neg L^F}$ all operators that add f are ordered before all operators that require f in Λ. Then, it is sufficient to have $\mathcal{W} = 2$ to guarantee that all paths $p \in \mathcal{B}$ that require f have an operator that adds f.*

Proof. Since all operators that add f are ordered first, we need two nodes in a layer to ensure that f is A-exact in each node. Consider the first layer $\mathcal{N}_i$ such that o_i requires f. Take a node $u \in \mathcal{N}_j$ $(j > i)$. If $f \in A_{\text{all}}(u)$, u does not need to be N-exact (Proposition 5.7). If $f \notin A_{\text{some}}(u)$, rules SR-R1 and SR-R2 eliminate arcs associated to operators that require f, so no splitting is required. ■

Algorithm 13 BDD Operator Ordering

```
1:  procedure OperatorOrdering(O, Q)
2:      while |Q| > 0 and |O| > 0 do
3:          Remove first fact f in Q
4:          for o ∈ O do
5:              if f ∈ add(o) then
6:                  Add o to Λ and remove o from O
7:          if f ∉ L^F then
8:              for o ∈ O do
9:                  if f ∈ pre(o) then
10:                     Add o to Λ and remove o from O
11:     return Λ
```

Our OperatorOrdering procedure (Algorithm 13) receives the set of operators and a priority queue of facts Q. In our implementation, we use the same fact priority queue used in the SplitDDNodes procedure (Algorithm 11). Then, the operator ordering is as follows: for each fact $f \in Q$ we insert operators $o \in O \setminus \Lambda$ that add f into Λ (lines 4-6) and then operators $o \in O \setminus \Lambda$ that require f (lines 8-10). Notice that when iterating over a fact landmark f, we omit including operators that require f since f is needed in all sr-plan. However, these operators will be added later on when iterating over other facts.

5.6 Relaxed BDD-based Heuristic

We now present our relaxed BDD heuristic for a DFP task Π^+ and show its admissibility and consistency. Given any reachable state s, we can construct a relaxed BDD $\mathcal{B}$ for s updating the initial state $\mathsf{s}_I = \mathsf{s}$. Then, the relaxed BDD heuristic $h^{\mathcal{B}}(\mathsf{s})$ corresponds to the shortest path in $\mathcal{B}$, i.e., $h^{\mathcal{B}}(\mathsf{s}) = c(\mathbf{t})$.

Theorem 5.4. *Consider a DFP task Π^+, a state s, and a relaxed BDD $\mathcal{B}$ for s with $\mathcal{W} \geq 1$ constructed using Algorithm 3. Then, $h^{\mathcal{B}}(\mathsf{s})$ given by the shortest path in $\mathcal{B}$ is admissible.*

Proof. Filtering rules SR-R1 and SR-R2 guarantee that no sr-plan is eliminated, while rules SR-R3 and SR-R4 guarantee the presence of at least one cost-optimal plan. Then, $h^{\mathcal{B}}(\mathsf{s}) \leq \mathsf{cost}(\pi^*_{\mathsf{sr}}) \leq h^+(\mathsf{s})$, where π^*_{sr} is the cost-optimal sr-plan from s and h^+ the perfect heuristic for Π^+. ■

As in the MDD case, our implementation constructs a relaxed BDD $\mathcal{B}$ for s_I and updates $\mathcal{B}$ during search. Given a state s and the sequence of operators to reach s from s_I, $\pi_{\mathsf{s}} = (o_1, .., o_k)$, we update $\mathcal{B}$ by removing all arcs $a \in \mathcal{A}$ with $\mathsf{o}(a) \in \pi_{\mathsf{s}}$ and $v_a = 0$. We then iteratively apply the top-down and bottom-up procedures over $\mathcal{B}$ and compute the heuristic as:

$$h^{\mathcal{B}}(\mathsf{s}) = \begin{cases} c(\mathbf{t}) - \mathsf{cost}(\pi_{\mathsf{s}}), & \pi_{\mathsf{s}} \in \mathcal{B}, \\ \infty, & \text{otherwise}, \end{cases} \tag{5.5}$$

where $\pi_{\mathsf{s}} \in \mathcal{B}$ represents that there exists a path $p \in \mathcal{B}$ with an arc $a \in p$ with $v_a = 1$ and $\mathsf{o}(a) = o$ for all $o \in \pi_{\mathsf{s}}$. As in the MDD case (see Section 5.4), $h^{\mathcal{B}}$ computed via (5.5) is a globally admissible heuristic (Karpas and Domshlak, 2012), which is sufficient to guarantee that a best-first search will find an optimal solution. In addition, (5.5) computes a consistent heuristic if neither the operator nor fact order changes when updating $\mathcal{B}$ for any state s (Theorem 5.5). As previously noted, the main reason for the consistency of $h^{\mathcal{B}}$ is that the successor state BDD is a subset of the current state BDD.

Theorem 5.5. *Consider a delete-free task Π^+, a state s, and a relaxed BDD $\mathcal{B}$ with $\mathcal{W} \geq 1$ constructed using Algorithm 3. Then, $h^{\mathcal{B}}(\mathsf{s})$ given by (5.5) is consistent.*

Proof. Consider a state s, an applicable operator o, and its successor state $\mathsf{s}' = \mathsf{succ}(\mathsf{s}, o)$. Given the relaxed BDD $\mathcal{B}_{\mathsf{s}_I}$ for s_I, let $\mathcal{B}_{\mathsf{s}}$ and $\mathcal{B}_{\mathsf{s}'}$ be the updated BDDs for state s and s', respectively. Since each BDD is updated from $\mathcal{B}_{\mathsf{s}_I}$ without changing the operator and fact order, every path $p \in \mathcal{B}_{\mathsf{s}'}$ is also in $\mathcal{B}_{\mathsf{s}}$. Then, we have $c(\mathbf{t}_{\mathsf{s}}) \leq c(\mathbf{t}_{\mathsf{s}'})$.

We know that $h^{\mathcal{B}}(\mathsf{s}) = c(\mathbf{t}_{\mathsf{s}}) - \mathsf{cost}(\pi_{\mathsf{s}})$ and $h^{\mathcal{B}}(\mathsf{s}') = c(\mathbf{t}_{\mathsf{s}'}) - \mathsf{cost}(\pi_{\mathsf{s}}) - \mathsf{cost}(o)$. Then, $h^{\mathcal{B}}(\mathsf{s}) - \mathsf{cost}(o) - h^{\mathcal{B}}(\mathsf{s}') = c(\mathbf{t}_{\mathsf{s}}) - c(\mathbf{t}_{\mathsf{s}'}) \leq 0$, and so $h^{\mathcal{B}}(\mathsf{s}) \leq \mathsf{cost}(o) + h^{\mathcal{B}}(\mathsf{s}')$. ■

5.6.1 Relationship with Disjunctive Landmarks

The set of fact landmarks L^F plays a key role on the relaxed BDD construction, specifically on the splitting and operator ordering algorithms. We show that the accuracy of our heuristic is linked to both L^F and the heuristics based on disjunctive operator landmarks.

Given a state s, a *disjunctive operator landmark* (Zhu and Givan, 2003; Helmert and Domshlak, 2009) is a set of operators $D \subseteq O$ such that at least one operator in D must be present in any plan from state s. Notice that each fact landmark $f \in L^F \setminus \mathsf{s}$ defines a disjunctive operator landmark $O(f) = \{o \in O : f \in \mathsf{add}(o)\}$, i.e., any plan from s needs an operator that adds fact f. In fact, any subset of fact landmarks $L \subseteq L^F \setminus \mathsf{s}$ defines a set of disjunctive landmarks $\mathcal{L} = \{O(f) : f \in L\}$.

The disjunctive operator landmark problem is defined as follows: given a set of disjunctive landmarks $\mathcal{L}$, we look for a minimum-cost set of operators such that there is one operator for each disjunctive operator landmark $D \in \mathcal{L}$. The MILP model $\mathcal{D}_{MILP}$ represents this hitting set problem (Bonet and Helmert, 2010) for any disjunctive landmark set $\mathcal{L}$, where $x_o \in \{0,1\}$ is binary variable representing if operator $o \in O$ is chosen or not. We will consider the optimal value of this problem to be the perfect disjunctive landmark heuristic $h^{\mathcal{L}}$. Notice that the hitting set problem is an NP-hard problem (Karp, 1972), so relaxations of the problem (e.g., LP relaxation) are often used as heuristics.

$$
\begin{aligned}
h^{\mathcal{L}} = \min \quad & \sum_{o \in O} \mathsf{cost}(o) x_a && && (\mathcal{D}_{MILP}) \\
\text{s.t.} \quad & \sum_{o \in D} x_a \geq 1 && \forall D \in \mathcal{L} \\
& x_o \in \{0,1\} && \forall o \in O
\end{aligned}
$$

Proposition 5.9 shows that the relaxed BDD heuristic is highly related to $h^{\mathcal{L}}$ when set $\mathcal{L}$ is defined over a subset of fact landmarks $L \subseteq L^F$. In fact, $h^{\mathcal{B}}$ dominates $h^{\mathcal{L}}$ if all the facts $f \in L$ are A-exact in all the BDD nodes.

Proposition 5.9. *Consider a planning task Π^+, a reachable state s, a relaxed BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ for s and a set of fact landmarks L^F for s. Let $L \subseteq (L^F \setminus \mathsf{s})$ be a subset of fact landmarks such that each fact $f \in L$ is A-exact in every node $u \in \mathcal{N}$. Then, $h^{\mathcal{L}}(\mathsf{s}) \leq h^{\mathcal{B}}(\mathsf{s})$ where $\mathcal{L}$ is defined over L.*

Proof. For any $\mathbf{r} - \mathbf{t}$ path $p \in \mathcal{B}$, consider $\pi_{\mathsf{sr}}(p)$ as the set of operators associated with p, i.e., $o \in \pi_{\mathsf{sr}}(p)$ if and only if there exists an arc $a \in p$ with $v_a = 1$ and $\mathsf{o}(a) = o$. We know that in every node $u \in \mathcal{N}$ each fact $f \in L$ is A-exact and also N-exact by definition ($N_{\text{all}}(\mathbf{r}) = L^F$). Then, for each path $p \in \mathcal{B}$ and $f \in L$, there exists at least one operator $o \in \pi_{\mathsf{sr}}(p)$ that is also in $O(f)$. Hence, the set of operators $\pi_{\mathsf{sr}}(p)$ for any $p \in \mathcal{B}$ is a feasible solution for $\mathcal{D}_{MILP}$, where $\mathcal{L} = \{O(f) : f \in L\}$. It follows that $h^{\mathcal{L}}(\mathsf{s}) \leq \mathsf{cost}(\pi_{\mathsf{sr}}(p))$ for all $p \in \mathcal{B}$, therefore $h^{\mathcal{L}}(\mathsf{s}) \leq h^{\mathcal{B}}(\mathsf{s})$. ■

5.7 Relaxed MDD and BDD Comparison

We now summarize the main differences in our relaxed MDD and BDD approaches and present guidelines on when to use each technique. Table 5.1 compares the graphical structures in terms of encoding, size, and heuristic. As presented in Section 5.3, our relaxed MDD $\mathcal{M}$ encodes the DFP task, where paths in $\mathcal{M}$ correspond to sequences of operators. In contrast, our relaxed BDD $\mathcal{B}$ represents the sequential relaxation of a DFP task and, hence, paths in $\mathcal{B}$ correspond to sets of operators (see Section 5.5). These differences affect the meaning of arcs and nodes. While an MDD arc in layer i represents the i-th operator in the sequence, a BDD arc in the same layer encodes the decision of selecting the associated operator. Also, nodes in $\mathcal{M}$ represent the aggregation of planning states, while nodes in $\mathcal{B}$ are aggregated sets of achieved and needed facts.

The encoding difference translates into a size difference. The number of layers in $\mathcal{M}$ is an upper bound on the number of operators in a cost-optimal plan, $n \geq |O|$, while $\mathcal{B}$ has exactly $m = |O|$ layers. Hence, given a maximum width $\mathcal{W}$, $\mathcal{M}$ has at most $n \cdot \mathcal{W}$ nodes and $\mathcal{B}$ has at most $|O| \cdot \mathcal{W}$. Since the maximum number of arcs emanating from an MDD node is $|O_0| = |O| + 1$, the maximum number of arcs in $\mathcal{M}$ is $n \cdot \mathcal{W} \cdot (|O| + 1)$. In contrast, $\mathcal{B}$ has at most $2 \cdot \mathcal{W} \cdot |O|$ arcs because each node has at most

Table 5.1: Relaxed DDs comparison

		Relaxed MDD $\mathcal{M}$	Relaxed BDD $\mathcal{B}$
Encoding	Problem	DFP	Sequential relaxation
	Path	Sequence of operators (plan)	Set of operators (sr-plan)
	Arcs	Applicable operators	Selected operators
	Nodes	Aggregated states	Aggregated achieved & needed facts
Size	$\lvert\mathcal{N}\rvert$	$O(n \cdot \mathcal{W})$	$O(\mathcal{W} \cdot \lvert O\rvert)$
	$\lvert\mathcal{A}\rvert$	$O(n \cdot \mathcal{W} \cdot (\lvert O\rvert + 1))$	$O(2 \cdot \mathcal{W} \cdot \lvert O\rvert)$
	$\lvert\mathcal{N}\rvert + \lvert\mathcal{A}\rvert$	$O(n \cdot \mathcal{W} \cdot (\lvert O\rvert + 2))$	$O(3 \cdot \mathcal{W} \cdot \lvert O\rvert)$
Heuristic	Method	Critical path	Shortest path
	Relates to	h^{max} and planning graph	Disjunctive landmarks

two emanating arcs. Notice that, even though $\mathcal{M}$ usually has a smaller number of nodes than $\mathcal{B}$, the number of arcs in $\mathcal{M}$ is higher than in $\mathcal{B}$. Overall, as long as $n \geq 3$, $\mathcal{B}$ is smaller than $\mathcal{M}$ in terms of number of nodes and arcs.

We also use different methods to calculate the relaxed DD heuristics and, thus, each one relates to different techniques. Since $h^{\mathcal{M}}$ is computed by a critical path procedure over $\mathcal{M}$, it has a strong relationship with critical path heuristics and the planning graph (see Section 5.4). On the other hand, $h^{\mathcal{B}}$ corresponds to the value of shortest path over $\mathcal{B}$ and it relates to disjunctive landmark heuristics (see Section 5.6).

Lastly, these differences give us guidelines on when to use each graphical structure. Relaxed MDDs should be preferred when the sequential aspect of the problem is predominant and the number of operators is small. In contrast, relaxed BDDs are suited for domains where the sequential relaxation is a good approximation to the original DFP task and there is a high number of fact landmarks. Our empirical results support the validity of these guidelines (see Section 5.10).

5.8 Exploiting the Relaxed DD Structure

Besides their use to compute admissible heuristics, we can exploit the graphical structure of relaxed MDDs and BDDs to reduce the search-space and dynamically improve their filtering rules. In the following, we explain how to extract plans from both graphical structures. We also show how we can leverage the BDD structure to find operator landmarks and identify redundant operators.

5.8.1 Plan Extraction Procedures

One of the main advantages of representing all the cost-optimal plans in a graphical structure, either a relaxed MDD or BDD, is that we can extract plans by traversing the graph. The cost of such plans can be used both to improve the cost-based filtering rules (i.e., DFP-R4 and SR-R4) and also to avoid sub-optimal states during search (see Section 5.9).

Given a reachable state s and its relaxed MDD $\mathcal{M}_{\mathsf{s}}$, Algorithm 14 shows our plan extraction procedure. The algorithm starts from an empty plan and greedily chooses a path in the MDD that has the potential to be a plan. Specifically, the algorithm starts in the root node $\mathbf{r} \in \mathcal{N}$ and iterates over each layer choosing an outgoing arc from the current node (lines 3-4). The SelectArcMDD procedure iterates over all the outgoing arcs of a node u and selects an arc a such that the corresponding operator v_a is applicable

Algorithm 14 MDD Plan Extraction Procedure

```
1: procedure MDDPlanExtraction(M_s, s)
2:     π = ∅, u = r
3:     for i ∈ {1, ..., n} do
4:         a = SelectArcMDD(s, M_s, u)
5:         Update node u = t(a) and current state s = s ∪ add(v_a)
6:         Add operator v_a to plan π
7:         if G ⊆ s then return π
8:     return ∅
```

in s and adds a new useful fact. We give priority to operators that add a higher number of new facts, in particular, if they add fact landmarks. The algorithm then updates the current node, state, and plan using the selected arc (lines 5-6). The procedure ends when the current state s is a goal state or it has iterated over all the layers.

Algorithm 15 BDD Plan Extraction Procedure

```
1: procedure BDDPlanExtraction(B_s, s)
2:     π = ∅
3:     while G ⊈ s do
4:         UpdatePathsBDD(π, B_s)
5:         L_O = FindApplicableOperators(s)
6:         o = SelectOperatorBDD(L, B_s)
7:         if o = ∅ then return ∅
8:         Add operator o to π and update state s = s ∪ add(o)
9:     return π
```

Since the relaxed BDD ignores the sequential aspect of the problem, we use a different approach to exact plans shown in Algorithm 15. Given a state s and its relaxed BDD $\mathcal{B}_\mathsf{s}$, the procedure starts with an empty plan π (line 2) and adds operators to π until all goals are satisfied. In each iteration, the procedure updates $\mathcal{B}_\mathsf{s}$ by keeping only the paths that have all the operators in π (line 4). Then, the procedure looks for all applicable operators in s that add at least one new fact, stores them in a list L_O, and uses $\mathcal{B}_\mathsf{s}$ to select the most promising operator (lines 5-6). Specifically, given a list of operators L_O, we greedily look for the operator o' that has a path in $\mathcal{B}_\mathsf{s}$ with the minimum cost, i.e.,

$$o' = \arg\min_{o \in L_O} \{c(s(a)) + \ell_a + c^{\uparrow}(t(a)) : a \in \mathcal{A}, \mathsf{o}(a) = o, v_a = 1\}.$$

Notice that the extraction procedure ends if SelectOperatorBDD does not select any operator, i.e., it returns an empty set (line 7). Otherwise, we include operator o to π and update state s (line 8).

5.8.2 Relaxed BDDs for Operator Pre-Processing

We also develop a simple procedure to identify cost-optimal operator landmarks and redundant operators using a relaxed BDD $\mathcal{B}$. Given a planning task Π^+ and a reachable state s, we say that an operator $o \in O$ is a *cost-optimal operator landmark* if every cost-optimal plan from s has operator o. Notice that the cost-optimal operator landmark definition is more restrictive than the operator landmark definition since it only considers cost-optimal plans. Any operator landmark is a cost-optimal operator landmark but the inverse is not necessarily true.

Given a relaxed BDD $\mathcal{B}$ for a DFP task Π^+ we identify cost-optimal operator landmarks as follows.

Consider a layer $\mathcal{N}_i$ ($i \in \{1, ..., m\}$) such that all arcs $a \in \mathcal{A}$ emanating from $\mathcal{N}_i$ have value $v_a = 1$. Then, all cost-optimal plans have operator o_i, i.e., o_i is an operator landmark for any cost-optimal plan. Similarly, we identify redundant operators if all arcs emanating from layer $\mathcal{N}_i$ have value $v_a = 0$. Then, no cost-optimal plan uses operator o_i, so o_i is a redundant operator that can be removed from O.

5.9 Implementation

In the following, we give a detailed explanation of our tree search algorithm and how it leverages the information from extracted plans. We also explain how the relaxed MDD and BDD are updated during search and the pre-processing procedures.

5.9.1 Binary Tree Search

We implement a branch-and-bound binary tree search procedure (Land and Doig, 1960), where our relaxed DD structures can be easily integrated. This type of search has had successful results when used with other admissible heuristics (Pommerening and Helmert, 2012) and optimization techniques (Imai and Fukunaga, 2015) that tackle cost-optimal DFP.

Our binary tree search implementation is a variant of the one used by Pommerening and Helmert (2012) where we use a best-first strategy instead of a depth-first. We define a state of our search as $S = \langle \mathsf{s}, \pi_{in}, \pi_{out}, c, h \rangle$ where s is a planning task state (i.e., set of facts), π_{in} and π_{out} correspond to the tree search decisions, c is the cost to reach S, and h the heuristic value. In particular, π_{in} is the set of operators that are currently in the plan and π_{out} the set of operators that are not allowed to be part of the plan.

Algorithm 16 illustrates the full binary tree search procedure. For notation purposes, we represent the components of a search state S as $S.\mathsf{s}$, $S.\pi_{in}$, $S.\pi_{out}$, $S.c$ and $S.h$. The algorithm starts with a single search state $S_0 = \langle \mathsf{s}_I, \emptyset, \emptyset, 0, 0 \rangle$ associated with the initial state when no decisions have been made (i.e., $S_0.\pi_{in} = S_0.\pi_{out} = \emptyset$). The cost of S_0 is 0, and the heuristic value is given by any admissible heuristic (e.g., $h^{\mathcal{M}}$ or $h^{\mathcal{B}}$). The algorithm keeps a priority queue of search states Q_S ordered in increasing value of $S.c + S.h$ with ties broken preferring higher values of $S.c$. In addition, the algorithm keeps track of the best plan found so far, π, and the best upper and lower bound (UB and LB, respectively) of the cost-optimal plan. Lines 2 to 6 initialize all of these values. Our specific implementation uses the FF procedure (Hoffmann and Nebel, 2001) to extract an initial plan (line 5).

The binary tree search procedure iterates over the states in Q_S until the queue is empty (line 7) or we can prove that our current plan (i.e., incumbent solution) is optimal (line 10). For each search state, we update the lower bound (line 9) and choose an operator to branch on (line 11). The ChooseOperator procedure looks for any non-forbidden operator that is applicable in the current state $S.\mathsf{s}$ and that adds at least one fact that is not in the current state (lines 26-29). If there is no operator that satisfies these conditions the algorithm returns an empty set (line 30). When ChooseOperator finds an operator, our search procedure creates two child states: one where the operator is forbidden and one where the operator is applied to the current state. If operator o has strictly positive cost, the first child is identical to its parent state with the exception that now o is in the forbidden set of operators (lines 14-16). In this case, we use the heuristic value of the parent state instead of computing the heuristic value again. Since zero-cost operators do not affect the cost of a plan, we avoid creating the first child if o is a zero-cost operator.

Algorithm 16 Binary Tree Search

```
1: procedure Binary Search(Π⁺)
2:     S_0 = ⟨s_I, ∅, ∅, 0, 0⟩
3:     S_0.h =ComputeHeuristic(S_0)
4:     Q_S = (S_0)
5:     π=GetInitialPlan(Π⁺)
6:     UB = cost(π), LB = S_0.h
7:     while |Q_S| > 0 do
8:         Retrieve first state S from Q_S
9:         if LB < S.c + S.h then LB = S.c + S.h
10:        if LB ≥ UB then return π
11:        o =ChooseOperator(S)
12:        if o = ∅ then continue
13:        % Create first child: operator is never selected %
14:        if cost(o) > 0 then
15:            S' = ⟨S.s, S.π_in, S.π_out ∪ {o}, S.c, S.h⟩
16:            Insert state S' to Q_S
17:        % Create second child: operator is selected %
18:        S'' = ⟨S.s ∪ add(o), S.π_in ∪ {o}, S.π_out, S.c + cost(o), 0⟩
19:        S''.h = ComputeHeuristic(S'')
20:        if S''.c + S''.h ≤ UB then
21:            Insert state S'' to Q_S
22:            π' = ExtractPlan(S'')
23:            if π' ≠ ∅ and cost(π') < cost(π) then
24:                π = π', UB = cost(π)
25:    return π

26: procedure ChooseOperator(S)
27:    for o ∈ O do
28:        if  o ∉ S.π_out and pre(o) ⊆ S.s and add(o) ⊈ S.s then
29:            return o
30:    return ∅
```

The second child state corresponds to the decision of including operator o to the current plan. As such, state $S''.\mathsf{s}$ considers the add effects of o, $S''.\pi_{in}$ includes o, and the cost and heuristic value are updated accordingly (line 18-19). If the cost estimation to a goal state is smaller or equal to our upper bound, we add the state into the queue (lines 20-21) and extract a plan. When the heuristic is given by either $h^{\mathcal{M}}$ or $h^{\mathcal{B}}$, we use their corresponding extraction plan procedures (Section 5.8). When using other heuristics this step can be simply checking that $S''.\pi_{in}$ is a plan. If the extracted plan has a smaller cost than our incumbent, we update π and our upper bound (lines 23-24).

5.9.2 Updating the BDD and MDD During Search

The computational effort to create a relaxed DD (MDD or BDD, accordingly) can be quite expensive depending on the number of operators and the chosen maximum width $\mathcal{W}$. To overcome the computational cost, our implementation creates a relaxed DD at the initial state and uses it to compute the heuristic for the other states in the search. For each search state S, we duplicate the initial relaxed DD and update it considering the tree search decisions, π_{in} and π_{out}. Hence, we keep two relaxed DD at all times during search: one for the initial state and one for the current search state S that it is been evaluated.

Given the initial state MDD $\mathcal{M}_{\mathsf{s}_I}$ and any state of the search S, $\mathcal{M}_{\mathsf{s}}$ is a copy of $\mathcal{M}_{\mathsf{s}_I}$ where we omit

all the arcs associated with operators in π_{out}. In addition, the first $k = |\pi_{in}|$ layers of $\mathcal{M}_{\mathbf{s}}$ have a single arc where arc a emanating from layer $\mathcal{N}_i$ $(i \leq k)$ is associated with the i-th operator in π_{in}. Similarly, the relaxed BDD for state S is a copy of $\mathcal{B}_{\mathbf{s}_I}$ where each arc a with $\mathtt{o}(a) \in \pi_{in}$ has value $v_a = 1$ and each arc a with $\mathtt{o}(a) \in \pi_{out}$ has a value $v_a = 0$.

Besides the computational gain, updating the relaxed DDs during search results in a consistent heuristic (see Sections 5.4 and 5.6). Moreover, our search algorithm remains complete since the relaxed DD heuristics only prune sub-optimal states.

5.9.3 Pre-processing Steps

Our planner includes a set of pre-processing tools to extract landmarks and identify redundant operators. We implement the same pre-processing algorithms used by Imai and Fukunaga (2014, 2015) since we mostly compare to their LP heuristic and MILP model. The landmark extraction algorithm is the same as the one described by Imai and Fukunaga (2015), which is a variant of existing procedures (Zhu and Givan, 2003; Keyder et al., 2010). The algorithm iterates over the operators and their add effects to identify fact landmark candidates and finally extract fact and operator landmarks.

We also include the Iterative Variable Elimination procedure by Imai and Fukunaga (2015) to identify redundant operators. The algorithm includes a Relevance Analysis that starts from the set of goals and iterates over the set of operators to identify relevant operators, i.e., operators that add goals or preconditions of other relevant operators. We also consider their Dominated Operator Elimination procedure that checks if an operator dominates another in terms of cost and add effects. Lastly, the authors present an Immediate Operator Application procedure for zero-cost operators, that, in our case, is implemented inside our binary tree search algorithm (Algorithm 16, line 14).

5.10 Empirical Evaluation

We now present an empirical study of our relaxed MDD and BDD heuristics. We analyze our heuristics' performance using different maximum widths $\mathcal{W} \in \{2, 4, 8, 16, 32, 64\}$ and compare their performance against a MILP model for DFP (Imai and Fukunaga, 2014, 2015) and its LP relaxation. The MDD, BDD, and LP are used as heuristics within our binary search algorithm (see Section 5.9.1), while the MILP is solved once using an external solver.

Table 5.2: Selected IPC domain names and abbreviations.

Name	Abbre.	Name	Abbre.	Name	Abbre.
barman-opt11	bar11	nomystery-opt11	nom11	transport-opt11	tra11
barman-opt14	bar14	openstacks-opt11	ope11	transport-opt14	tra14
childsnack-opt14	chi14	parking-opt11	par11	visitall-opt11	vis11
elevators-opt11	ele11	parking-opt14	par14	visitall-opt14	vis14
floortile-opt11	flo11	pegsol-opt11	peg11	woodworking-opt11	woo11
floortile-opt14	flo14	scanalyzer-opt11	sca11		
ged-opt14	ged14	sokoban-opt11	sok11		

We test all approaches over delete-free version of domains from the IPC2011 and IPC2014 competitions. We restrict ourselves to domains with no negative preconditions and no conditional effects.[1] Table

[1] No domains from IPC2018 satisfy these requirements.

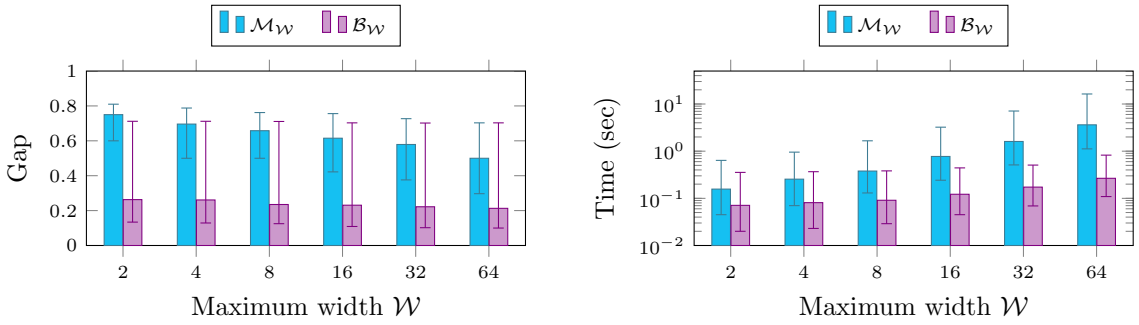
5.2 shows the selected domains and their abbreviated names. Our experiments consider a 30 minute time limit and a 2GB memory limit. We use CPLEX 12.9 (IBM, 2019) to solve the LP and MILP models. Everything is coded in C++.

To make the comparison as fair as possible, we calculate an initial upper bound using the FF heuristic (Hoffmann and Nebel, 2001) and implement a plan extraction procedure for the LP heuristic. Whenever the LP returns an integer solution, we check if it is a plan. If so, we update the global upper bound accordingly.

5.10.1 Relaxed DD Heuristics Analysis

We first analyze the heuristic quality of our relaxed DDs in the initial state $\mathtt{s}_I$. To do so, we compute the optimality gap (i.e., relative distance to the perfect heuristic) for a heuristic h at the initial state as $gap(h) = (h^+(\mathtt{s}_I) - h(\mathtt{s}_I))/h^+(\mathtt{s}_I)$. Notice that an optimality gap closer to zero means a more informative heuristic.

Figure 5.6a compares the median optimality gap for each relaxed DD heuristic using different maximum widths. The error bars correspond to the first and third quartile of the gap. These values are calculated over all the instances where the cost-optimal plan (i.e., $h^+(\mathtt{s}_I)$) was available, 314 out of 374 instances. The figure shows that as the maximum width $\mathcal{W}$ increases the gap decreases. However, there is a trade-off between informative heuristics and computational time. Figure 5.6b shows the median time to construct and compute a relaxed DD heuristic in the initial state, where the error bars correspond to the first and third quartile. Since the relaxed DD width grows exponentially, the time of its construction does too.



(a) Median optimality gap.

(b) Median heuristic time computation.

Figure 5.6: Initial heuristic comparison for different maximum widths.

From Figure 5.6b we also observe that the time to construct a relaxed MDD is much higher than the time for a relaxed BDD. This is mostly explained by the size discrepancy between the two graphical structures (see Section 5.7).

While Figure 5.6a shows that in median the relaxed BDD heuristic is more informative, this tendency depends on the DFP domain. Figure 5.7 compares the average optimality gap for our best relaxed MDD and BDD heuristics (i.e., with $\mathcal{W} = 64$) and the LP heuristic (see Appendix B.1 for a complete list of optimality gaps for all widths). The figure shows that in most domains the LP heuristic has the smallest gap and $\mathcal{M}_{64}$ the largest ones. Nonetheless, the relaxed MDD heuristic has an exceptional performance in the two transport domains (*tra11* and *tra14*). Additionally, the $\mathcal{B}_{64}$ heuristic is perfect in *nom11*, a

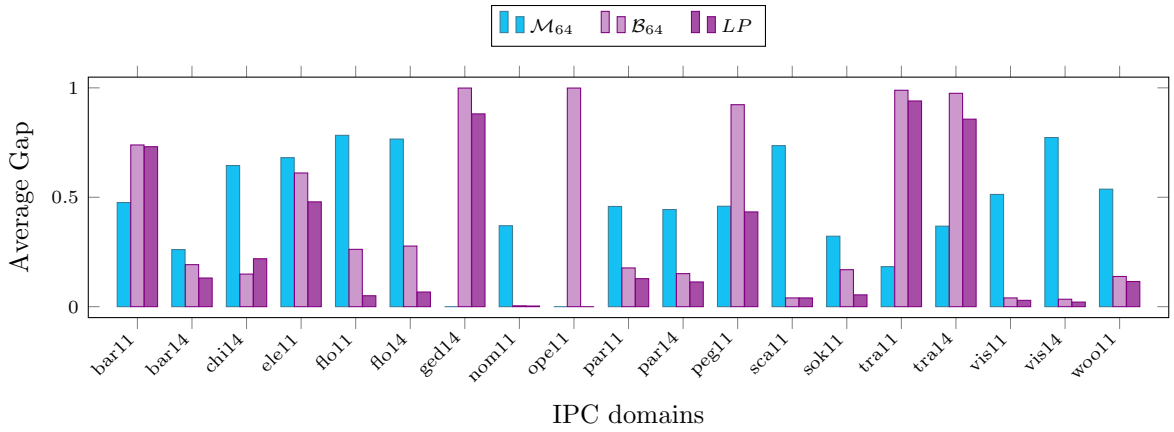


Figure 5.7: Initial heuristic comparison ($\mathcal{W} = 64$).

domain with a large number of fact landmarks and where the optimal sr-plan cost is equal to the cost of the optimal DFP plan in all instances.

Figure 5.7 also shows that the $\mathcal{M}_{64}$ heuristic is perfect for all instances in domains *ged14* and *ope11* (i.e., gap = 0). These two domains have several zero-cost operators and in all instances the optimal plan needs exactly one operator with positive cost. Since our search procedure skips branching on applicable zero-cost operators (see Section 5.9.1), our MDD can easily identify the needed positive cost operator. In contrast, $\mathcal{B}_{64}$ evaluates to zero in all instances of these domains since the optimal cost of the sequential relaxation is zero. This explains the larger gap of $\mathcal{B}_{64}$, which is always equal to 1.

5.10.2 Effectiveness of MDD and BDD Heuristics

As discussed in the previous section, the quality of our relaxed DD heuristics depends on the specific domain at hand. We believe that relaxed MDDs generate informative heuristics when the sequential aspect of the problem is predominant (e.g., in the transport domain), while relaxed BDDs have better performance when the sequential relaxation is a good approximation to the original DFP task and there are a high number of fact landmarks. These hypotheses arise from our construction procedures, their theoretical relationship with existing techniques, and the results presented in Figure 5.7.

Figure 5.8: Examples of random instances for 10×4 visit-all domain with $|G| = 4$. The left drawing illustrates the full grid setting (i.e., goals in any cell) and the right grid depicts the half grid setting (i.e., goals in the right half of the grid).

We test our hypotheses in modified instances from the visit-all domain using $\mathcal{M}$ and $\mathcal{B}$ with $\mathcal{W} = 4$. We consider a 10×4 grid with different number of randomly placed goals, $|G| \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}$. In all instances, the agent starts in the bottom left corner cell. We consider two different grid settings depicted in Figure 5.8. The left one considers one goal in the initial cell and the others placed randomly in the grid. The right grid setting also allocates one goal in the initial cell but

all the other goals are allocated randomly in the right half of the grid, i.e., the blue region in the grid. We create 10 random instances for each grid configuration (i.e., full and half) and number of goals.

Notice that the half grid setting is adversarial for the sequential relaxation. Since sr-plans are not forced to consider operators applicable to the initial state, cost-optimal sr-plans can avoid using operators in the left half of the grid. In addition, changing the number of goals allows us to analyze the effectiveness of our heuristic with different numbers of fact landmarks.

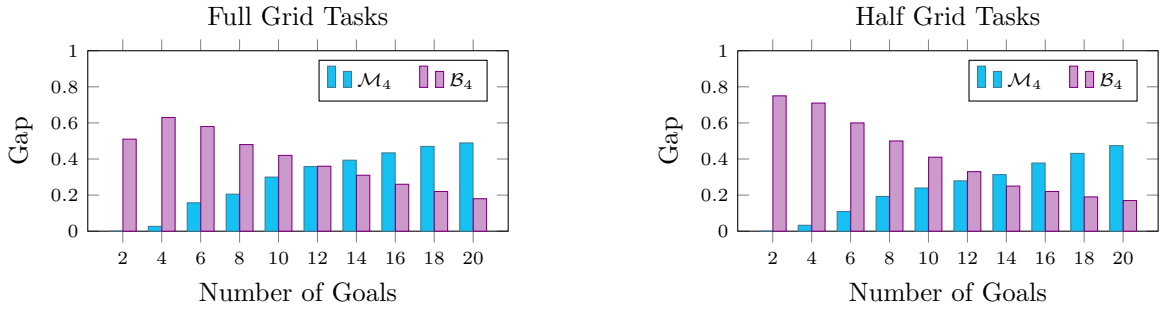


Figure 5.9: Initial heuristic comparison for different maximum widths.

Figures 5.9 present the average optimality gap at the initial state for each heuristic and configuration. The plots show that $\mathcal{M}_4$ is more informative when the number of goals is small (i.e., $|G| \leq 10$). In these cases, our critical path heuristic to the most expensive goals is a good estimate for the cost-optimal plan. However, when the number of goals increases, the cost to reach the most expensive goals is a weak estimate.

When all the goals are in the right half of the grid the sequential relaxation is a bad approximation for the DFP task since there is no need to use any operators in the left side of the room. As a consequence, the optimality gap for $\mathcal{B}_4$ is worse in the half grid setting than in the full grid. Also, notice that in this domain the number of fact landmarks is equivalent to the number of goals. Hence, fewer goals mean a less informative disjunctive landmark heuristic. This statement correlates with the behavior of $\mathcal{B}_4$, where we see an improvement in the optimality gap when the number of goals increases.

These experiments illustrate our claim for these domains: $\mathcal{M}$ is more informative when we have a small number of goals and the operator sequencing is important. In contrast, $\mathcal{B}$ is informative in the presence of a large number of fact landmarks and when the sequential relaxation is a good estimate of the DFP task.

5.10.3 Overall Performance Evaluation

We now present the performance of our approaches and compare them with state-of-the-art techniques. Table 5.3 shows the coverage (i.e., number of instances solved) using our relaxed DD heuristics, where column "#" corresponds to the number of instances in each domain. Overall, the relaxed BDD heuristics solve approximately 100 more instances than when using the relaxed MDD heuristics. This correlates with the results presented in Figures 5.6 and 5.7 which show that the relaxed BDD heuristics are less computationally expensive and more informative in most domains. The only exception is the two transport domains where the relaxed MDD heuristics are more informative and solve more instances. Table 5.3 also highlights the trade-off of using different maximum widths. In some domains, it is preferable to use a smaller relaxed DDs while in others bigger diagrams solve more instances.

Table 5.3: Coverage for different relaxed MDDs and BDDs maximum widths.

		Number of Instances Solved											
domain	#	$\mathcal{M}_2$	$\mathcal{M}_4$	$\mathcal{M}_8$	$\mathcal{M}_{16}$	$\mathcal{M}_{32}$	$\mathcal{M}_{64}$	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
bar11	20	0	1	**4**	**4**	**4**	**4**	**0**	**0**	**0**	**0**	**0**	**0**
bar14	14	0	0	3	3	**6**	**6**	**14**	**14**	**14**	**14**	**14**	**14**
chi14	20	**4**	**4**	**4**	**4**	**4**	**4**	18	18	**20**	**20**	**20**	**20**
ele11	20	**16**	14	12	11	8	7	**20**	**20**	19	18	16	15
flo11	20	**2**	**2**	**2**	**2**	1	0	4	4	4	**5**	4	4
flo14	20	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
ged14	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
nom11	20	5	6	6	6	**8**	**8**	16	18	18	18	18	**20**
ope11	20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
par11	20	0	0	0	0	**1**	**1**	**6**	4	3	3	3	3
par14	20	0	0	0	0	0	**1**	**9**	**9**	**9**	**9**	7	6
peg11	20	**19**	**19**	**19**	**19**	**19**	17	**19**	**19**	18	18	18	17
sca11	20	**1**	**1**	**1**	**1**	**1**	**1**	**7**	**7**	6	6	5	5
sok11	20	**20**	18	18	18	17	16	18	18	19	19	19	**20**
tra11	20	**2**	**2**	**2**	**2**	1	1	**1**	**1**	**1**	**1**	0	0
tra14	20	1	1	**3**	**3**	**3**	2	**1**	**1**	**1**	**1**	**1**	0
vis11	20	**9**	**9**	**9**	**9**	**9**	**9**	**16**	**16**	**16**	**16**	**16**	**16**
vis14	20	**3**	**3**	**3**	**3**	**3**	**3**	**17**	**17**	16	16	16	16
woo11	20	2	3	3	3	4	**6**	14	**18**	17	17	17	17
Total	374	124	123	**129**	128	**129**	126	221	**225**	222	222	215	214

Figure 5.10 compares the number of states evaluated when using relaxed DDs of extreme widths (i.e., $\mathcal{W} = 2$ and $\mathcal{W} = 64$) for instances where both techniques find optimal solutions. The x and y coordinates in the plots correspond to the number of states evaluated by a relaxed DD with $\mathcal{W} = 2$ and $\mathcal{W} = 64$, respectively. We can see that for both relaxed DDs, most of the points are below the diagonal, which shows that bigger widths result in fewer evaluated states.[2] However, points on the diagonal indicate that bigger widths do not always affect the number of states evaluated. This behavior holds, for example, for relaxed BDDs that have a marginal increase in heuristic quality when the width increases (see Figure 5.6a).

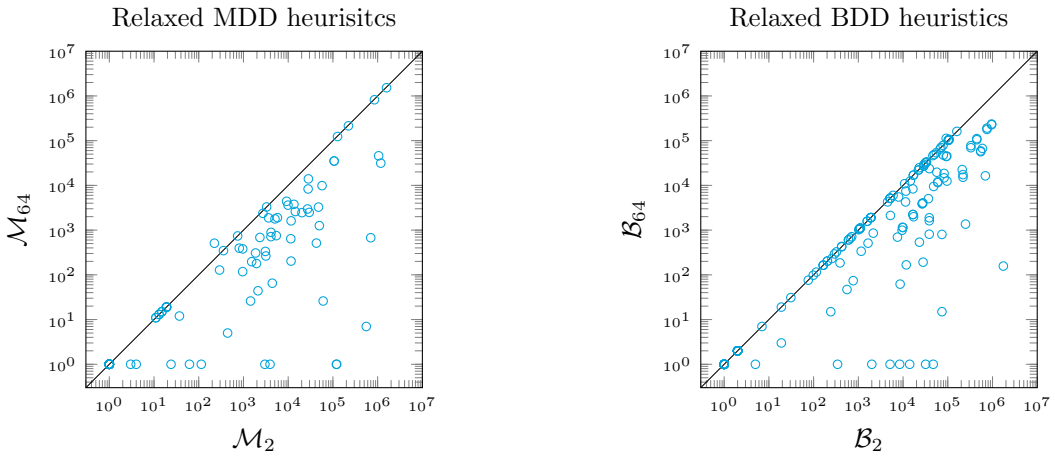


Figure 5.10: Number of states evaluated.

While using a larger width can lead to a significant reduction of states evaluated, Figure 5.11 shows

[2]The only point above the diagonal corresponds to an instance where, by chance, $\mathcal{W} = 64$ extracted the cost-optimal plan later during search.

that this reduction might not translate to faster solving time. Both plots have most points above the diagonal, illustrating that small-width relaxed DDs solve the problem faster. Nevertheless, there are some exceptional cases where larger and more informative relaxed DDs reduce the computational time by orders of magnitude.

The behavior illustrated in Figure 5.10 is mostly explained by the time to obtain our DD heuristics. As shown in Figure 5.6b, the construction time of the relaxed DDs at the root node grows exponentially with the maximum width. Thus, while large DDs provide more informative heuristics, the time to construct such DDs is high. In light of this fact, we believe that more efficient DD construction procedures could significantly reduce the time to solve the problem.

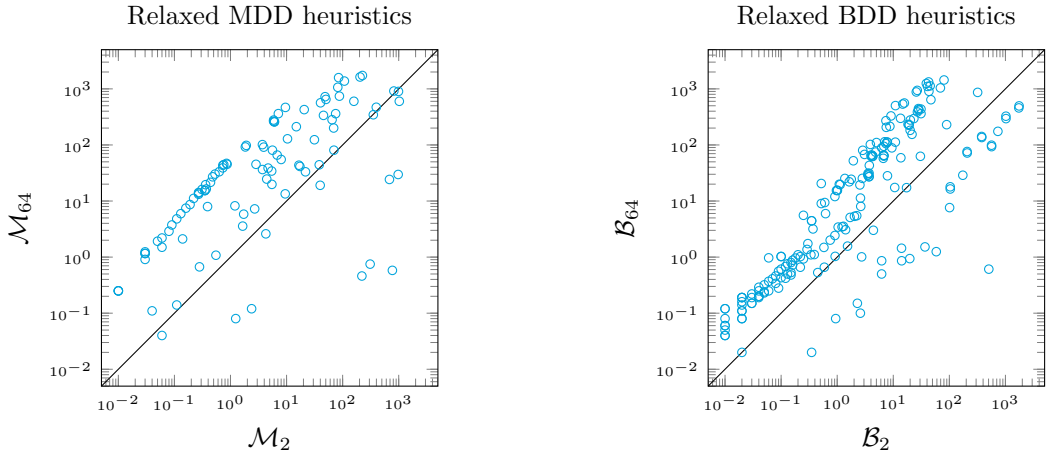


Figure 5.11: Time (sec) to solve a DFP task.

Lastly, we compare the best performing relaxed MDDs and BDDs with the LP heuristic and the MILP model. The first set of columns in Table 5.4 shows the number of instances solved by each approach. MILP has the best coverage, followed by our best performing relaxed BDD and the LP heuristic. Nonetheless there are some domains where our relaxed MDD heuristic outperforms the MILP model (i.e., the first *transport* domain and *pegsol*). Also, our BDD heuristic outperforms the MILP model in *elevators*, *pegsol* and *scanalyzer*.

We notice that the DD heuristics achieve higher coverage than MILP in domains where the LP relaxation is weak. In particular, the MILP model has a large number of big-M constraints when the sequencing aspect of the problem is predominant and, therefore, a weak relaxation (i.e., *transport* domains). We also notice that our tree-search procedure handles domains with zero-cost operators better (see Section 5.9.1), which explains the superior coverage of our approach in *ele11* and *peg11*. It is also interesting to see the large coverage difference between the MILP model and LP heuristic, which is due to the highly optimized MILP solvers and sophisticated techniques implemented in them (e.g., cut generation).

There are a number of domains where the BDD heuristic has a higher coverage than the LP heuristic even though the later has a smaller average gap (i.e., domains bar14, par11, par14, and woo11). There are two factors that played a key role in reducing the number of states evaluated and, thus, explaining these results. First, in these domains, the BDD heuristic is usually more informative than the LP heuristic when evaluating states closer to a goal state. Second, in general the BDD plan extraction procedure finds cost-optimal plans earlier during search than the LP methodology.

Table 5.4: Comparison with state-of-the-art approaches.

		Coverage					Average Time (sec)					Average # States Evaluated			
domain	#	MILP	LP	$\mathcal{B}_4$	$\mathcal{M}_8$	$\mathcal{M}_{32}$	MILP	LP	$\mathcal{B}_4$	$\mathcal{M}_8$	$\mathcal{M}_{32}$	LP	$\mathcal{B}_4$	$\mathcal{M}_8$	$\mathcal{M}_{32}$
bar11	20	**8**	0	0	4	4	-	-	-	-	-	-	-	-	-
bar14	14	**14**	9	**14**	3	6	**0.7**	454.8	7.1	257.2	2.7	23,602.7	15,765.7	143,597.7	**506.0**
chi14	20	**20**	6	18	4	4	**0.4**	89.7	1.4	313.2	339.8	6,440.5	**5,996.5**	675,891.8	671,269.3
ele11	20	18	17	**20**	12	8	366.3	114.7	**46.0**	531.4	887.3	64,572.0	64,818.9	14,515.1	**5,771.4**
flo11	20	**20**	14	4	2	1	**0.3**	3.8	12.0	607.8	1417.3	**1,681.0**	23,998.0	222,565.0	146,945.0
flo14	20	**20**	16	1	0	0	-	-	-	-	-	-	-	-	-
ged14	20	**20**	**20**	**20**	**20**	**20**	391.1	39.5	**2.1**	8.6	40.8	25,773.8	2,110.9	**9.1**	**9.1**
nom11	20	**20**	18	18	6	8	0.9	0.8	**0.1**	11.4	0.5	8.5	**1.0**	2,100.5	11.0
ope11	20	**20**	**20**	**20**	**20**	**20**	0.9	0.8	**0.1**	1.1	6.1	**1.0**	2.0	**1.0**	**1.0**
par11	20	**18**	5	4	0	1	-	-	-	-	-	-	-	-	-
par14	20	**20**	7	9	0	0	-	-	-	-	-	-	-	-	-
peg11	20	16	**20**	19	19	19	180.5	**9.7**	158.0	116.2	251.3	9,907.7	228,516.2	9,130.1	**3,411.3**
sca11	20	6	5	**7**	1	1	0.3	0.3	**0.0**	0.2	0.1	20.0	**19.0**	348.0	348.0
sok11	20	**20**	**20**	18	18	17	151.2	**7.0**	9.7	94.5	203.3	**1,104.4**	6,895.9	5,885.6	3,197.6
tra11	20	0	0	1	**2**	1	-	-	-	-	-	-	-	-	-
tra14	20	**3**	0	1	**3**	**3**	-	-	-	-	-	-	-	-	-
vis11	20	**20**	16	16	9	9	**0.3**	1.0	1.1	84.4	101.6	**408.4**	3,574.8	50,985.2	14,804.4
vis14	20	**20**	17	17	3	3	**0.3**	0.5	1.4	234.2	265.5	**158.7**	3,802.0	145,634.3	45,795.7
woo11	20	**20**	14	18	3	4	0.6	6.1	**0.2**	473.9	282.8	**372.7**	450.0	54,806.0	8,154.0
Total\Ave.	374	**303**	224	225	129	129	143.6	33.9	**28.5**	121.6	187.9	**11,828.5**	40,311.1	43,175.9	29,588.8

The second and third set of columns of Table 5.4 present the average run time and states evaluated by each approach over the instances that all of them solved. Since the MILP model is not implemented inside our search procedure, we only present the average states evaluated for the other techniques. The results illustrate that there is a wide performance variability and different heuristics should be preferred in specific domains.

In some domains (e.g., *nom11* and *ope11*) the heuristic methods find optimal plans in the initial state, which is unusual in A^* search. Since our search algorithm has a bounding procedure (see Section 5.9.1), it stops as soon as the best lower bound is equal to the incumbent. Hence, if the heuristic at the initial state is perfect and the extracted plan has optimal cost, there is no need to evaluate any other state.

5.10.4 Using BDDs as a Pre-Processing Tool

Our last set of experiments evaluates the use of relaxed BDDs as a pre-processing tool. Table 5.5 shows the average percentage of redundant operators identified by a relaxed BDD in the initial state *after* applying traditional techniques (see Section 5.9.3). We can see that the average percentage is high in some domains, especially in *ged*, *nomystery* and *sokoban* with 6% to 22%. Notice that the identified redundant operators correspond to operators that, when used, will lead to sub-optimal or non-minimal plans. Hence, our approach identifies more redundant operators in instances with an accurate initial incumbent solution and where the sequential relaxation is a good approximation to the DFP task.

Lastly, Table 5.6 shows the average number of cost-optimal operator landmarks found by our relaxed BDDs in the initial state after applying the standard pre-processing techniques (see Section 5.9.3). Our approach was able to identify a significant number of cost-optimal landmarks in some domains, in particular *ged* and *sokoban*. Both domains have a large number of zero-cost operators, which might explain why our relaxed BDDs identify more cost-optimal landmarks.

Table 5.5: Using Relaxed BDDs to find redundant operators.

	Average % of Redundant Actions					
domain	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
barman-opt11	1.2%	1.2%	1.2%	1.2%	1.2%	1.2%
barman-opt14	1.0%	1.0%	1.0%	1.0%	1.0%	1.0%
childsnack-opt14	0%	0%	0%	0%	0%	0%
elevators-opt11	0.3%	0.3%	0.4%	0.4%	0.4%	0.4%
floortile-opt11	0%	0%	0%	0%	0%	0%
floortile-opt14	0%	0%	0%	0%	0%	0%
ged-opt14	6.2%	6.2%	6.2%	6.2%	6.2%	6.2%
nomystery-opt11	21.6%	22.4%	22.9%	23.8%	24.7%	25.8%
openstacks-opt11	0%	0%	0%	0%	0%	0%
parking-opt11	0.8%	0.8%	0.8%	0.8%	0.8%	0.8%
parking-opt14	0.8%	0.8%	0.8%	0.8%	0.8%	0.8%
pegsol-opt11	0%	0%	0%	0%	0%	0%
scanalyzer-opt11	0%	0%	0%	0%	0%	0%
sokoban-opt11	8.7%	8.7%	8.7%	8.9%	8.9%	9.1%
transport-opt11	0%	0%	0%	0%	0%	0%
transport-opt14	0%	0%	0%	0%	0%	0%
visitall-opt11	5.3%	5.3%	5.3%	5.3%	5.3%	5.3%
visitall-opt14	0.4%	0.4%	0.4%	0.4%	0.4%	0.4%
woodworking-opt11	0%	0%	0%	0%	0%	0%

Table 5.6: Using Relaxed BDDs to find cost-optimal operator landmarks.

	Average Number of Cost-Optimal Landmarks					
domain	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
barman-opt11	0	0	0	0	0	0
barman-opt14	0	0	0	0	0	0
childsnack-opt14	0	0	0	0	0	0
elevators-opt11	4.7	4.7	4.7	4.7	4.7	4.7
floortile-opt11	0	0	0	0	0	0
floortile-opt14	0	0	0	0	0	0
ged-opt14	148.4	148.4	148.4	148.4	148.4	148.4
nomystery-opt11	7.5	7.5	7.5	7.5	7.5	7.5
openstacks-opt11	0	0	0	0	0	0
parking-opt11	0	0	0	0	0	0
parking-opt14	0	0	0	0	0	0
pegsol-opt11	0.1	0.1	0.1	0.1	0.1	0.1
scanalyzer-opt11	0	0	0	0	0	0
sokoban-opt11	63.8	63.8	63.8	63.8	63.8	63.8
transport-opt11	0	0	0	0	0	0
transport-opt14	0	0	0	0	0	0
visitall-opt11	2.8	2.8	2.8	2.8	2.8	2.8
visitall-opt14	3.4	3.4	3.4	3.4	3.4	3.4
woodworking-opt11	0.5	0.5	0.5	0.5	0.5	0.5

5.11 Conclusions

This work presents new admissible heuristics for delete-free planning tasks based on relaxed decision diagrams. We introduce a novel relaxed MDD encoding for a planning task and a relaxed BDD representation for its sequential relaxation. This work includes a theoretical analysis of both heuristics and relates them to existing techniques in the literature. We show that relaxed DDs can be used beyond heuristic computation. In particular, they enable the extraction of high-quality delete-free plans and relaxed BDDs can identify cost-optimal landmarks and redundant operators.

Our experimental results show that relaxed MDDs are suited for domains with a high number of sequential decisions, while relaxed BDDs perform better in domains with a large number of fact landmarks. Overall, relaxed BDDs have competitive performance compared to an LP-based heuristic, but are still far from state-of-the-art MILP techniques. Nonetheless, our two relaxed DD heuristics achieve better coverage than a MILP model in four IPC domains.

While this work focuses on delete-free planning tasks, our proposed DD heuristics are novel to the planning community and can be used in a wider variety of planning problems. In particular, the DD heuristics are admissible for classical AI planning tasks and could be implemented in any planner that uses the STRIP formalism. However, it is not clear to us how to efficiently implement these heuristics, since it might require the planner to build a new DD in each node of the search space.

Another future direction is to combine DDs with other heuristics to solve, e.g., cost-optimal classical AI planning tasks. In particular, we can encode the DD graphs as a LP network flow model and use them inside the operator counting framework (Pommerening et al., 2014). Another alternative is to use the LP model of our DDs and combine it with other LP-based heuristics using the recently introduced Lagrangian decomposition framework in classical AI planning (Pommerening et al., 2019).

Lastly, our DD heuristics could be used to solve numerical planning tasks by extending our node encoding to consider numerical variables. In particular, it would be interesting to see how our MDD encoding could be extended to, e.g., represent the interval relaxation of numerical planning tasks (Piacentini et al., 2018).

Chapter 6

Cut Generation and Lifting for Binary Optimization Problems

Cutting plane methodologies have played a key role in the theoretical and computational development of mathematical programming (Bixby et al., 2004; Nemhauser and Wolsey, 1988). These procedures iteratively refine a relaxation of the problem by adding valid inequalities (i.e., cuts) that separate fractional points from the feasible set. Since the number of cuts added by a cutting plane routine can be exponentially large, this technique is usually used as a sub-routine in a branch-and-bound tree search procedure (Cornuéjols, 2008).

Extensive literature has focused on cuts that exploit special problem substructure, leading to an array of techniques that are now integral in state-of-the-art solvers (Lodi, 2010). For general problems, cuts are obtained either by leveraging disjunctive reformulations (Balas et al., 1993, 1996) or by *lifting*, i.e., relaxing an initial inequality so that it is valid for a higher-dimensional polyhedron (Gomory, 1969; Wolsey, 1976). Lifting procedures are commonly used to strengthen a valid inequality by modifying its coefficients to obtain a constraint that defines a facet of the convex hull of the solution set. These lifting techniques were first introduced for the knapsack problem (Balas, 1975; Padberg, 1975) and are currently employed in more general problem structures (Louveaux and Wolsey, 2003).

In this work, we study both a cut generation procedure and a lifting approach for general binary optimization problems of the form

$$\max_{\boldsymbol{x} \in \mathcal{X} \subseteq \{0,1\}^n} \boldsymbol{c}^\top \boldsymbol{x}, \tag{BP}$$

where the feasible set $\mathcal{X}$ is arbitrary, e.g., possibly represented by a conjunction of linear and/or non-linear constraints. Our methodologies consist of exploiting network structure via a binary decision diagram (BDD) embedding of $\mathcal{X}$. Several BDD encodings have already been investigated for linear and non-linear problems (Behle, 2007; Bergman et al., 2019; Lozano and Smith, 2018) and are used to exploit submodularity (Bergman and Cire, 2018) or more general combinatorial structure (Bergman et al., 2016b).

We propose a sequential lifting procedure that can be applied to any initial valid inequality (e.g., given by another cutting-plane technique). The lifting algorithm uses 0-1 disjunctions derived from a BDD representation of $\mathcal{X}$ to lift inequalities while maintaining their validity. We show that each step of our sequential lifting, when applicable, increases the dimension of the face by at least one, and we

establish conditions for which the inequality becomes facet-defining. We also draw connections between our procedure and existing lifting techniques from disjunctive programming (Balas, 2018), showing that our approach generalizes well-known lifting procedures for 0-1 inequalities (Balas, 1975; Hammer et al., 1975; Perregaard and Balas, 2001).

For our cut generation approach, we propose a new linear formulation of the BDD polytope based on capacitated flows, which leads to an alternative cut generation linear program (CGLP) for separating infeasible points from $\mathcal{X}$. We show that the set of cuts derived from this model defines the convex hull of the solutions encoded by the BDD, i.e., $\mathcal{X}$. Moreover, in contrast to recent cutting-plane algorithms based on BDDs (Davarnia and van Hoeve, 2020; Tjandraatmadja and van Hoeve, 2019), our CGLP does not require any additional information about $\mathcal{X}$, such as interior points or normalization constraints. Finally, for practical purposes, we build on this model to present a weaker but computationally faster alternative that solves a combinatorial max-flow/min-cut problem over the BDD to generate valid inequalities.

For optimization problems where a BDD for $\mathcal{X}$ may be exponentially large in n, our lifting and cutting procedures remain valid when considering instead a limited-size relaxed BDD for BP, i.e., where the BDD encodes a superset of $\mathcal{X}$. This approach is similar in spirit, e.g., to when a linear relaxation is used to lift cover inequalities of a single knapsack constraint (Balas, 1975). Nonetheless, here we exploit the discrete relaxation provided by the BDD as opposed to a continuous relaxation, which captures some of the combinatorial structure of the problem.

As a case study, we apply our combinatorial cut-and-lift procedure to a class of binary second-order cone programming problems (SOCPs). Second-order cone (SOC) inequalities arise in many applications, including network (Atamtürk and Bhardwaj, 2018), assortment (Şen et al., 2018), and over-commitment (Cohen et al., 2019) problems. In fact, these SOCP applications correspond to reformulations of chance-constrained stochastic problems where the stochastic variables have a normal distribution (Van de Panne and Popp, 1963; Lobo et al., 1998).

This chapter focuses on SOCPs of the following form

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \left\{ \boldsymbol{c}^\top \boldsymbol{x} : \quad \boldsymbol{a}_j^\top \boldsymbol{x} + ||D_j^\top \boldsymbol{x} - \boldsymbol{e}_j||_2 \leq b_j, \quad \forall j \in \{1, ..., m\} \right\}, \tag{SP}$$

where $||\cdot||_2$ is the Euclidean norm and, for each j, $\boldsymbol{a}_j, \boldsymbol{e}_j$, and D_j are real vectors and matrices of appropriate dimension, respectively. This type of problem is currently supported by commercial solver such as CPLEX (IBM, 2019) and Gurobi (Gurobi Optimization, 2020). However, solver performance on problems with SOC inequalities is still not comparable to the integer linear programming (ILP) case, despite advances in linearization methods (Vielma et al., 2008, 2017) and cutting-planes for SOCPs (Atamtürk and Narayanan, 2009, 2010; Atamtürk et al., 2013; Bhardwaj, 2015; Lodi et al., 2019).

We investigate problems with multiple SOC inequalities, each reformulated with an appropriate BDD encoding. We experiment on the SOC knapsack benchmark (Atamtürk and Narayanan, 2009; Joung and Park, 2017) and over 270 randomly generated instances with general-form second-order cone inequalities, incorporating our combinatorial cut-and-lift approach into CPLEX. We show that our combinatorial cut-and-lift procedure achieves a 52.2% average root gap reduction and solves 17 more instances on the knapsack benchmark when compared to existing cut-and-lift methodologies (Atamtürk and Narayanan, 2009). Similarly, our procedure outperforms CPLEX on the random dataset by achieving a 35.3% average root gap reduction, solving 31 more instances (168 vs. 137), and reducing the mean run-time threefold.

Main contributions. Our first contribution is a general combinatorial lifting procedure which can be applied to any binary problem BP where the feasibility set is encoded by one or multiple (relaxed) BDDs. We show theoretical properties of the lifted inequality, including sufficient conditions to obtain facet-defining inequalities. The procedure relates to several lifting algorithms based on 0-1 disjunctions and it is the first one to leverage the combinatorial structure of the problem via BDDs.

Our second key contribution is a novel cutting-plane algorithm defined over a BDD. This procedure extends the literature on BDD-based cuts by proposing a new cut generation approach that formulates the separation problem as a joint-capacity max-flow problem over the network. We show that our cuts define the convex hull of $\mathcal{X}$ and present a theoretical comparison with existing BDD-based cutting-plane approaches. In addition, we introduce a tractable but weaker alternative to our BDD-based cuts and prove that these cuts are stronger when using reduced BDDs.

Finally, we present an extensive numerical analysis that evaluates the effectiveness of our combinatorial cut-and-lifting procedure for SOCP problems. We tested our approach over the well-known SOC knapsack constraints and general-form SOC inequalities coming from chance-constrained stochastic problems. Overall, our procedure outperforms CPLEX in both scenarios and achieves better performance than existing cut-and-lifting techniques for SOC knapsack constraints.

This work is currently under review in *Mathematical Programming, Special Issue on Global Solution of Integer, Stochastic, and Nonconvex Optimization Problems* (Castro et al., 2020b). This work won the Canadian Operation Research Society (CORS) student paper competition and was the runner-up for the INFORMS Computing Society (ICS) student paper award.

Outline. The remainder of the chapter is as follows. Section 6.1 describes related works in the BDD and lifting literature and Section 6.2 introduces the notation used through this chapter. Section 6.3 describes our combinatorial lifting procedure while Section 6.4 details our BDD-based cutting-plane algorithm. Section 6.5 introduces the binary SOCP problem and describes the BDD encoding for SOC inequalities. Lastly, Sections 6.6 and 6.7 present the empirical evaluation and final remarks, respectively.

6.1 Related Work

Recent research has shown the versatility of BDDs as a modeling tool for linear and non-linear constraints (Andersen et al., 2007; Bergman and Lozano, 2020; Bergman and Cire, 2018) and several other combinatorial structures (see Chapter 3 for further references). This has motivated the integration of BDDs with IP technologies, in particular in the context of cut generation (Behle, 2007; Tjandraatmadja, 2018). Becker et al. (2005) presented the first BDD cut generation procedure based on an iterative subgradient algorithm that relies on a longest-path problem over the BDD. Behle (2007) formalized this procedure and proposed a branch-and-cut algorithm that employs BDDs to generate exclusion and implication cuts. The author also introduced the network flow model employed by most BDD cutting-plane procedures (Tjandraatmadja and van Hoeve, 2019; Davarnia and van Hoeve, 2020).

Tjandraatmadja and van Hoeve (2019) recently demonstrated how to generate target cuts from polar set, using relaxed BDDs to obtain more computationally tractable procedures. Davarnia and van Hoeve (2020) proposed an iterative method to generate outer-approximations for non-linear inequalities. Both works introduce BDD-based cutting models, which we further discuss and compare to our approach in Section 6.4.4. Lastly, Lozano and Smith (2018) designed a class of BDD cuts for two-stage stochastic

programming problems using BDDs to encode second-stage decisions. The authors propose a CGLP where arc capacities are given by first-stage decisions, which resembles our approach (see Section 6.4). These BDD cuts were also applied to a two-stage operation room scheduling problem (Guo et al., 2019).

While the literature on BDD cutting-plane procedures has recently grown, to the best of our knowledge, this is the first work that leverages BDDs for lifting any type of inequality. Behle (2007) proposes lifting cover inequalities using classic techniques that compute new coefficients one at a time (Wolsey and Nemhauser, 1999) and where each sub-problem is solved using a BDD. On a related note, Becker et al. (2005) also present a mechanism that uses 0-1 disjunctions over a BDD to obtain new inequalities with potentially higher dimension. Their technique differs from ours on the procedure to obtain the new inequality and their theoretical guarantees: Their new inequality is valid but its face dimension might not increase and the inequality might not separate fractional points that the original inequality does.

Our combinatorial lifting has a strong relationship to lifting algorithms based on 0-1 disjunctions (Balas, 2018), including procedures for knapsack inequalities (Balas, 1975; Padberg, 1975, 1973) and submodular functions (Hammer et al., 1975; Atamtürk and Narayanan, 2009). In particular, our methodology is closely related to the n-step lifting procedure by Perregaard and Balas (2001), which generalizes previous works. A brief description of this procedure can be found below and its relationship to our combinatorial lifting algorithm is further explored in Section 6.3.4.

We also note that our lifting procedure can be categorized as a tilting approach. As shown by Espinoza et al. (2010), lifting and tilting are highly related techniques that can be seen as special cases of a more general optimization problem. In fact, our lifting procedure has the key properties of titling techniques (Espinoza, 2006), i.e., it returns a valid inequality with an induced face that strictly contains the induced face of the starting inequality (see Theorem 6.1).

Iterative Lifting Procedure based on Disjunctive Programming. We now review Perregaard and Balas (2001) n-step lifting procedure that is closely related to our methodology. Given a mixed-integer linear programming (MILP) problem of the form $\max_{\boldsymbol{x}\in\mathbb{R}^n}\{\boldsymbol{c}^\top\boldsymbol{x}\colon A\boldsymbol{x}\le\boldsymbol{b}, x_i\in\mathbb{Z}\ \forall i\in I'\subseteq I\}$ with $I=\{1,\dots,n\}$, the authors propose the relaxation

$$\max_{\boldsymbol{x}\in\mathbb{R}^n}\left\{\boldsymbol{c}^\top\boldsymbol{x}:\ A\boldsymbol{x}\le\boldsymbol{b},\ \bigvee_{k\in K} D^k\boldsymbol{x}\le\boldsymbol{d}^k, x_i\in\mathbb{Z}\quad\forall i\in I''\subset I'\right\}, \tag{DP}$$

where fewer variables are constrained to be integral. Set K defines the disjunctive constraints and is typically derived by considering the 0-1 integrality constraints (e.g., $x_i\le 0\vee x_i\ge 1$).

Let P_{DP} be the set of solutions of DP. The n-step procedure considers two inputs: (a) an inequality $\boldsymbol{\pi}^\top\boldsymbol{x}\le\pi_0$ that supports $\text{conv}(P_{DP})$; and (b) an arbitrary target inequality $\widetilde{\boldsymbol{\pi}}^\top\boldsymbol{x}\le\widetilde{\pi}_0$ that is tight for all integer points in $F(\boldsymbol{\pi})$. The procedure uses a parameter γ to lift the supporting inequality towards the target inequality, generating a new lifted inequality $(\boldsymbol{\pi}+\gamma\widetilde{\boldsymbol{\pi}})^\top\boldsymbol{x}\le\pi_0+\gamma\widetilde{\pi}_0$ that is valid for $\text{conv}(P_{DP})$. In particular, if $\widetilde{\boldsymbol{\pi}}^\top\boldsymbol{x}\le\widetilde{\pi}_0$ is *not* valid for P_{DP}, it can be shown that there is a finite maximal γ^* given by the disjunctive program

$$\gamma^*=\min_{\boldsymbol{x},x_0}\left\{\pi_0x_0-\boldsymbol{\pi}^\top\boldsymbol{x}:\ A\boldsymbol{x}-\boldsymbol{b}x_0\le 0,\ \bigvee_{k\in K} D^k\boldsymbol{x}-\boldsymbol{d}^kx_0\le 0,\right.$$
$$\left.\widetilde{\pi}_0x_0-\widetilde{\boldsymbol{\pi}}^\top\boldsymbol{x}=-1,\ \ x_0\ge 0,\ \ x_i\in\mathbb{Z}\ \ \forall i\in I''\subset I'\right\}.$$

Under the same assumptions, the lifted inequality becomes a facet of $\text{conv}(P_{DP})$ if the procedure is repeated n times, using the lifted inequality and an appropriate target inequality.

Similarly, our approach is a sequential procedure that relies on disjunctive inequalities for lifting. It differs from existing methods in that we exploit the combinatorial structure encoded by a BDD as opposed to a disjunctive program relaxation. Such a BDD may encode, e.g., complex non-linear constraints that are not necessarily convex (Bergman and Cire, 2018). Furthermore, we also exploit the network to derive a tractable and efficient way to compute several disjunctions simultaneously, while previous algorithms are typically restricted to a small number of disjunctions (Perregaard and Balas, 2001). To highlight the connection between methods, we show in Section 6.3.4 that our lifting technique becomes a special case of Perregaard and Balas (2001) under a restricted setting.

6.2 Notation

This section introduces the notation used throughout this chapter. For convenience, we assume $n \geq 1$ and let $I = \{1, ..., n\}$ represent the component indices of any point $\boldsymbol{x}$ in an n-dimensional set.

Facets and Convex Hulls. We denote by $\dim(P)$ the dimension of a polytope $P \subseteq [0,1]^n$. An inequality $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ with $\boldsymbol{\pi} \in \mathbb{R}^n$ and $\pi_0 \in \mathbb{R}$ is valid for P if $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ holds for all $\boldsymbol{x} \in P$. The inequality defines a face of P if $F(\boldsymbol{\pi}) = \{\boldsymbol{x} \in P : \boldsymbol{\pi}^\top \boldsymbol{x} = \pi_0\}$ is not empty, i.e., the inequality *supports* P. A face $F(\boldsymbol{\pi})$ is a *facet* if $\dim(F(\boldsymbol{\pi})) = \dim(P) - 1$; in such a case, $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ is *facet-defining*. Finally, we denote the convex hull of P by $\text{conv}(P)$.

Binary Decision Diagrams. We use the BDD notation introduced in Chapter 2. For a given feasible set $\mathcal{X} \subseteq \{0,1\}^n$, a BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ for $\mathcal{X}$ is a layered directed acyclic graph where the node set $\mathcal{N}$ is partitioned into $n+1$ layers $\mathcal{N} = (\mathcal{N}_1, ..., \mathcal{N}_{n+1})$. We associate a value $v_a \in \{0,1\}$ to each arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$ that represents the value assign to the i-th variable in $\boldsymbol{x}$, i.e., paths traversing arc a are such that $x_i = v_a$.

Recall that $\mathcal{B}$ is exact for set $\mathcal{X} \subseteq \{0,1\}^n$ when $\mathcal{X} = \text{Sol}(\mathcal{B})$, i.e., there is a one-to-one relationship between the points in $\mathcal{X}$ and the $\mathbf{r}-\mathbf{t}$ paths in $\mathcal{B}$. Alternatively, $\mathcal{B}$ is relaxed when $\mathcal{X} \subseteq \text{Sol}(\mathcal{B})$, i.e., every point in $\mathcal{X}$ maps to a path in $\mathcal{B}$ but the converse is not necessarily true. We discuss the construction and relaxation techniques used for our SOCP case study in Section 6.5.

Example 6.1 Consider the knapsack feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ introduced in Example 2.1. Figure 6.1 illustrates two exact BDDs for $\mathcal{X}$: $\mathcal{B}_1$ on the left-hand side and $\mathcal{B}_2$ on the right-hand side. Dashed and solid arcs have a value of 0 and 1, respectively. Each point $\boldsymbol{x} \in \mathcal{X}$ is represented by a path in $\mathcal{B}_1$ and $\mathcal{B}_2$. For example, $\boldsymbol{x} = (1,0,0,1) \in \mathcal{X}$ is encoded by the path $((\mathbf{r}, u_2), (u_2, u_4), (u_4, u_5), (u_5, \mathbf{t}))$ in $\mathcal{B}_1$, and by the path $((\mathbf{r}, u'_2), (u'_2, u'_5), (u'_5, u'_6), (u'_6, \mathbf{t}))$ in $\mathcal{B}_2$. □

6.3 Combinatorial Lifting

We now present our combinatorial lifting procedure and develop its structural properties. Throughout this section, we assume that, for a given $\mathcal{X} \subseteq \{0,1\}^n$:

(a) $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ is a valid inequality that supports $\text{conv}(\mathcal{X})$;

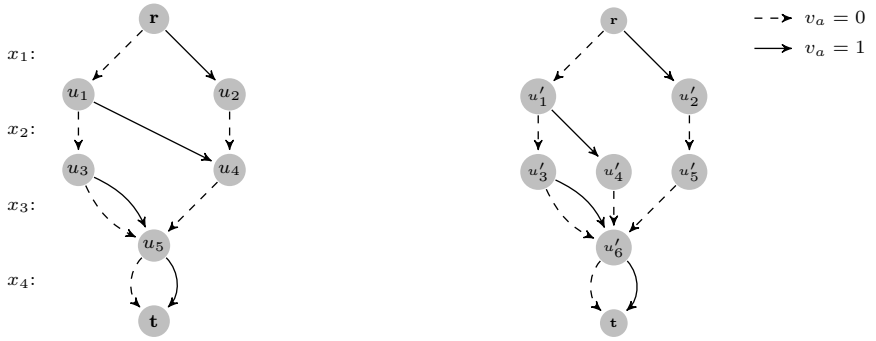


Figure 6.1: Two BDDs $\mathcal{B}_1$ (left-hand side) and $\mathcal{B}_2$ (right-hand side) with $\text{Sol}(\mathcal{B}_1) = \text{Sol}(\mathcal{B}_2)$. $\mathcal{B}_1$ is a reduced BDD.

(b) $\mathcal{B}$ is an exact BDD for $\mathcal{X}$, i.e., $\text{Sol}(\mathcal{B}) = \mathcal{X}$; and

(c) For any $i \in I$, there exists two points $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ such that their i-th components have different values, i.e., $x_i = 0$ and $x'_i = 1$.

Assumption (a) is a common lifting condition that is satisfied by setting $\pi_0 = \max_{\boldsymbol{x} \in \mathcal{X}} \left\{ \boldsymbol{\pi}^\top \boldsymbol{x} \right\}$. This, in turn, can be enforced in linear time in the size of $\mathcal{B}$ (see Section 6.3.2). Assumption (b) is needed for our theoretical results but it can be relaxed in practice (see Section 6.6). For (c), we can soundly remove any i-th component not satisfying the assumption, adjusting n accordingly.

Our goal is to lift $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ and better represent $\text{conv}(\mathcal{X})$ by exploiting the network structure of $\mathcal{B}$. The resulting cuts are valid for any subset $\mathcal{X}' \subseteq \mathcal{X}$; e.g., when $\mathcal{B}$ (and hence $\mathcal{X}$) is a relaxation of some feasible set.

We begin in Section 6.3.1 by introducing our lifting mechanism based on variable disjunctions. We then present in Section 6.3.2 a methodology that computes such a lifting in polynomial time in the size of $\mathcal{B}$ (i.e., in the number of nodes and arcs). Next, in Section 6.3.3 we incorporate the technique in a sequential procedure and investigate the dimension of the resulting face. Finally, we depict the relationship with previous disjunctive methodologies in Section 6.3.4.

6.3.1 Disjunctive Slack Lifting

The core element of our lifting procedure is what we denote as a *disjunctive slack vector* (or *d-slack* in short). The i-th component of the d-slack indicates the change in the maximum values of the left-hand side of $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ when varying x_i. This is formalized in Definition 6.1.

Definition 6.1. *The disjunctive slack vector* $\boldsymbol{\lambda}(\boldsymbol{\pi})$ *with respect to* $\boldsymbol{\pi}$ *is*

$$\lambda_i(\boldsymbol{\pi}) = \lambda_i^0(\boldsymbol{\pi}) - \lambda_i^1(\boldsymbol{\pi}), \quad \forall i \in I,$$

with $\lambda_i^0(\boldsymbol{\pi}) = \max_{\boldsymbol{x} \in \mathcal{X}} \{\boldsymbol{\pi}^\top \boldsymbol{x} : x_i = 0\}$ *and* $\lambda_i^1(\boldsymbol{\pi}) = \max_{\boldsymbol{x} \in \mathcal{X}} \{\boldsymbol{\pi}^\top \boldsymbol{x} : x_i = 1\}$.

For notational convenience, we let $I^-(\boldsymbol{\pi}) = \{i \in I : \lambda_i(\boldsymbol{\pi}) < 0\}$, $I^0(\boldsymbol{\pi}) = \{i \in I : \lambda_i(\boldsymbol{\pi}) = 0\}$, and $I^+(\boldsymbol{\pi}) = \{i \in I : \lambda_i(\boldsymbol{\pi}) > 0\}$ be a partition of I with respect to negative, zero, and positive d-slacks, respectively. Lemma 6.1 presents key properties of d-slacks used for our main results.

Lemma 6.1. *For any* $\boldsymbol{\lambda}(\boldsymbol{\pi})$ *and index* $i \in I$,

(1) $i \in I^-(\boldsymbol{\pi})$ *if and only if* $x_i = 1$ *for all* $\boldsymbol{x} \in F(\boldsymbol{\pi})$.

(2) $i \in I^+(\boldsymbol{\pi})$ *if and only if* $x_i = 0$ *for all* $\boldsymbol{x} \in F(\boldsymbol{\pi})$.

(3) $i \in I^0(\boldsymbol{\pi})$ *if and only if there exists* $\boldsymbol{x}, \boldsymbol{x}' \in F(\boldsymbol{\pi})$ *with* $x_i = 0$ *and* $x'_i = 1$.

Proof. For the necessary conditions, consider first $x_i = 1$ for all $\boldsymbol{x} \in F(\boldsymbol{\pi})$. Since the solutions when optimizing over $\boldsymbol{\pi}^\top \boldsymbol{x}$ must belong to the face $F(\boldsymbol{\pi})$, we must necessarily have $\lambda_i^1(\boldsymbol{\pi}) > \lambda_i^0(\boldsymbol{\pi})$ and the d-slack $\lambda_i(\boldsymbol{\pi})$ is negative. Analogous reasoning holds for the other two cases.

For the sufficient conditions, consider first $i \in I^-(\boldsymbol{\pi})$. Then $\lambda_i^1(\boldsymbol{\pi}) = \pi_0$ and $\lambda_i^0(\boldsymbol{\pi}) < \pi_0$, i.e., all $\boldsymbol{x} \in F(\boldsymbol{\pi})$ are such that $x_i = 1$. The same argument can be applied to the case $i \in I^+(\boldsymbol{\pi})$. Lastly, if $i \in I^0(\boldsymbol{\pi})$, $\lambda_i^0(\boldsymbol{\pi}) = \lambda_i^1(\boldsymbol{\pi}) = \pi_0$. Thus, there exists $\boldsymbol{x} \in F(\boldsymbol{\pi})$ that maximizes $\lambda_i^1(\boldsymbol{\pi})$ (i.e., $x_i = 1$) and $\boldsymbol{x}' \in F(\boldsymbol{\pi})$ that maximizes $\lambda_i^0(\boldsymbol{\pi})$ (i.e., $x'_i = 0$). ■

We now show in Theorem 6.1 how to apply the d-slacks to lift $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$. The resulting inequality is valid for $\mathcal{X}$ (and thereby $\text{conv}(\mathcal{X})$), the dimension of the face necessarily increases, and points separated by the original inequality are still separated after lifting. This last characteristic is important, e.g., if the input inequality $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ was derived to separate a fractional point. Note that we require a d-slack with a non-zero component to lift the inequality, as we later illustrate in Example 6.2.

Theorem 6.1. *Suppose* $\lambda_i(\boldsymbol{\pi}) \neq 0$ *for some* $i \in I$. *Let* $\langle \boldsymbol{\pi}', \pi'_0 \rangle$ *be such that*

$$\pi'_j = \begin{cases} \pi_j & \text{if } j \neq i, \\ \pi_j + \lambda_j(\boldsymbol{\pi}) & \text{otherwise,} \end{cases} \qquad \forall j \in I,$$

and

$$\pi'_0 = \begin{cases} \pi_0 & \text{if } i \in I^+(\boldsymbol{\pi}), \\ \pi_0 + \lambda_i(\boldsymbol{\pi}) & \text{otherwise.} \end{cases}$$

The following properties hold:

(1) $\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi'_0$ *is valid for* $\mathcal{X}$.

(2) $F(\boldsymbol{\pi}) \subset F(\boldsymbol{\pi}')$ *and* $\dim(F(\boldsymbol{\pi}')) \geq \dim(F(\boldsymbol{\pi})) + 1$.

(3) For any $\bar{\boldsymbol{x}} \in [0,1]^n$ *with* $\boldsymbol{\pi}^\top \bar{\boldsymbol{x}} > \pi_0$, *we have that* $\boldsymbol{\pi}'^\top \bar{\boldsymbol{x}} > \pi'_0$.

Proof. Let $\boldsymbol{x} \in \mathcal{X}$. We begin by showing (1) and (2). Assume first that $i \in I^+(\boldsymbol{\pi})$. By construction, $\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi'_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} + \lambda_i(\boldsymbol{\pi}) x_i \leq \pi_0$. If $x_i = 0$, the lifted inequality is equivalent to the original and therefore valid. Otherwise, if $x_i = 1$, $i \in I^+(\boldsymbol{\pi})$ implies that $\lambda_i(\boldsymbol{\pi}) = \pi_0 - \lambda_i^1(\boldsymbol{\pi})$. Thus,

$$\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi'_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} + \pi_0 - \lambda_i^1(\boldsymbol{\pi}) \leq \pi_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} \leq \lambda_i^1(\boldsymbol{\pi}).$$

The last inequality above holds because we are restricting to the case $x_i = 1$ and, by definition, $\lambda_i^1(\boldsymbol{\pi}) = \max_{\boldsymbol{x}' \in \mathcal{X}} \{\boldsymbol{\pi}^\top \boldsymbol{x}' : x'_i = 1\}$. Since $x'_i = 0$ for all $\boldsymbol{x}' \in F(\boldsymbol{\pi})$ (Lemma 6.1), the lifted inequality is tight for all $\boldsymbol{x}' \in F(\boldsymbol{\pi})$, i.e., $F(\boldsymbol{\pi}) \subset F(\boldsymbol{\pi}')$. Notice also that this inequality is also tight for $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}' \in \mathcal{X}} \{\boldsymbol{\pi}^\top \boldsymbol{x}' : x'_i = 1\}$, i.e., $\boldsymbol{x}^* \in F(\boldsymbol{\pi}')$. Then, $\boldsymbol{x}^*$ is affinely independent to all points of $F(\boldsymbol{\pi})$ and therefore $\dim(F(\boldsymbol{\pi}')) \geq \dim(F(\boldsymbol{\pi})) + 1$.

Assume now that $i \in I^-(\boldsymbol{\pi})$. Once again by construction,

$$\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi'_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} + \lambda_i(\boldsymbol{\pi}) x_i \leq \pi_0 + \lambda_i(\boldsymbol{\pi}).$$

If $x_i = 1$, the lifted inequality is equivalent to the original and therefore valid. Otherwise, if $x_i = 0$, $i \in I^-(\boldsymbol{\pi})$ implies that $\lambda_i(\boldsymbol{\pi}) = \lambda_i^0(\boldsymbol{\pi}) - \pi_0$. Thus,

$$\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi'_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0 + \lambda_i^0(\boldsymbol{\pi}) - \pi_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} \leq \lambda_i^0(\boldsymbol{\pi}).$$

The last inequality above holds because $x_i = 0$ and, by definition, $\lambda_i^0(\boldsymbol{\pi}) = \max_{\boldsymbol{x}' \in \mathcal{X}} \{\boldsymbol{\pi}^\top \boldsymbol{x}' : x'_i = 0\}$. As before, notice that this inequality is tight for $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}' \in \mathcal{X}} \{\boldsymbol{\pi}^\top \boldsymbol{x}' : x'_i = 0\}$, i.e., $\boldsymbol{x}^* \in F(\boldsymbol{\pi}')$. Since $x'_i = 1$ for all $\boldsymbol{x}' \in F(\boldsymbol{\pi})$ (Lemma 6.1), the inequality is tight for all $\boldsymbol{x}' \in F(\boldsymbol{\pi})$, $\boldsymbol{x}^*$ is affinely independent to all points of $F(\boldsymbol{\pi})$, and therefore $\dim(F(\boldsymbol{\pi}')) \geq \dim(F(\boldsymbol{\pi})) + 1$.

Lastly, we demonstrate (3). We consider $i \in I^+(\boldsymbol{\pi})$; the other case is analogous. Given a fractional point $\bar{\boldsymbol{x}} \in [0,1]^n$ as defined above, we have $\boldsymbol{\pi}'^\top \bar{\boldsymbol{x}} = \boldsymbol{\pi}^\top \bar{\boldsymbol{x}} + \lambda_i(\boldsymbol{\pi}) \bar{x}_i > \pi_0 + \lambda_i(\boldsymbol{\pi}) \bar{x}_i \geq \pi_0 = \pi'_0$. ■

Example 6.2 Let $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and consider an inequality $x_1 + x_2 \leq 1$ supporting $\text{conv}(\mathcal{X})$. The d-slack is $\boldsymbol{\lambda}(\boldsymbol{\pi}) = (0,0,1,0)^\top$ and the lifted inequality with respect to $\lambda_3(\boldsymbol{\pi}) = 1$ is $\boldsymbol{\pi}'^\top \boldsymbol{x} = x_1 + x_2 + x_3 \leq 1$. Note that $\boldsymbol{\pi}'^\top \boldsymbol{x} \leq 1$ is facet-defining for $\text{conv}(\mathcal{X})$ and $\boldsymbol{\lambda}(\boldsymbol{\pi}') = \mathbf{0}$. □

6.3.2 Extracting Disjunctive Slacks from BDDs

Identifying d-slacks $\boldsymbol{\lambda}(\boldsymbol{\pi})$ is a non-trivial task since we are required to solve $2n$ binary optimization problems, i.e., one for each component $i \in I$ and values 0 and 1. We now develop a procedure that generates the d-slacks by exploiting the network representation of a BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ for $\mathcal{X}$. We also show that the procedure's complexity is linear in the number of arcs $|\mathcal{A}|$. Thus, computing $\boldsymbol{\lambda}(\boldsymbol{\pi})$ becomes tractable when either $\mathcal{B}$ is small or a relaxed BDD is considered (see Section 6.6).

First, for each arc $a \in \mathcal{A}$ with value $v_a \in \{0,1\}$ and source $s(a) \in \mathcal{N}_i$ for some $i \in I$, we associate a length of $\ell_a = \pi_i \cdot v_a$. Note that the longest $\mathbf{r} - \mathbf{t}$ path of $\mathcal{B}$ with respect to such lengths maximizes $\boldsymbol{\pi}^\top \boldsymbol{x}$ over $\mathcal{X}$. Similarly, given the $\mathbf{r} - \mathbf{t}$ paths $\mathcal{P}$ of $\mathcal{B}$, let

$$L_a = \max \left\{ \sum_{k=1}^{n} \pi_k \cdot v_{a_k} : p = (a_1, \ldots, a_n) \in \mathcal{P},\ a_i = a \right\}$$

be the longest-path value conditioned on all paths that include arc a. Because each index $i \in I$ is uniquely associated with layer $\mathcal{N}_i$, it follows that

$$\lambda_i^j(\boldsymbol{\pi}) = \max_{a \in \mathcal{A}} \{L_a : s(a) \in \mathcal{N}_i, v_a = j\}, \quad \forall i \in I, \forall j \in \{0,1\},$$

and the final d-slacks are obtained by the differences $\lambda_i^0(\boldsymbol{\pi}) - \lambda_i^1(\boldsymbol{\pi})$ for all i.

The values L_a are derived by performing two longest-path computations over $\mathcal{B}$. Specifically, let $\mathcal{A}^{\text{in}}(u)$ and $\mathcal{A}^{\text{out}}(u)$ be the set of incoming and outgoing arcs of a node $u \in \mathcal{N}$, respectively. The solution

of the recursion

$$\begin{aligned} &h(\boldsymbol{\pi}, \mathbf{r}) = 0, \\ &h(\boldsymbol{\pi}, u) = \max_{a \in \mathcal{A}^{\text{in}}(u)} \{h(\boldsymbol{\pi}, s(a)) + \ell_a\}, \qquad \forall u \in \mathcal{N}_i, \forall i \in \{2, \dots, n+1\} \end{aligned}$$

provides the longest-path value $h(\boldsymbol{\pi}, u)$ from $\mathbf{r}$ to u, while the recursion

$$\begin{aligned} &h^{\uparrow}(\boldsymbol{\pi}, u) = \max_{a \in \mathcal{A}^{\text{out}}(u)} \{h^{\uparrow}(\boldsymbol{\pi}, t(a)) + \ell_a\}, \qquad \forall u \in \mathcal{N}_i,\ \forall i \in \{1, \dots, n\}, \\ &h^{\uparrow}(\boldsymbol{\pi}, \mathbf{t}) = 0, \end{aligned}$$

provides the longest-path value $h^{\uparrow}(\boldsymbol{\pi}, u)$ from u to $\mathbf{t}$. The values $h(\boldsymbol{\pi}, u)$ can be calculated via a top-down pass on $\mathcal{B}$, i.e., starting from $\mathbf{r}$ and considering one layer $\mathcal{N}_2, \dots, \mathcal{N}_{n+1}$ at a time. Analogously, the values $h^{\uparrow}(\boldsymbol{\pi}, u)$ are obtained via a bottom-up pass on $\mathcal{B}$, i.e., starting from $\mathbf{t}$ and considering one layer $\mathcal{N}_n, \mathcal{N}_{n-1}, \dots, \mathcal{N}_1$ at a time. Then, the longest-path value for any arc a is given by

$$L_a = h(\boldsymbol{\pi}, s(a)) + \ell_a + h^{\uparrow}(\boldsymbol{\pi}, t(a)).$$

Since each arc is traversed twice via the top-down and bottom-up passes, the complexity of the procedure is $\mathcal{O}(|A|)$.

6.3.3 Sequential Lifting and Dimension Implications

The lifting procedure detailed in Theorem 6.1 can be applied sequentially to strengthen an inequality. Specifically, we start with $\langle \boldsymbol{\pi}, \pi_0 \rangle$ satisfying our main assumptions (a)-(c). Next, we calculate the d-slacks, choose $i \in I$ such that $\lambda_i(\boldsymbol{\pi}) \neq 0$, and apply Theorem 6.1 to obtain the tuple $\langle \boldsymbol{\pi}', \pi_0' \rangle$ defining the lifted inequality. We re-apply this operation with the new $\langle \boldsymbol{\pi}', \pi_0' \rangle$, and repeat until $\lambda_i(\boldsymbol{\pi}) = 0$ for all $i \in I$. The procedure stops in a finite number of iterations since the face dimension increases after each iteration; see property (2) of Theorem 6.1. We summarize the procedure in Algorithm 17.

Algorithm 17 Sequential Combinatorial Lifting Procedure

```
1: procedure CombinatorialLifting(⟨π, π_0⟩, B )
2:     Calculate the disjunctive slacks λ(π)
3:     while λ(π) ≠ 0 do
4:         Choose i ∈ I such that λ_i(π) ≠ 0
5:         Apply Theorem 6.1 to calculate ⟨π', π_0'⟩
6:         Update inequality ⟨π, π_0⟩ = ⟨π', π_0'⟩
7:         Recalculate λ(π)
8:     return ⟨π, π_0⟩
```

The choice of i in step 4 of Algorithm 17 is critical to the dimension of the resulting face, as illustrated in Example 6.3. In particular, arbitrary choices do not necessarily lead to a facet-defining inequality.

Example 6.3 Consider the set $\mathcal{X} = \{x \in \{0,1\}^3 : 5x_1 + 2x_2 + 3x_3 \leq 6\}$ and inequality $\boldsymbol{\pi}^\top \boldsymbol{x} = x_1 + x_2 + x_3 \leq 2$ that supports $\text{conv}(\mathcal{X})$. We have $\boldsymbol{\lambda}(\boldsymbol{\pi}) = (1, -1, -1)^\top$ and the lifted inequality with respect to $\lambda_1(\boldsymbol{\pi}) = 1$ is $\boldsymbol{\pi'}^\top \boldsymbol{x} = 2x_1 + x_2 + x_3 \leq 2$ and has $\boldsymbol{\lambda}(\boldsymbol{\pi}') = \mathbf{0}$. The lifted inequality is not facet-defining since $\dim(\text{conv}(\mathcal{X})) = 3$ and $\dim(F(\boldsymbol{\pi}')) = 1$.

If we instead lift $x_1+x_2+x_3 \leq 2$ with respect to $\lambda_2(\boldsymbol{\pi}) = -1$ the lifted inequality is $\boldsymbol{\pi}'^{\top}\boldsymbol{x} = x_1+x_3 \leq 1$ and $\boldsymbol{\lambda}(\boldsymbol{\pi}') = \mathbf{0}$. In this case, the lifted inequality is facet-defining since $\dim(F(\boldsymbol{\pi}')) = 2$. □

In order to understand the impact of the index choice, we first show in Lemma 6.2 a relationship between d-slacks and the dimension of the face. Specifically, the cardinality of $I^0(\boldsymbol{\pi})$ bounds $\dim(F(\boldsymbol{\pi}))$. We later use this result to gauge when the sequential procedure leads to a facet-defining inequality.

Lemma 6.2. *The dimension of a face $F(\boldsymbol{\pi})$ satisfies $\dim(F(\boldsymbol{\pi})) \leq |I^0(\boldsymbol{\pi})|$. Moreover, $|I^0(\boldsymbol{\pi})| = 0$ if $\dim(F(\boldsymbol{\pi})) = 0$.*

Proof. For any $i \in I^-(\boldsymbol{\pi}) \cup I^+(\boldsymbol{\pi})$, the value of x_i is fixed at either 0 or 1 for all $\boldsymbol{x} \in F(\boldsymbol{\pi})$ according to Lemma 6.1. Thus, the dimension of $\dim(F(\boldsymbol{\pi}))$ is bounded by $|I^0(\boldsymbol{\pi})|$, since at most $|I^0(\boldsymbol{\pi})|+1$ affinely independent points can be obtained from $F(\boldsymbol{\pi})$. Now, consider the case when $\dim(F(\boldsymbol{\pi})) = 0$ and assume that $I^0(\boldsymbol{\pi}) \neq \emptyset$ for the purpose of contradiction. There exist $\boldsymbol{x}, \boldsymbol{x}' \in F(\boldsymbol{\pi})$ such that $x_i \neq x'_i$ for any $i \in I^0(\boldsymbol{\pi})$. These two points are affinely independent and therefore $\dim(F(\boldsymbol{\pi})) \geq 1$. Thus, $I^0(\boldsymbol{\pi}) = \emptyset$ if $\dim(F(\boldsymbol{\pi})) = 0$. ■

Example 6.3 depicts a case where $|I^0(\boldsymbol{\pi})|$ increases faster than the number of affinely independent points in $F(\boldsymbol{\pi})$. In view of Lemma 6.2, we would like to choose i so that $|I^0(\boldsymbol{\pi})|$ increases at a "slower rate", since each lifting operation increases $\dim(F(\boldsymbol{\pi}))$ by at least one according to Theorem 6.1-(2). We show in Theorem 6.2 that the slow increase of $|I^0(\boldsymbol{\pi})|$ occurs when there exists a unique slack with minimum non-zero absolute value.

Theorem 6.2. *Suppose there exists $i \notin I^0(\boldsymbol{\pi})$ such that $|\lambda_i(\boldsymbol{\pi})| < |\lambda_{i'}(\boldsymbol{\pi})|$ for all $i' \notin I^0(\boldsymbol{\pi})$ $(i' \neq i)$. Then, for $\langle \boldsymbol{\pi}', \pi'_0 \rangle$ obtained when lifting $\langle \boldsymbol{\pi}, \pi_0 \rangle$ with respect to $\lambda_i(\boldsymbol{\pi})$, $\dim(F(\boldsymbol{\pi}')) = \dim(F(\boldsymbol{\pi})) + 1$ and $|I^0(\boldsymbol{\pi}')| = |I^0(\boldsymbol{\pi})| + 1$.*

Proof. From Lemma 6.1, it suffices to show that, for any $\boldsymbol{x} \in F(\boldsymbol{\pi}')$ and $i' \notin I^0(\boldsymbol{\pi})$ such that $i' \neq i$, we have: 1) $i' \in I^+(\boldsymbol{\pi})$ implies that $x_{i'} = 0$; and 2) $i' \in I^-(\boldsymbol{\pi})$ implies that $x_{i'} = 1$. In such cases, an index i' that was originally in $I^-(\boldsymbol{\pi})$ or $I^+(\boldsymbol{\pi})$ will remain in its original partition $I^-(\boldsymbol{\pi}')$ or $I^+(\boldsymbol{\pi}')$ for the lifted $\boldsymbol{\pi}'$. The statement then follows due to Theorem 6.1-(2) and Lemma 6.2.

We will focus our attention to the case $\lambda_i(\boldsymbol{\pi}) > 0$ (the others are analogous). For any $\boldsymbol{x} \in F(\boldsymbol{\pi}')$, we have by construction that $\boldsymbol{\pi}^{\top}\boldsymbol{x} = \pi'_0 - \lambda_i(\boldsymbol{\pi})x_i \geq \pi_0 - \lambda_i(\boldsymbol{\pi})$. Assume, for the purpose of a contradiction, that $x_{i'} = 1$ and that $\lambda_{i'}(\boldsymbol{\pi}) > 0$. Thus, $\lambda^1_{i'}(\boldsymbol{\pi}) \geq \boldsymbol{\pi}^{\top}\boldsymbol{x} \geq \pi_0 - \lambda_i(\boldsymbol{\pi})$. Moreover, $\lambda^0_{i'}(\boldsymbol{\pi}) = \pi_0$. This implies that $\lambda_{i'}(\boldsymbol{\pi}) = \lambda^0_{i'}(\boldsymbol{\pi}) - \lambda^1_{i'}(\boldsymbol{\pi}) \leq \pi_0 - \pi_0 + \lambda_i(\boldsymbol{\pi}) \leq \lambda_i(\boldsymbol{\pi})$ and hence $0 < \lambda_{i'}(\boldsymbol{\pi}) \leq \lambda_i(\boldsymbol{\pi})$. This cannot hold since $|\lambda_i(\boldsymbol{\pi})| < |\lambda_{i'}(\boldsymbol{\pi})|$.

Similarly, assume that $\lambda_{i'}(\boldsymbol{\pi}) < 0$ and $x_{i'} = 0$. Then, $\lambda^0_{i'}(\boldsymbol{\pi}) \geq \pi_0 - \lambda_i(\boldsymbol{\pi})$ and $\lambda^1_{i'}(\boldsymbol{\pi}) = \pi_0$. This implies that $\lambda_{i'}(\boldsymbol{\pi}) = \lambda^0_{i'}(\boldsymbol{\pi}) - \lambda^1_{i'}(\boldsymbol{\pi}) \geq \pi_0 - \lambda_i(\boldsymbol{\pi}) - \pi_0 = -\lambda_i(\boldsymbol{\pi})$. Thus, $0 > \lambda_{i'}(\boldsymbol{\pi}) \geq -\lambda_i(\boldsymbol{\pi})$. This contradicts $|\lambda_i(\boldsymbol{\pi})| < |\lambda_{i'}(\boldsymbol{\pi})|$. ■

Theorem 6.2 provides a simple choice rule based on picking i with the minimum absolute slack. It also indicates when this rule will converge to a facet-defining inequality. We formalize it in Corollary 6.1 below, which directly follows from Theorem 6.2.

Corollary 6.1. *If $\dim(F(\boldsymbol{\pi})) = |I^0(\boldsymbol{\pi})|$, the sequential lifting procedure (Algorithm 17) with the minimum slack absolute rule produces a facet-defining inequality if, at each lifting iteration except the last, the chosen $i \in I$ is such that $|\lambda_i(\boldsymbol{\pi})| < |\lambda_{i'}(\boldsymbol{\pi})|$ for all $i' \notin I^0(\boldsymbol{\pi})$ $(i' \neq i)$.*

Finally, we note that, in general, it may not be possible to achieve a facet-defining inequality regardless of the index choice. For example, all non-zero d-slacks can have the same absolute value and the cardinality of $|I^0(\boldsymbol{\pi})|$ might increase by more than one while the dimension of $F(\boldsymbol{\pi})$ does not (see Example 6.3).

6.3.4 Relationship with Lifting Based on Disjunctive Programming

We now formalize the connection between our lifting methodology and the n-step lifting procedure by Perregaard and Balas (2001) mentioned in Section 6.2. Assume that $\mathcal{X} = \{A\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{x} \in \{0,1\}^n\}$ for a matrix A and vector $\boldsymbol{b}$ of appropriate dimensions. We consider a relaxation of the form DP-L that includes one disjunctive term for each index $i \in I$ and removes all integrality constraints:

$$\max_{\boldsymbol{x}} \left\{ \boldsymbol{c}^\top \boldsymbol{x} :\ A\boldsymbol{x} \leq \boldsymbol{b},\ \bigvee_{i' \in I} (x_{i'} \leq 0) \vee (x_{i'} \geq 1),\ \boldsymbol{x} \in [0,1]^n \right\}. \tag{DP-L}$$

Proposition 6.1 below shows that, for DP-L, the optimal parameter in the n-step lifting is such that $\gamma^* = |\lambda_i(\boldsymbol{\pi})|$ when using the individual binary disjunctions as target inequalities.

Proposition 6.1. *Suppose $\lambda_i(\boldsymbol{\pi}) \neq 0$ for some $i \in I$. Then, $\gamma^* = |\lambda_i(\boldsymbol{\pi})|$ if we employ either $x_i \leq 0$ or $x_i \geq 1$ as a target inequality in the n-step procedure.*

Proof. Consider the case when $i \in I^+(\boldsymbol{\pi})$ and let $\boldsymbol{e}_i$ be the i-th column of an $n \times n$ identity matrix. Since all $\boldsymbol{x} \in F(\boldsymbol{\pi})$ have $x_i = 0$, $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = \boldsymbol{e}_i^\top \boldsymbol{x} = x_i \leq 0$ is an invalid target inequality for $\text{conv}(\mathcal{X})$ and satisfies $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = x_i = 0$ for all $\boldsymbol{x} \in F(\boldsymbol{\pi})$. The system that defines γ^* is therefore

$$\gamma^* = \min_{\boldsymbol{x} \in [0,1]^n, x_0 \geq 0} \left\{ \pi_0 x_0 - \boldsymbol{\pi}^\top \boldsymbol{x} :\ A\boldsymbol{x} \leq \boldsymbol{b},\ -x_i = -1,\ \bigvee_{i' \in I} (x_{i'} \leq 0) \vee (-x_{i'} + x_0 \leq 0) \right\}. \tag{6.1}$$

It follows from (6.1) and $x_i = 1$ that $x_0 \leq 1$. Without loss of generality, we consider $x_0 = 1$. The system reduces to:

$$\begin{aligned}
\gamma^* &= \min_{\boldsymbol{x}, x_0} \left\{ \pi_0 x_0 - \boldsymbol{\pi}^\top \boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b},\ x_i = 1,\ \boldsymbol{x} \in \{0,1\}^n,\ x_0 = 1 \right\} \\
&= \pi_0 - \max_{\boldsymbol{x}} \{ \boldsymbol{\pi}^\top \boldsymbol{x} : \boldsymbol{x} \in \mathcal{X},\ x_i = 1 \} \\
&= \lambda_i^0(\boldsymbol{\pi}) - \lambda_i^1(\boldsymbol{\pi}) = \lambda_i(\boldsymbol{\pi}).
\end{aligned}$$

The last equality comes from $i \in I^+(\boldsymbol{\pi})$ and $\lambda_i^0(\boldsymbol{\pi}) = \pi_0$. The proof for $i \in I^-(\boldsymbol{\pi})$ and target inequality $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = x_i \geq 1$ is analogous. ■

Proposition 6.1 indicates when these two techniques are equivalent. By taking $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = x_i \leq 0$ as the target inequality, we obtain $\gamma^* = \lambda_i(\boldsymbol{\pi}) > 0$. The lifted inequality $(\boldsymbol{\pi} + \gamma^* \widetilde{\boldsymbol{\pi}})^\top \boldsymbol{x} = (\boldsymbol{\pi} + \lambda_i(\boldsymbol{\pi}) \boldsymbol{e}_i)^\top \boldsymbol{x} \leq \pi_0$ is equivalent to the lifted inequality in Theorem 6.1. Similarly, using target inequality $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = -x_i \leq -1$ would result in $\gamma^* = -\lambda_i(\boldsymbol{\pi}) > 0$. Then, the lifted inequalities are equivalent, i.e., $\boldsymbol{\pi} + \gamma^* \widetilde{\boldsymbol{\pi}} = \boldsymbol{\pi} + \lambda_i(\boldsymbol{\pi}) \boldsymbol{e}_i$ and $\pi_0 + \gamma^* \widetilde{\pi}_0 = \pi_0 + \lambda_i(\boldsymbol{\pi})$.

While the techniques are equivalent in this restricted case, our approach is valid for any binary set $\mathcal{X}$ and, thus, can handle models where a BDD (or BDD relaxation) is a more advantageous representation

in comparison to a linear description of $\mathcal{X}$ (Bergman and Cire, 2018). We also note that the BDD network structure allows us to efficiently compute the disjunctive terms in a combinatorial fashion.

6.4 A Combinatorial Cutting-Plane Algorithm

While the BDD-based lifting procedure developed in Section 6.3 can enhance inequalities from any cutting-plane methodology, we now exploit similar concepts to derive new valid inequalities for $\mathcal{X}$ based on the network structure of $\mathcal{B}$. In particular, we design inequalities that separate points from $\mathcal{X}$ by only relying on the combinatorial structure encoded by $\mathcal{B}$. Thus, no other specific structure (e.g., linearity, submodularity, or gradient information) is required.

We assume, as before, that we are given an exact BDD $\mathcal{B}$ for $\mathcal{X}$, i.e., $\text{Sol}(\mathcal{B}) = \mathcal{X}$. Our cutting-plane method is based on a novel linear description of $\mathcal{B}$ as an extended capacitated flow problem. We present this formulation in Section 6.4.1 and our BDD-based cut generation linear program in Section 6.4.2. For cases where solving such model is not computationally practical, in Section 6.4.3 we develop a weaker but more efficient combinatorial cutting-plane method based on a max-flow/min-cut problem over $\mathcal{B}$. Finally, we show in Section 6.4.4 the relationship between our approach and existing BDD cutting-plane techniques (Davarnia and van Hoeve, 2020; Tjandraatmadja and van Hoeve, 2019).

6.4.1 A New BDD Polytope

Existing BDD-based cut generation procedures (Lozano and Smith, 2018; Tjandraatmadja and van Hoeve, 2019; Davarnia and van Hoeve, 2020) rely on the network flow formulation $\text{NF}(\mathcal{B})$ introduced by Behle (2007). The network flow model $\text{NF}(\mathcal{B})$ is given by a continuous relaxation of the original variables, $\boldsymbol{x} \in [0,1]^n$, and a set of variables $\boldsymbol{y} \in \mathbb{R}_+^{|\mathcal{A}|}$ that represent the flow traversing each BDD arc. Intuitively, the flow over each path $p \in \mathcal{P}$ can be seen as the weight of its corresponding point $\boldsymbol{x}^p$. Then, by restricting the total flow to have value one, the flow variables represent a convex combination of the points in $\text{Sol}(\mathcal{B})$.

$$
\begin{aligned}
\text{NF}(\mathcal{B}) = \big\{ (\boldsymbol{x};\boldsymbol{y}) \in [0,1]^n \times \mathbb{R}_+^{|\mathcal{A}|} : \\
\sum_{a \in \mathcal{A}^{\text{out}}(u)} y_a - \sum_{a \in \mathcal{A}^{\text{in}}(u)} y_a = 0, \qquad & \forall u \in \mathcal{N} \setminus \{\mathbf{r}, \mathbf{t}\}, && \text{(6.2a)} \\
\sum_{a \in \mathcal{A}^{\text{out}}(\mathbf{r})} y_a = \sum_{a \in \mathcal{A}^{\text{in}}(\mathbf{t})} y_a = 1, & && \text{(6.2b)} \\
\sum_{a \in \mathcal{A}: s(a) \in \mathcal{N}_i, v_a = 1} y_a = x_i, & \forall i \in I \big\}. && \text{(6.2c)}
\end{aligned}
$$

Equalities (6.2a) and (6.2b) are balance-of-flow constraints over $\mathcal{B}$. Constraint (6.2c) links the arcs of $\mathcal{B}$ with solutions $\boldsymbol{x}$. In particular, the polytope $\text{NF}(\mathcal{B})$ projected over the $\boldsymbol{x}$ variables is equivalent to the convex hull of all solutions represented by $\mathcal{B}$, i.e., $\text{Proj}_{\text{x}}(\text{NF}(\mathcal{B})) = \text{conv}(\mathcal{X})$ (Behle, 2007). Note that $\text{NF}(\mathcal{B})$ is a special case of the network flow model presented in Chapter 3 for general DDs.

One of the main drawbacks of $\text{NF}(\mathcal{B})$ is that constraint (6.2c) only considers flow variables associated with arc labels equal to one (i.e., $v_a = 1$). Thus, there is no constraint explicitly limiting the flow passing through arcs with label equal to zero. As a consequence, existing cut generation procedures based on $\text{NF}(\mathcal{B})$ have to solve an unbounded linear programming model and these procedure consider special cases

for unbounded solutions (Tjandraatmadja and van Hoeve, 2019) or introduce normalization constraints to avoid them (Davarnia and van Hoeve, 2020).

We propose an alternative formulation of $\mathrm{NF}(\mathcal{B})$ that addresses its main limitations and use the reformulation to define our cutting-plane algorithms. The new formulation, here denoted by $\mathrm{JNF}(\mathcal{B})$, corresponds to a joint capacitated network-flow polytope. The new model maintains the flow conservation constraints, (6.2a) and (6.2b), and replaces (6.2c) with (6.3a) and (6.3b) below. Both inequalities enforce a common capacity for arcs in a layer with the same value and, thus, constrain the flow of all arcs in $\mathcal{B}$. Proposition 6.2 shows that the two formulations are equivalent and, thus, $\mathrm{Proj}_{\mathrm{x}}(\mathrm{JNF}(\mathcal{B})) = \mathrm{conv}(\mathcal{X})$.

$$\mathrm{JNF}(\mathcal{B}) = \big\{(\boldsymbol{x};\boldsymbol{y}) \in [0,1]^n \times \mathbb{R}_+^{|\mathcal{A}|} : (6.2a) - (6.2b),$$

$$\sum_{a\in\mathcal{A}: s(a)\in\mathcal{N}_i, v_a=1} y_a \le x_i, \qquad \forall i \in I, \tag{6.3a}$$

$$\sum_{a\in\mathcal{A}: s(a)\in\mathcal{N}_i, v_a=0} y_a \le 1 - x_i, \qquad \forall i \in I \big\}. \tag{6.3b}$$

Proposition 6.2. *$JNF(\mathcal{B}) = NF(\mathcal{B})$.*

Proof. Consider $(\boldsymbol{x}';\boldsymbol{y}') \in \mathrm{NF}(\mathcal{B})$, so $(\boldsymbol{x}';\boldsymbol{y}')$ satisfies (6.2a) and (6.2b). Since (6.2c) holds, $(\boldsymbol{x}';\boldsymbol{y}')$ also satisfies (6.3a). From the flow conservation constraints, (6.2a) and (6.2b), the flow traversing each layer $\mathcal{N}_i$ is exactly one, i.e.,

$$\sum_{a\in\mathcal{A}:\ s(a)\in\mathcal{N}_i} y'_a = 1 \quad \Rightarrow \quad \sum_{a\in\mathcal{A}: s(a)\in\mathcal{N}_i:\ v_a=1} y'_a + \sum_{a\in\mathcal{A}: s(a)\in\mathcal{N}_i:\ v_a=0} y'_a = 1$$

$$\Rightarrow \quad \sum_{a\in\mathcal{A}: s(a)\in\mathcal{N}_i:\ v_a=0} y'_a = 1 - x'_i.$$

Then, $(\boldsymbol{x}';\boldsymbol{y}') \in \mathrm{JNF}(\mathcal{B})$. Consider now $(\boldsymbol{x}';\boldsymbol{y}') \in \mathrm{JNF}(\mathcal{B})$. Since flows traversing a layer sum to one, constraints (6.3a) and (6.3b) are satisfied as equalities and therefore (6.2c) holds for $(\boldsymbol{x}';\boldsymbol{y}')$. ■

6.4.2 General BDD Flow Cuts

Our cutting-plane procedure formulates a max-flow optimization problem over $\mathrm{JNF}(\mathcal{B})$ to identify and separate points $\boldsymbol{x}' \notin \mathrm{conv}(\mathcal{X})$, given by (6.4) below:

$$z(\mathcal{B};\boldsymbol{x}') = \max_{\boldsymbol{y}\in\mathbb{R}_+^{|\mathcal{A}|}} \left\{ \sum_{a\in\mathcal{A}^{\mathrm{out}}(\mathbf{r})} y_a :\ (6.2a), (6.3a) - (6.3b),\ \boldsymbol{x} = \boldsymbol{x}' \right\}. \tag{6.4}$$

We note that (6.4) omits the constraint enforcing the flow to be equal to one as in (6.2b). We argue in Lemma 6.3 that the condition $z(\mathcal{B};\boldsymbol{x}') = 1$ is necessary and sufficient to check if $\boldsymbol{x}'$ belongs to $\mathrm{conv}(\mathcal{X})$.

Lemma 6.3. *$\boldsymbol{x}' \in \mathrm{conv}(\mathcal{X})$ if and only if $z(\mathcal{B};\boldsymbol{x}') = 1$.*

Proof. Constraints (6.3a) and (6.3b) enforce that the flow in each layer i is at most $x'_i + 1 - x'_i = 1$. Thus, $z(\mathcal{B};\boldsymbol{x}') \le 1$. Consider $\boldsymbol{x}' \in \mathrm{conv}(\mathcal{X})$. From Proposition 6.2, there exists $\boldsymbol{y}' \in \mathbb{R}_+^{|\mathcal{A}|}$ such that $\sum_{a\in\mathcal{A}^{\mathrm{out}}(\mathbf{r})} y'_a = 1$ and therefore $z(\mathcal{B};\boldsymbol{x}') = 1$. For the converse, suppose $z(\mathcal{B};\boldsymbol{x}') = 1$. It follows that there exists $\boldsymbol{y}' \in \mathbb{R}_+^{|\mathcal{A}|}$ such that $(\boldsymbol{x}';\boldsymbol{y}') \in \mathrm{JNF}(\mathcal{B}) = \mathrm{NF}(\mathcal{B})$, so $\boldsymbol{x}' \in \mathrm{conv}(\mathcal{X})$. ■

Our BDD-based cut generation linear program (CGLP) uses the dual of (6.4) to generate valid inequalities that cut off $\boldsymbol{x}' \notin \text{conv}(\mathcal{X})$. Consider $\boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|}$ as the dual variables associated with constraints (6.2a), and $\boldsymbol{\nu}, \boldsymbol{\eta} \in \mathbb{R}^n_+$ as the dual variables for (6.3a) and (6.3b), respectively. The resulting model is

$$\min_{\boldsymbol{\omega},\boldsymbol{\nu},\boldsymbol{\eta}} \quad \sum_{i\in I} x'_i\nu_i + \sum_{i\in I}(1-x'_i)\eta_i \tag{BDD-CGLP}$$

$$\text{s.t.} \quad \omega_{t(a)} - \omega_{s(a)} + v_a\nu_i + (1-v_a)\eta_i \geq 0, \qquad \forall i \in I, a \in \mathcal{A}, s(a) \in \mathcal{N}_i, \tag{6.5a}$$

$$\omega_{t(a)} + v_a\nu_1 + (1-v_a)\eta_1 \geq 1, \qquad \forall a \in \mathcal{A}^{\text{out}}(\mathbf{r}), \tag{6.5b}$$

$$-\omega_{s(a)} + v_a\nu_n + (1-v_a)\eta_n \geq 0, \qquad \forall a \in \mathcal{A}^{\text{in}}(\mathbf{t}), \tag{6.5c}$$

$$\boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|},\ \boldsymbol{\nu}, \boldsymbol{\eta} \in \mathbb{R}^n_+. \tag{6.5d}$$

Let $w(\mathcal{B}; \boldsymbol{x}')$ be the optimal solution value of BDD-CGLP. Strong duality and Lemma 6.3 imply that we can identify if a point $\boldsymbol{x}'$ belongs to $\text{conv}(\mathcal{X})$ if $w(\mathcal{B}; \boldsymbol{x}') = 1$. Furthermore, we can use the optimal solution $(\boldsymbol{\nu}^*; \boldsymbol{\eta}^*)$ to create a valid cut when $w(\mathcal{B}; \boldsymbol{x}') < 1$. Specifically, the cut is given by

$$\sum_{i\in I} x_i\nu^*_i + \sum_{i\in I}(1-x_i)\eta^*_i \geq 1. \tag{6.6}$$

Theorem 6.3 shows that the set of all cuts of the form (6.6) describes $\text{conv}(\mathcal{X})$.

Theorem 6.3. *Let $\Lambda(\mathcal{B})$ be the set of extreme points of the BDD-CGLP polyhedron defined by (6.5a)-(6.5d). Furthermore, let $P_{\mathcal{B}}$ be the set of points $\boldsymbol{x} \in [0,1]^n$ that satisfy (6.6) for all $(\boldsymbol{\nu}; \boldsymbol{\eta}) \in \text{Proj}_{\boldsymbol{\nu},\boldsymbol{\eta}}(\Lambda(\mathcal{B}))$. Then, $\text{conv}(\mathcal{X}) = P_{\mathcal{B}}$.*

Proof. Consider a point $\boldsymbol{x}' \in \text{conv}(\mathcal{X})$. Lemma 6.3 guarantees that $z(\mathcal{B}; \boldsymbol{x}') = w(\mathcal{B}; \boldsymbol{x}') = 1$, so constraint (6.6) holds for any extreme point of BDD-CGLP. Now consider a point $\boldsymbol{x}' \in P_{\mathcal{B}}$. Since $\boldsymbol{x}'$ satisfies (6.6) for all extreme points in $\Lambda(\mathcal{B})$, we have that $w(\mathcal{B}; \boldsymbol{x}') \geq 1$ and, thus, $w(\mathcal{B}; \boldsymbol{x}') = z(\mathcal{B}; \boldsymbol{x}') = 1$. Finally, using Lemma 6.3, we have that $\boldsymbol{x}' \in \text{conv}(\mathcal{X})$. ■

Thus, our cutting-plane procedure separates points $\boldsymbol{x}' \notin \text{conv}(\mathcal{X})$ by solving BDD-CGLP. The procedure returns a cut of the form (6.6) where $(\boldsymbol{\nu}^*; \boldsymbol{\eta}^*)$ is an optimal solution of $w(\mathcal{B}; \boldsymbol{x}')$.

6.4.3 Combinatorial BDD Flow Cuts

The above cutting-plane procedure requires solving a linear program with $|\mathcal{A}|$ constraints and $|\mathcal{N}| + 2n$ variables. Obtaining $w(\mathcal{B}; \boldsymbol{x}')$, thus, could be computationally expensive for instances where $\mathcal{B}$ is large. We propose now an alternative cut-generation procedure based on BDD-CGLP that involves a combinatorial and more efficient max-flow solution over $\mathcal{B}$.

First, we consider a reformulation of $\text{JNF}(\mathcal{B})$ where the joint capacity constraints are replaced by individual constraints for each arc, i.e., a standard capacitated network flow polytope over $\mathcal{B}$:

$$\text{CNF}(\mathcal{B}) = \big\{(\boldsymbol{x}; \boldsymbol{y}) \in [0,1]^n \times \mathbb{R}^{|\mathcal{A}|}_+ : (6.2a) - (6.2b),$$

$$y_a \leq x_i, \qquad \forall a \in \mathcal{A}, s(a) \in \mathcal{N}_i, v_a = 1, i \in I, \tag{6.7a}$$

$$y_a \leq 1 - x_i, \qquad \forall a \in \mathcal{A}, s(a) \in \mathcal{N}_i, v_a = 0, i \in I\big\}. \tag{6.7b}$$

Proposition 6.3. *$\text{JNF}(\mathcal{B}) \subseteq \text{CNF}(\mathcal{B})$ and for any integer $\boldsymbol{x} \notin \text{conv}(\mathcal{X})$ we have $\boldsymbol{x} \notin \text{Proj}_{\text{x}}(\text{CNF}(\mathcal{B}))$.*

Proof. Consider $\boldsymbol{x}' \in \text{JNF}(\mathcal{B})$. By construction, $\boldsymbol{x}'$ satisfies (6.2a)-(6.2b). Since $\boldsymbol{x}'$ satisfies (6.3a) and (6.3b), it follows that $\boldsymbol{x}'$ holds for (6.7a) and (6.7b).

Now take an integer point $\boldsymbol{x}' \notin \text{conv}(\mathcal{X})$ and $\boldsymbol{y}' \in \mathbb{R}_+^{|\mathcal{A}|}$ such that $(\boldsymbol{x}'; \boldsymbol{y}')$ satisfies (6.2a), (6.7a)-(6.7b). Notice that such a $\boldsymbol{y}'$ exists (e.g., $\boldsymbol{y}' = \boldsymbol{0}$). By construction, there is no path $p \in \mathcal{P}$ associated with $\boldsymbol{x}'$. From constraints (6.7a)-(6.7b), in any path $p \in \mathcal{P}$ there exists an arc $a \in p$ with capacity zero (i.e., $y_a \leq 0$). We can then deduce that $\boldsymbol{y}' = \boldsymbol{0}$, therefore $(\boldsymbol{x}'; \boldsymbol{y}')$ violates (6.2b). Finally, for any $\boldsymbol{x}' \in \{0,1\}^n \setminus \text{conv}(\mathcal{X})$ there is no $\boldsymbol{y}' \in \mathbb{R}_+^{|\mathcal{A}|}$ such that $(\boldsymbol{x}'; \boldsymbol{y}') \in \text{CNF}(\mathcal{B})$. ■

Proposition 6.3 shows that for any integer point $\boldsymbol{x}'$, $\boldsymbol{x}' \notin \mathcal{X}$ implies $\boldsymbol{x}' \notin \text{Proj}_{\text{x}}(\text{CNF}(\mathcal{B}))$. Example 6.4 illustrates that, conversely, there might exist fractional points $\boldsymbol{x}' \notin \text{conv}(\mathcal{X})$ such that $\boldsymbol{x}' \in \text{Proj}_{\text{x}}(\text{CNF}(\mathcal{B}))$, and hence $\text{CNF}(\mathcal{B})$ is a weaker representation than $\text{JNF}(\mathcal{B})$.

Example 6.4 Consider our example $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$, a fractional point $\boldsymbol{x}' = (0.4, 0.6, 0.4, 1)$, and the exact BDD $\mathcal{B}_1$ in Figure 6.1. It is easy to see that $\boldsymbol{x}' \notin \text{conv}(\mathcal{X})$ since $7x'_1 + 5x'_2 + 4x'_3 + x'_4 = 8.4 \geq 8$. However, there exists a $\boldsymbol{y}' \in \mathbb{R}_+^{|\mathcal{A}|}$ such that $(\boldsymbol{x}', \boldsymbol{y}') \in \text{CNF}(\mathcal{B})$ with value $y_{(\mathbf{r},u_1)} = 0.6$, $y_{(\mathbf{r},u_2)} = 0.4$, $y_{(u_1,u_4)} = 0.2$, $y_{(u_1,u_3)} = 0.4$, $y_{(u_2,u_4)} = 0.4$, $y_{(u_3,u_4)} = 0.4$, $y_{(u_4,u_5)} = 0.6$, $y_{(u_5,\mathbf{t})} = 1$, and all other arcs with flow equal to zero. □

Similar to the general BDD flow cuts, we use the dual of the max-flow version of $\text{CNF}(\mathcal{B})$ to identify points that do not belong to $\text{conv}(\mathcal{X})$. Consider $\boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|}$ as the dual variables associated with constraints (6.2a) and $\boldsymbol{\alpha}$ the dual variables associated with constraints (6.7a)-(6.7b). Then, the separation problem for the alternative BDD cuts is as follow:

$$
\begin{aligned}
\min_{\boldsymbol{\omega},\boldsymbol{\alpha}} \quad & \sum_{i \in I} \left(\sum_{a \in \mathcal{A}: s(a) \in \mathcal{N}_i, v_a = 1} x'_i \alpha_a \; + \sum_{a \in \mathcal{A}: s(a) \in \mathcal{N}_i, v_a = 0} (1 - x'_i) \alpha_a \right) && && \text{(CN-CGLP)} \\
\text{s.t.} \quad & \omega_{t(a)} - \omega_{s(a)} + \alpha_a \geq 0, && \forall i \in I, a \in \mathcal{A}, s(a) \in \mathcal{N}_i, \\
& \omega_{t(a)} + \alpha_a \geq 1, && \forall a \in \mathcal{A}^{\text{out}}(\mathbf{r}), \\
& -\omega_{s(a)} + \alpha_a \geq 0, && \forall a \in \mathcal{A}^{\text{in}}(\mathbf{t}), \\
& \boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|}, \; \boldsymbol{\alpha} \in \mathbb{R}_+^{|\mathcal{A}|}.
\end{aligned}
$$

Let $w^{\text{r}}(\mathcal{B}; \boldsymbol{x}')$ be the optimal solution value of CN-CGLP. Proposition 6.3 implies that for any $\boldsymbol{x}' \in \text{conv}(\mathcal{X})$, $w^{\text{r}}(\mathcal{B}; \boldsymbol{x}') = 1$. It follows that inequality (6.9) holds for any $\boldsymbol{x} \in \text{conv}(\mathcal{X})$, where $\boldsymbol{\alpha}^*$ is optimal to CN-CGLP:

$$
\sum_{i \in I} \left(\sum_{a \in \mathcal{A}: s(a) \in \mathcal{N}_i, v_a = 1} x_i \alpha^*_a + \sum_{a \in \mathcal{A}: s(a) \in \mathcal{N}_i, v_a = 0} (1 - x_i) \alpha^*_a \right) \geq 1. \tag{6.9}
$$

Of important note is that CN-CGLP is a classical min-cut problem, i.e., we are searching for a maximum-capacity arc cut in the network that certifies that a point does not belong to the convex hull of $\mathcal{X}$. While the resulting inequalities are not as strong as the general BDD cuts from BDD-CGLP, we can leverage max-flow/min-cut combinatorial algorithms to solve it more efficiently in the size of the BDD. Several algorithms are readily available to that end (Ahuja et al., 1993) and provide both primal and dual solutions to CN-CGLP.

Furthermore, another consequence of the design of such cuts is that their strength depends on the BDD size. That is, two BDDs $\mathcal{B}$ and $\mathcal{B}'$ encoding the same set $\mathcal{X}$ might generate different combinatorial

flow cuts because of distinct min-cut solutions. We show in Theorem 6.4 that the reduced BDD, which is unique, generates the tightest CNF($\mathcal{B}$) formulation and is hence critical in such a formulation. We note that a reduced BDD can be generated in polynomial time in $\mathcal{B}'$ for any $\mathcal{B}'$ representing the desired solution set (Bryant, 1986).

Theorem 6.4. *Let $\mathcal{B}^{\mathrm{r}} = (\mathcal{N}^{\mathrm{r}}, \mathcal{A}^{\mathrm{r}})$ be the reduced version of $\mathcal{B}$, i.e., $\mathrm{Sol}(\mathcal{B}^{\mathrm{r}}) = \mathrm{Sol}(\mathcal{B})$, and for each layer $i \in I$, $|\mathcal{N}_i^{\mathrm{r}}| \leq |\mathcal{N}_i|$. Then, CNF($\mathcal{B}^{\mathrm{r}}$) $\subseteq$ CNF($\mathcal{B}$).*

Proof. Consider $\mathcal{P}^{\mathrm{r}}$ to be the set of $\mathbf{r}-\mathbf{t}$ paths in $\mathcal{B}^{\mathrm{r}}$. First, $\mathrm{Sol}(\mathcal{B}^{\mathrm{r}}) = \mathrm{Sol}(\mathcal{B})$ implies that, for any $\mathbf{r}-\mathbf{t}$ path $p \in \mathcal{P}$, there exists a unique $\mathbf{r}-\mathbf{t}$ path $p' \in \mathcal{P}^{\mathrm{r}}$ such that $\boldsymbol{x}^p = \boldsymbol{x}^{p'}$. Thus, we will consider that the set of paths in both BDDs are equivalent, i.e., $\mathcal{P}^{\mathrm{r}} = \mathcal{P}$.

Let $\mathcal{A}_i = \{a \in \mathcal{A} \colon s(a) \in \mathcal{N}_i\}$ and $\mathcal{A}_i^{\mathrm{r}} = \{a \in \mathcal{A}^{\mathrm{r}} \colon s(a) \in \mathcal{N}_i^{\mathrm{r}}\}$. Since $\mathcal{B}^{\mathrm{r}}$ is unique, there exists a unique surjective function $f_i : \mathcal{A}_i \to \mathcal{A}_i^{\mathrm{r}}$ that maps arcs from $\mathcal{B}$ to $\mathcal{B}^{\mathrm{r}}$ for each layer $i \in I$. Thus, for every arc $a \in \mathcal{A}_i^{\mathrm{r}}$, let us define the pre-image of f_i as $f_i^{-1}(a) = \{a' \in \mathcal{A}_i : \ f(a') = a\}$, i.e., the subset of arcs in $\mathcal{A}_i$ that map to arc $a \in \mathcal{A}_i^{\mathrm{r}}$. Next, denote by $\Gamma(\mathcal{B}; a) = \{p \in \mathcal{B} : a \in p\}$ the set of paths in a BDD that traverse an arc $a \in \mathcal{A}$. From the construction procedure of $\mathcal{B}^{\mathrm{r}}$ given $\mathcal{B}$ (Bryant, 1986), $\Gamma(\mathcal{B}^{\mathrm{r}}; a) = \bigcup_{a' \in f_i^{-1}(a)} \Gamma(\mathcal{B}; a')$ for all $a \in \mathcal{A}_i^{\mathrm{r}}$, i.e., the set of $\mathbf{r}-\mathbf{t}$ paths passing through a is equivalent to the set of $\mathbf{r}-\mathbf{t}$ paths passing through all the arcs in $f_i^{-1}(a)$.

Now consider the path formulation of CNF($\mathcal{B}$), $\mathrm{CNF}^P(\mathcal{B})$. It suffices to show that $\mathrm{CNF}^P(\mathcal{B}^{\mathrm{r}}) \subseteq \mathrm{CNF}^P(\mathcal{B})$. Since the paths in $\mathcal{B}$ and $\mathcal{B}^{\mathrm{r}}$ are equivalent, we will consider equivalent variables $\boldsymbol{w}$ for $\mathrm{CNF}^P(\mathcal{B}^{\mathrm{r}})$ and $\mathrm{CNF}^P(\mathcal{B})$.

$$\mathrm{CNF}^P(\mathcal{B}) = \Big\{(\boldsymbol{x}; \boldsymbol{w}) \in [0,1]^n \times \mathbb{R}_+^{|\mathcal{P}|} : \\ \sum_{p \in \mathcal{P} : a \in p} w_p \leq x_i, \qquad \forall a \in \mathcal{A}_i, v_a = 1, i \in I, \tag{6.10a}$$

$$\sum_{p \in \mathcal{P} : a \in p} w_p \leq 1 - x_i, \qquad \forall a \in \mathcal{A}_i, v_a = 0, i \in I \Big\}. \tag{6.10b}$$

Using the path equivalence $\Gamma(\mathcal{B}^{\mathrm{r}}; a) = \bigcup_{a' \in f_i^{-1}(a)} \Gamma(\mathcal{B}; a')$ for any $a \in \mathcal{A}_i^{\mathrm{r}}$ and $i \in I$, we have that

$$\sum_{p \in \mathcal{P}^{\mathrm{r}} : a \in p} w_p = \sum_{a' \in f_i^{-1}(a)} \sum_{p \in \mathcal{P} : a' \in p} w_p, \qquad \forall a \in \mathcal{A}_i^{\mathrm{r}}, \ i \in I.$$

Thus, constraints (6.10a) and (6.10b) of $\mathrm{CNF}^P(\mathcal{B}^{\mathrm{r}})$ are tighter since they restrict more paths than for the case of $\mathrm{CNF}^P(\mathcal{B})$. This implies that, for any $(\boldsymbol{x}'; \boldsymbol{w}') \in \mathrm{CNF}^P(\mathcal{B}^{\mathrm{r}})$, $(\boldsymbol{x}'; \boldsymbol{w}') \in \mathrm{CNF}^P(\mathcal{B})$. ■

Theorem 6.4 indicates that the reduced BDD can separate more points than any other BDD representing the same solution set. We also note in passing that the variable ordering plays a role on the size of the BDD and, hence, on the effectiveness of the weaker combinatorial BDD flow cuts. Investigating variable orderings for specific problem classes and how they impact the cuts (theoretically and computationally) may lead to new research avenues.

6.4.4 Relationship with Existing BDD Cut Generation Procedures

The two existing BDD-based CGLPs rely on dual reformulations of NF($\mathcal{B}$), and, thus, also describe conv($\mathcal{X}$) (Tjandraatmadja and van Hoeve, 2019; Davarnia and van Hoeve, 2020). However, these tech-

niques rely on additional information: Tjandraatmadja and van Hoeve (2019) CGLP requires an interior point of $\text{conv}(\mathcal{X})$ and Davarnia and van Hoeve (2020) must incorporate possibly non-linear normalization constraints. In contrast, BDD-CGLP exploits the structure of $\mathcal{B}$ directly to describe $\text{conv}(\mathcal{X})$. We now detail these two BDD-based CGLPs and highlight the main theoretical differences to BDD-CGLP.

Consider $\boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|}$ as the dual variables associated with constraints (6.2a) and (6.2b), and $\boldsymbol{\theta} \in \mathbb{R}^n$ as the dual variables associated with (6.2c). The two BDD-based CGLP models in the literature employ flow inequalities of the form

$$\omega_{t(a)} - \omega_{s(a)} + \theta_i v_a \geq 0, \qquad \forall i \in I, a \in \mathcal{A}, s(a) \in \mathcal{N}_i. \tag{6.11}$$

Notice that (6.11) resembles the flow inequalities (6.5a)-(6.5c) of BDD-CGLP. However, our flow constraints use two sets of positive dual variables for each BDD layer (i.e., $\boldsymbol{\nu}, \boldsymbol{\eta} \in \mathbb{R}^n_+$) instead of the single unbounded set of variables $\boldsymbol{\theta} \in \mathbb{R}^n$. This difference emerges because (6.2c) only bounds the arc flow variables $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{A}|}_+$ with value $v_a = 1$, while our joint-capacity constraints (6.3a)-(6.3b) bound all variables $\boldsymbol{y}$. This is one reason, e.g., why BDD-CGLP does not require any normalization constraints as in previous techniques.

Formally, Tjandraatmadja and van Hoeve (2019) propose a BDD-based CGLP (6.12) to generate target cuts. Their CGLP derives a valid inequality that intersects the ray passing through an interior point $\boldsymbol{u} \in \text{conv}(\mathcal{X})$ and the fractional point $\boldsymbol{x}' \in [0,1]^n$ to be cut-off. The procedure returns a cut $\boldsymbol{\theta}^{*\top}\boldsymbol{x}' \leq 1 + \boldsymbol{\theta}^{*\top}\boldsymbol{u}$ whenever the optimal value of (6.12) is greater than one.

$$\max_{\boldsymbol{\omega},\boldsymbol{\theta}} \left\{ \boldsymbol{\theta}^\top(\boldsymbol{x}' - \boldsymbol{u}) : \ (6.11),\ \omega_{\mathbf{t}} = 0,\ \omega_{\mathbf{r}} = 1 + \boldsymbol{\theta}^\top \boldsymbol{u} \right\}. \tag{6.12}$$

Davarnia and van Hoeve (2020) circumvent the need of an interior point by proposing a simpler but possibly non-linear BDD-based CGLP presented in (6.13). The model checks if any $\boldsymbol{x}'$ can be represented as a linear combination of points in $\mathcal{X}$, i.e., whether there exists a $\boldsymbol{\theta}, \boldsymbol{\omega}$ such that $\boldsymbol{\theta}^\top \boldsymbol{x}' = \omega_{\mathbf{t}}$. Otherwise, their procedure returns a valid inequality $\boldsymbol{\theta}^{*\top}\boldsymbol{x} \leq \omega^*_{\mathbf{t}}$. Since the model may be unbounded, the optimization problem (6.13) includes normalization constraints $\mathcal{C}(\boldsymbol{\omega}, \boldsymbol{\theta}) \leq 0$ which are potentially non-linear. The CGLP (6.13) is addressed by an iterative subgradient separation algorithm.

$$\max_{\boldsymbol{\omega},\boldsymbol{\theta}} \left\{ \boldsymbol{\theta}^\top \boldsymbol{x}' - \omega_{\mathbf{t}} : \ (6.11),\ \omega_{\mathbf{r}} = 0,\ \mathcal{C}(\boldsymbol{\omega}, \boldsymbol{\theta}) \leq 0 \right\}. \tag{6.13}$$

Note that, in our approach, we either solve BDD-CGLP (a linear program) or a single max-flow/min-cut problem, both relying only on $\mathcal{B}$.

6.5 Case Study: Second-order Cone Inequalities

For our numerical evaluation, we apply our combinatorial cut-and-lift procedure to binary problems with SOC inequalities. Recall that problem *SP* considers a linear objective and m constraints of the form

$$\boldsymbol{a}^\top \boldsymbol{x} + ||D^\top \boldsymbol{x} - \boldsymbol{e}||_2 \leq b \quad \Leftrightarrow \quad \boldsymbol{a}^\top \boldsymbol{x} + \sqrt{\sum_{k \in \{1,\dots,l\}} (\boldsymbol{d}_k^\top \boldsymbol{x} - e_k)^2} \leq b, \tag{6.14}$$

where $\boldsymbol{a}, \boldsymbol{e} \in \mathbb{R}^n$ are real vectors, $D \in \mathbb{R}^{l \times n}$ is a matrix with l rows of dimension n, and $b \in \mathbb{R}$ is the right-hand-side constant.

We propose a novel BDD encoding for the general SOC inequalities (6.14) using a recursive reformulation (Bergman and Cire, 2018). Our formulation considers $l+1$ sets of state variables, $\boldsymbol{Q}_0, \boldsymbol{Q}_1, ..., \boldsymbol{Q}_l$, where each set of variables has $n+1$ stages, i.e., $\boldsymbol{Q}_k \in \mathbb{R}^{n+1}$ for each $k \in \{0, 1, ..., l\}$. State variables $\boldsymbol{Q}_0$ represent the value of the linear term (i.e., $\boldsymbol{a}^\top \boldsymbol{x}$), while $\boldsymbol{Q}_k$ encodes the k-th linear expression in the quadratic term (i.e., $\boldsymbol{d}_k^\top \boldsymbol{x} - e_k$). The recursive model for (6.14) is given by

$$
\begin{aligned}
\text{RSOC} := \big\{ (\boldsymbol{x}; \boldsymbol{Q}) \in \{0,1\}^n \times \mathbb{R}^{(l+1)\times(n+1)} : & \\
Q_{0,0} = 0,\ Q_{k,0} = e_k, & \qquad \forall k \in \{1, ..., l\}, && \text{(6.15a)} \\
Q_{0,i} = Q_{0,i-1} + a_i x_i, & \qquad \forall i \in I, && \text{(6.15b)} \\
Q_{k,i} = Q_{k,i-1} + d_{ki} x_i, & \qquad \forall i \in I,\ k \in \{1, ..., l\}, && \text{(6.15c)} \\
Q_{0,n} + \sqrt{\sum_{k \in \{1,...,l\}} (Q_{k,n})^2} \le b & \qquad \big\}. && \text{(6.15d)}
\end{aligned}
$$

The first set of equalities (6.15a) initialize the state variables at stage 0. Equalities (6.15b) and (6.15c) correspond to the recursive formulas for each linear expressions, and constraint (6.15d) enforces the SOC inequality. Notice that $\text{Proj}_x(RSOC)$ is equivalent to the feasible set of the SOC inequality (6.14).

This recursive model can be used for any type of SOC inequality. In particular, we employ RSOC to construct BDDs for chance constraints of the form $\mathbb{P}(\boldsymbol{\xi}^\top \boldsymbol{x} \le b) \ge \epsilon$ where $\boldsymbol{\xi}$ is a random variable with normal distribution $N(\boldsymbol{a}, D)$ and $\epsilon \in [0.5, 1]$. The constraint can be reformulated as SOC inequality

$$
\boldsymbol{a}^\top \boldsymbol{x} + \Phi^{-1}(\epsilon) ||D\boldsymbol{x}||_2 \le b \quad \Leftrightarrow \quad \boldsymbol{a}^\top \boldsymbol{x} + \Omega \sqrt{\sum_{k \in \{1,...,n\}} (\boldsymbol{d}_k^\top \boldsymbol{x})^2} \le b, \tag{6.16}
$$

where we use $\Omega = \Phi^{-1}(\epsilon)$ for simplicity. Notice that (6.16) is a special case of (6.14) where D is square matrix and $\boldsymbol{e} = \boldsymbol{0}$. For additional experiments with cutting-plane methods for SOC inequalities, we also present a BDD encoding for SOC knapsack inequalities (Atamtürk and Narayanan, 2009; Joung and Park, 2017), i.e., where $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix and the SOC constraint is given by

$$
\boldsymbol{a}^\top \boldsymbol{x} + \Omega \sqrt{\sum_{i \in I} d_{ii}^2 x_i} \le b. \tag{6.17}
$$

We develop a simpler recursive model for (6.17) with only two sets of state variables, $\boldsymbol{Q}_0$ and $\boldsymbol{Q}_1$. As before, $\boldsymbol{Q}_0$ represents the linear term and $\boldsymbol{Q}_1$ encodes the linear term inside the square root. Thus, the recursive model is given by

$$
\begin{aligned}
\text{RSOC-K} := \big\{ (\boldsymbol{x}; \boldsymbol{Q}) \in \{0,1\}^n \times \mathbb{R}^{2\times(n+1)} :\ & \text{(6.15a)}, \text{(6.15b)}, \\
Q_{1,i} = Q_{1,i-1} + d_{ii}^2 x_i, & \qquad \forall i \in I, \\
Q_{0,n} + \Omega \sqrt{\sum_{i \in I} Q_{1,n}} \le b & \qquad \big\}.
\end{aligned}
$$

In the following, we present our BDD encodings for these two SOC inequalities. Our implementation creates one (relaxed) BDD for each constraint, i.e., the encoding considers only one inequality. However,

the BDD construction procedure can be easily extended to consider multiple inequalities, for example, by enforcing each constraint one at a time over the BDD (Cire and van Hoeve, 2012).

6.5.1 BDD Encoding for General SOC Inequalities

We now introduce the BDD representation for the general SOC inequality (6.14) based on a recursive formulation of the constraint. Since the BDD encodes the set of solutions and does not consider the objective function, we limit our exposition to the elements needed to define the state-transition system.

Our recursive model has $n+1$ stages and n binary decision variables $\boldsymbol{x} \in \{0,1\}^n$. A state variable has $l+1$ values $\boldsymbol{S} = (Q_0, Q_1, ..., Q_l)$, where Q_0 represents the value of of the linear term (i.e., $\boldsymbol{a}^\top \boldsymbol{x}$), while $\boldsymbol{Q}_k$ encodes the k-th linear expression in the quadratic term (i.e., $\boldsymbol{d}_k^\top \boldsymbol{x} - e_k$) for $k \in \{1, ..., l\}$. Then, the initial state $\boldsymbol{S}_1$ is given by the constant term in each linear expression, i.e., $Q_0 = 0$ and $Q_k = -e_k$ for every $k \in \{1, ..., l\}$. The state transition function $\phi_i(\boldsymbol{S}, x)$ updates the values of each state given the value of the decision variable at stage i as

$$\phi_i(\boldsymbol{S}, x) = (Q_0 + a_i x,\ Q_1 + d_{1i} x,\ \dots\ ,\ Q_l + d_{li} x).$$

To enforce the SOC inequality, we restrict the last-stage feasibility set. Thus, the feasibility set for state $\boldsymbol{S}$ and stage $i = \{1, ..., n-1\}$ is $X_i(\boldsymbol{S}) = \{0, 1\}$, while the feasibility set for the n-stage is given by

$$X_n(\boldsymbol{S}) = \left\{ x \in \{0,1\} :\ Q_0 + a_n x + \Omega \sqrt{\sum_{k=1}^{l} (Q_k + d_{kn} x)^2} \leq b \right\}.$$

We consider the above transition system to create a BDD for SOC inequality (6.14). Since the size of an exact BDD can grow exponentially over the number of variables n, we consider a relaxed BDD instead. We choose the iterative refinement procedure to construct our relaxed BDD (see Algorithm 3 in Section 2.2), since the only constraint of the problem is enforced in the last stage. Thus, we need bottom-up information to prune the BDD during construction to obtain a tight discrete relaxation. We also reduce the resulting BDD since this is beneficial for our combinatorial cuts (see Section 6.4.3). We now describe the components of the construction procedure, i.e., relaxed states, filtering rules, and splitting.

Relaxed States and Filtering Rules. Our encoding considers relaxed states that over and under approximate each value Q_k of a state $\boldsymbol{S}$ as done for separable functions (see Section 2.2.2). For a given node $u \in \mathcal{N}$ and $k \in \{0, ..., l\}$, relaxed states $Q_k^{\min}(u)$ and $Q_k^{\max}(u)$ under and over approximate their respective linear term for all $\mathbf{r} - u$ paths in $\mathcal{B}$. These values are initialized at the root node, $Q_0^{\min}(\mathbf{r}) = Q_0^{\max}(\mathbf{r}) = 0$ and $Q_k^{\min}(\mathbf{r}) = Q_k^{\max}(\mathbf{r}) = -e_k$, and are updated in a top-down procedure for each node $u \in \mathcal{N}_i$ as

$$Q_k^{\min}(u) = \min_{a \in \mathcal{A}^{\text{in}}(u)} \{Q_k^{\min}(s(a)) + v_a f_{k,i-1}\}, \qquad Q_k^{\max}(u) = \max_{a \in \mathcal{A}^{\text{in}}(u)} \{Q_k^{\max}(s(a)) + v_a f_{k,i-1}\},$$

where $f_{0i} = a_i$ and $f_{ki} = d_{ki}$ for each $k \in \{1, ..., l\}$.

Similarly, we define bottom-up relaxed states for each Q_k value in a node. For each node $u \in \mathcal{N}$ and $k \in \{0, ..., l\}$, relaxed states $Q_k^{\uparrow \min}(u)$ and $Q_k^{\uparrow \max}(u)$ under and over approximate their respective linear

term considering all $u - \mathbf{t}$ paths in $\mathcal{B}$. The relaxed states have value equal to zero in the terminal node $Q_k^{\uparrow\min}(\mathbf{t}) = Q_k^{\uparrow\max}(\mathbf{t}) = 0$ and the values for any node $u \in \mathcal{N}_i$ is given by:

$$Q_k^{\uparrow\min}(u) = \min_{a \in \mathcal{A}^{\text{out}}(u)} \{Q_k^{\uparrow\min}(t(a)) + v_a f_{ki}\}, \qquad Q_k^{\uparrow\max}(u) = \max_{a \in \mathcal{A}^{\text{out}}(u)} \{Q_k^{\uparrow\max}(t(a)) + v_a f_{ki}\},$$

for all $k \in \{0, 1, ..., l\}$ and $i \in I$.

We utilized the above relaxed states to identify infeasible arcs. Proposition 6.4 presents our filtering rule and proves its validity. The main idea here is to under-approximate the left-hand-side (LHS) of the SOC inequality (6.14) to identify infeasible arcs.

Proposition 6.4. *An arc $a \in \mathcal{A}$ emanating from layer $\mathcal{N}_i$ can be removed from $\mathcal{B}$ if:*

$$Q_0^{\min}(s(a)) + a_i v_a + Q_0^{\uparrow\min}(t(a)) + \Omega\sqrt{\sum_{k=1}^{l} (g_k(a))^2} > b, \tag{SOC-R1}$$

where $g_k(a)$ for $k \in \{1, ..., l\}$ is given by:

$$g_k(a) = \begin{cases} Q_k^{\min}(s(a)) + d_{ki} v_a + Q_k^{\uparrow\min}(t(a)), & \text{if } \; Q_k^{\min}(s(a)) + d_{ki} v_a + Q_k^{\uparrow\min}(t(a)) > 0, \\ Q_k^{\max}(s(a)) + d_{ki} v_a + Q_k^{\uparrow\max}(t(a)), & \text{if } \; Q_k^{\max}(s(a)) + d_{ki} v_a + Q_k^{\uparrow\max}(t(a)) < 0, \\ 0, & \text{otherwise.} \end{cases}$$

Proof. Notice that $(g_k(a))^2$ is a lower bound for $(\boldsymbol{d}_k^\top \boldsymbol{x} - d_k)^2$ for all paths traversing arc $a \in \mathcal{A}$. The validity of this lower bound follows from the relaxed state definition and the quadratic function. Then, the LHS of SOC-R1 under approximates the LHS of (6.14) for all paths traversing arc a. Therefore, an arc a that satisfies SOC-R1 can be removed from $\mathcal{B}$ since all the path traversing a correspond to invalid assignments for (6.14). ■

Splitting Procedure. Our procedure focuses on each relaxed state component Q_k one at a time to make it exact, i.e., we split nodes so $Q_k^{\min}(u) = Q_k^{\max}(u)$ for each node $u \in \mathcal{N}$ and $k \in \{0, 1, ..., l\}$. We first sort indices in $\{0, 1, ..., l\}$ so as to split first the components Q_k that can potentially contribute more to violate the constraint, i.e., we sort the indices depending on the maximum value that their linear/quadratic term can have. Then, for a given $k \in \{0, 1, ..., l\}$, we split nodes in a layer to reduce difference $Q_k^{\max}(u) - Q_k^{\min}(u)$ given a maximum width $\mathcal{W}$.

Example 6.5 Consider the following binary set with an SOC constraints $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^3 : 3x_1 + x_2 + x_3 + \sqrt{(x_1 + x_2 + 2x_3)^2 + (x_1 + 3x_2 - x_3 + 3)^2} \leq 8\}$. Figure 6.2 depicts some of the steps to construct an exact BDD for X. The left most diagram corresponds to a width-one BDD for this problem. The top-down state at the root node is $((Q_0^{\min}(\mathbf{r}), Q_0^{\max}(\mathbf{r})), (Q_1^{\min}(\mathbf{r}), Q_1^{\max}(\mathbf{r})), (Q_2^{\min}(\mathbf{r}), Q_2^{\max}(\mathbf{r}))) = ((0,0), (0,0), (3,3))$, while for node u_1 is $((0,3), (0,1), (3,4))$.

The middle BDD illustrates the resulting BDD after splitting node u_1. The resulting nodes, u_1' and u_1'', have top-down state $((0,0), (0,0), (3,3))$ and $((3,3), (1,1), (4,4))$, respectively. In addition, the gray arc from u_1'' to u_2 corresponds to an invalid assignment: the bottom-up information of u_2 is $((0,1), (0,2), (-1,0))$, thus, SOC-R1 evaluates to $10.3 > 8$. □

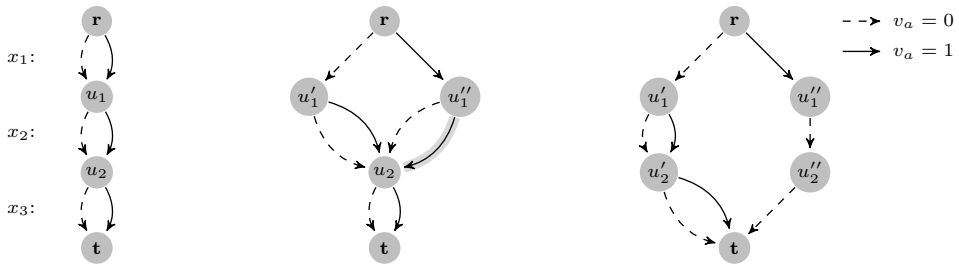


Figure 6.2: BDD construction procedure for set $\mathcal{X}$ defined in Example 6.5. The figure depicts a width-one BDD (left), a BDD after the splitting and filtering procedure over $\mathcal{N}_2$ (middle), and the resulting exact reduced BDD (right).

6.5.2 BDD Encoding for SOC Knapsack

The BDD encoding for the SOC knapsack constraint is a variant of the one presented for the general case since we need fewer state variables. As in the general case, the recursive model for (6.17) has $n+1$ stages and n binary decision variables $\boldsymbol{x} \in \{0,1\}^n$. However, in this case we have 2-dimensional states $\boldsymbol{S} = (Q_0, Q_1)$, where Q_0 represents the value of the linear term (i.e., $\boldsymbol{\mu}^\top \boldsymbol{x}$), while $\boldsymbol{Q}_1$ encodes the linear expression inside the square root. Then, the initial state is given by $\boldsymbol{S}_1 = (0,0)$. Similarly to the general SOC case, the state transition function $\phi_i(\boldsymbol{S}, x)$ updates the values of each state given the value of the decision variable at stage i as $\phi_i(\boldsymbol{S}, x) = (Q_0 + a_i x,\ Q_1 + d_{ii}^2 x)$. Once again, only the last-stage feasibility set enforces the knapsack SOC inequality, i.e., the feasibility set for stages $i = \{1, ..., n-1\}$ and state $\boldsymbol{S}$ is $X_i(\boldsymbol{S}) = \{0,1\}$ and for the last stage we have:

$$X_n(\boldsymbol{S}) = \left\{ x \in \{0,1\} :\ Q_0 + a_n x + \Omega\sqrt{Q_1 + d_{nn}^2 x} \leq b \right\}.$$

We consider the same BDD construction procedure as for the general SOC case, i.e., the iterative refinement procedure described in Algorithm 3 with an additional BDD reduction step before returning $\mathcal{B}$. The splitting procedure is analogous to the one for general SOC case. The relaxed states and filtering rules are described below.

Relaxed States and Filtering Rules. As for general SOC case, the relaxed states under and over approximate Q_0 and Q_1 inside the BDD. For each node $u \in \mathcal{N}$ and $k \in \{0,1\}$, relaxed states $Q_k^{\min}(u)$ and $Q_k^{\max}(u)$ under and over approximate their respective linear term for all $\mathbf{r} - u$ paths in $\mathcal{B}$. The relaxed states have value zero at the root node, i.e., $Q_k^{\min}(\mathbf{r}) = Q_k^{\max}(\mathbf{r}) = 0$ for $k \in \{0,1\}$. We update the relaxed state values for all other nodes in the BDD using a top-down procedure and the formulas for separable inequalities (see Section 2.2.2).

Similarly, we define bottom-up relaxed states for Q_0 and Q_1 in each BDD node. For each node $u \in \mathcal{N}$ and $k \in \{0,1\}$, relaxed states $Q_0^{\uparrow\min}(u)$ and $Q_1^{\uparrow\min}(u)$ under-approximate their respective linear term considering all $u - \mathbf{t}$ paths in $\mathcal{B}$. The relaxed states have value zero in the terminal state $Q_0^{\uparrow\min}(\mathbf{t}) = Q_1^{\uparrow\min}(\mathbf{t}) = 0$ and are updated using analogous formulas to the ones for the general SOC inequality. We avoid creating bottom-up relaxed states that over approximate Q_0 and Q_1 (i.e., $Q_0^{\uparrow\max}$ and $Q_1^{\uparrow\max}$) since they are not needed in our filtering rule.

We employ the above relaxed states to create the following filtering rule. An arc $a \in \mathcal{A}$ emanating

from layer $\mathcal{N}_i$ can be removed from $\mathcal{B}$ if:

$$Q_0^{\min}(s(a)) + a_i v_a + Q_0^{\uparrow\min}(t(a)) + \Omega\sqrt{Q_1^{\min}(s(a)) + d_{ii}^2 v_a + Q_1^{\uparrow\min}(t(a))} > b. \tag{KSOC-R1}$$

Notice that KSOC-R1 is simpler than the filtering rule for the general case since the coefficients d_{ii}^2 are always positive and there is no quadratic function. Thus, we only need the minimum value of each linear term to under estimate the LHS of the SOC knapsack inequality. The validity of KSOC-R1 follows from the relaxed states definitions and inequality (6.17).

6.6 Empirical Evaluation and Discussion

This section presents a numerical analysis of our combinatorial cut-and-lift procedure for *SP* (see Section 6.5). We create a BDD for each of the m SOC constraints and apply our procedure to each such constraint at the root node of the branch-and-bound tree. For any fractional point $\boldsymbol{x} \in [0,1]^n$, we iterate over each BDD until one of them generates either a general or combinatorial BDD flow cut, as we describe in detail below. We then lift the inequality using Algorithm 17. The procedure ends when $\boldsymbol{x}$ cannot be cut-off by any BDD.

We test our approach over the SOC knapsack (SOC-K) benchmark Atamtürk and Narayanan (2009); Joung and Park (2017) with $n \in \{100, 125, 150\}$ variables and $m \in \{10, 20\}$ constraints. We also generate a random set of instances (SOC-CC) for SOC inequalities coming from chance constraints (i.e., inequality (6.16)) following a similar procedure to the one used for SOC-K. We consider $n \in \{75, 100, 125\}$, $m \in \{10, 20\}$, $\Omega \in \{1, 3, 5\}$, and a density of $2/\sqrt{n}$ over all the constraints. Parameters $\boldsymbol{a}_j$, D_j, and $\boldsymbol{c}$ are sampled from a discrete uniform distribution with $\boldsymbol{a}_j \in [-50, 50]^n$, $D_j \in [-20, 20]^{n\times n}$, and $\boldsymbol{c} \in [0, 100]^n$. Parameters b_j are given by

$$b_j = t \cdot \left(\sum_{i\in I} a_{ji}^+ + \Omega \sqrt{\sum_{i\in I} \max\left\{ \sum_{k\in I} d_{jik}^+, \sum_{k\in I} d_{jik}^- \right\}^2} \right), \qquad \forall j \in \{1, ..., m\},$$

where $t \in \{0.1, 0.2, 0.3\}$ is the constraint tightness, $f^+ := \max\{0, f\}$, and $f^- := \max\{0, -f\}$ for any $f \in \mathbb{R}$. Note that b_j with $t = 0.3$ will remove approximately 50% of the possible assignments for $\boldsymbol{x} \in \{0, 1\}^n$. Then, we generate 5 random instances for each parameter combination of n, m, Ω, and t, i.e., a total of 270 instances.

We implement four variants of our approach to test the effectiveness of the BDD cuts and lifting procedure. The first two, BW and BWL, apply the weaker combinatorial BDD cutting-plane procedure (see Section 6.4.3), where BW omits the lifting procedure and BWL includes it. The other two variants, BG and BGL, use the combinatorial BDD flow cuts first and try the general BDD flow cuts (see Section 6.4.2) if the weaker approach fails to produce a cut. As before, BGL utilizes our lifting procedure in every generated constraint while BG does not.

We also implement the cover cuts and lifting procedure by Atamtürk and Narayanan (2009) for the SOC-K case. We test their cover cuts with and without their continuous lifting, C and CL, respectively. In addition, we experiment using their cover cuts in conjunction with our BDD lifting, BCL.

Our procedures are implemented in `C++` over the IBM ILOG CPLEX 12.9 solver using the `UserCuts` callback at the root node of the search. All experiments consider a single thread, a one-hour time limit,

and the linearization strategy (i.e., `MIQCPStrat = 2`) to solve the SOC problems.[1] We deactivate all solver cuts when running our techniques (i.e., BW, BWL, BG, and BGL) and the cover cut variants (i.e., C, CL, and BCL) to evaluate their effectiveness on their own. Notice that, given the `UserCuts` callback, our techniques omit the `Presolve` option and use `TraditionalSearch`. We use the same configuration when running CPLEX to have a fair comparison.[2]

We tested with three maximum BDD widths $\mathcal{W} = \{2000, 3000, 4000\}$, i.e., the maximum number of nodes per layer allowed in a relaxed BDD. We present results for the width with the best overall performance, $\mathcal{W} = 4000$ (we include detailed results for other widths in Appendix C.1). Notice that most of the created BDDs are relaxed due to the width limit, especially when $n \geq 100$.

6.6.1 Overall Performance and Comparison

We now present the empirical performance of our combinatorial cut-and-lift procedure. We show aggregated results for our techniques, CPLEX, and the cover cuts. See Appendix C.2 and C.3 for detailed results for each approach and data set.

Results for the SOC-CC instances. Table 6.1 presents the average results of all techniques over the 270 SOC-CC instances. The first column is the number of problems solved to optimality. The second and third columns correspond to the average root gap and final gap across all instances, i.e., $gap = (UB - LB)/LB$, with UB the root/final upper bound and LB the best lower bound found by all techniques. The fourth and fifth columns present the average solving time (including the BDD construction time) and nodes explored, respectively, for the subset of instances that all techniques solve. The sixth column shows the average number of cuts added by either the solver (i.e., for CPLEX) or our techniques. The last column presents the average percentage of cuts that are lifted at least one time.

Table 6.1: Aggregated results showing the overall performance of each technique for SOC-CC.

	# Solve	Root Gap	Final Gap	Time (sec)	# Nodes	# Cuts	% Lifted
CPLEX	137	20.8%	7.7%	550.4	176,860.3	128	-
BW	139	20.0%	7.3%	409.5	252,865.5	31	-
BWL	150	15.7%	5.9%	280.8	150,200.1	23	90.7%
BG	166	13.5%	4.6%	210.0	84,592.0	324	-
BGL	**168**	**13.4%**	**4.4%**	**169.7**	**78,058.3**	132	72.0%

Table 6.1 shows that all our variants have better performance than CPLEX. Specifically, BGL has the best performance solving 31 more instances than CPLEX. The difference on instances solved can be explained by the root node gap reduction for our approaches. Also, our lifting variants consistently solve more instances and are on average faster than BW and BG. The average run time for instances solved is significantly lower for our techniques. However, the number of nodes explored does not necessarily correlate with shorter solving time when looking at the disaggregated results (see Appendix C.3). We believe that this is influenced by the number of cuts added at the root node and the resulting root gap.

Figure 6.3 depicts the performance of each algorithm for the SOC-CC instances. The graph illustrates the number of instances solved over time (left side) and the accumulated number of instances over a final gap range (right side). We see a clear dominance of our general BDD flow cuts (i.e., BG and BGL)

[1] Preliminary experiments show that this was the best strategy across all techniques.
[2] Preliminary results show, in fact, that CPLEX performs better on the problem sets when `Presolve` is *deactivated*.

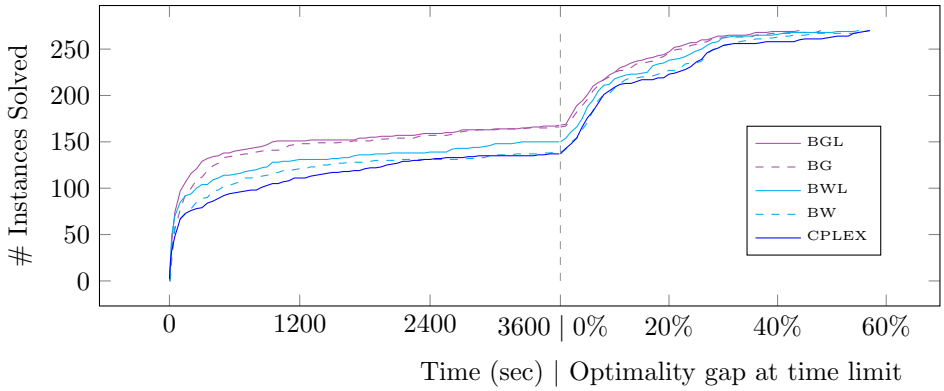


Figure 6.3: Profile plot comparing the accumulated number of instances solved over time (left) and the accumulated number of instances over a final gap range (right) for SOC-CC dataset.

and also the impact of lifting in the number of instances solved and the gap reduction. In particular, BG and BGL have the largest gap reductions and BWL has a consistently smaller gap than CPLEX. BW also improves upon CPLEX by a small margin.

While the results show that our techniques are on average superior to CPLEX, we uncovered problem characteristics where our best approach, BGL, performs significantly better. Figure 6.4 shows two plots comparing the root gap of BGL and CPLEX for the SOC-CC instances and different values of Ω and t. In each plot, an (x, y) point represents the root gap for an instance given by the x-axis and the y-axis technique, respectively. Overall, we can see that BGL achieves a smaller or equal root gap than CPLEX, however, the difference is considerably larger when $\Omega \geq 3$ and $t = 0.1$.

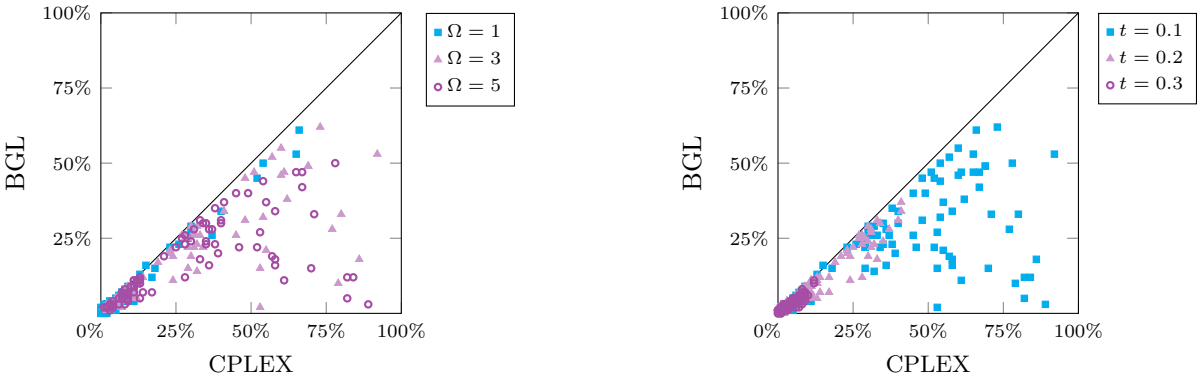


Figure 6.4: Root node gap comparison with CPLEX over the SOC-CC instances. The right plot illustrate the gap difference for different Ω values and the left plot for different t values.

The problems become more challenging with a larger Ω (i.e., the quadratic term is more predominant) because the SOC relaxation and its linearization are quite weak. Thus, our combinatorial cut-and-lift can potentially generate stronger cuts than CPLEX. In fact, the left plot in Figure 6.4 shows all instances with $\Omega = 1$ close to the diagonal, while problems with $\Omega \in \{3, 5\}$ have larger gap reductions. Lastly, the right plot of Figure 6.4 shows that BGL has significantly smaller gaps than CPLEX over instances with a smaller t (i.e., smaller solution sets per constraint). Our relaxed BDDs were closer to exact BDDs in these cases, thus, making our cuts more effective.

Results for the SOC-K instances. We now present the average performance over the 90 instances in the SOC-K data set. Table 6.2 shows a summary of the results where the columns have the same meaning as in Table 6.1. The table shows that the general BDD flow cuts (i.e., BG and BGL) solve up to 17 more instances than the best baselines. Moreover, BGL has a 52.5% gap reduction when compared to CL and an order of magnitude reduction in nodes explored. Also, our lifting procedure slightly outperforms the continuous SOC lifting (Atamtürk and Narayanan, 2009); BCL achieves a smaller average root and final gap than BCL, in addition to decreasing the run time and nodes explored. Lastly, BW has a significant performance improvement when enhanced by our lifting procedure, BWL, while there is a marginal improvement for lifting cover cuts (i.e., BCL and CL).

Table 6.2: Aggregated results showing the overall performance of each technique for SOC-K.

	# Solve	Root Gap	Final Gap	Time (sec)	# Nodes	# Cuts	% Lifted
CPLEX	70	2.84%	0.39%	174.7	161,210.1	123.5	-
C	71	3.29%	0.41%	233.4	489,853.8	96.4	-
CL	70	2.81%	0.34%	153.2	306,060.5	95.7	98.6%
BCL	70	2.67%	0.27%	**117.8**	199,002.4	81.0	70.0%
BW	64	3.63%	0.53%	500.1	938,585.4	240.6	-
BWL	74	2.80%	0.26%	161.6	310,699.6	54.9	99.2%
BG	76	**1.34%**	0.16%	468.7	48,056.1	1087.0	-
BGL	**88**	**1.33%**	**0.01%**	139.0	**35,661.5**	192.4	80.5%

To get a better understanding of this results, Figure 6.5 illustrates the performance profile of each algorithm for the SOC-K dataset. The plot shows that BGL has the best performance in terms of instances solved over time and final gap. We also observed a large jump on instances solved over time when the general and combinatorial cuts are enhanced with our lifting mechanism. For instance, BW has the weakest performance but BWL solves more instances than CPLEX and the cover cut alternatives.

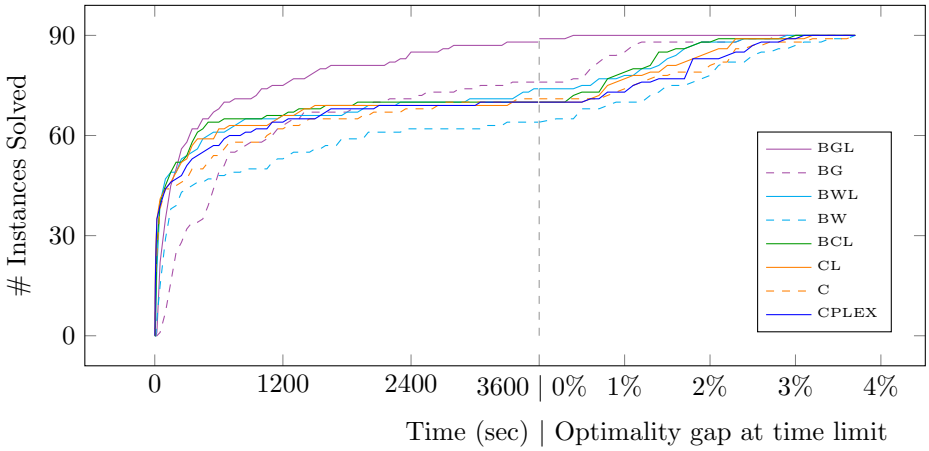


Figure 6.5: Profile plot comparing the accumulated number of instances solved over time (left), and the accumulated number of instances over a final gap range (right) for SOC-K dataset.

Similarly to the SOC-CC case, our best variant BGL performs significantly better than CL and CPLEX when the quadratic term of the SOC-K constraints is predominant (i.e., when Ω is larger than one). Figure 6.6 compares the root gap of BGL, CPLEX, and CL. From here we see that BGL has similar root gap to the existing techniques when $\Omega = 1$ and a considerably smaller gaps when $\Omega \geq 3$. The

SOC-K constraint is closer to a linear knapsack inequality when Ω is small, thus, CPLEX linearization technique is more effective approximating the convex hull in this case. Since our techniques rely on the combinatorial set of feasible solutions instead of the geometry of the constraint, the performance of our cuts is more stable across different values of Ω.

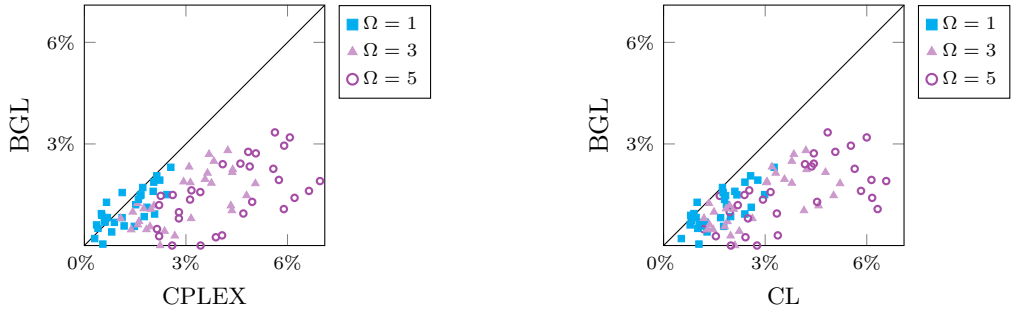


Figure 6.6: Root gap comparison between BGL, CPLEX, and CL for different values of Ω.

6.6.2 Effectiveness of BDD-based Cutting and Lifting Procedure

We now evaluate the effectiveness of our four variants by comparing their optimality gaps at the root node. We first focus on the results over the SOC-CC dataset. Figure 6.7 depicts three plots making a pairwise comparison of the root gap for all variants. The left plot compares BW and BG, where BG achieves equal or better gaps than BW since most points lie on or below the diagonal. Recall that the weaker combinatorial BDD flow cuts are incomplete, thus, it is expected that BG has smaller gaps than BW. The gap difference translates into 27 more instances solved by BG than BW (see Table 6.1).

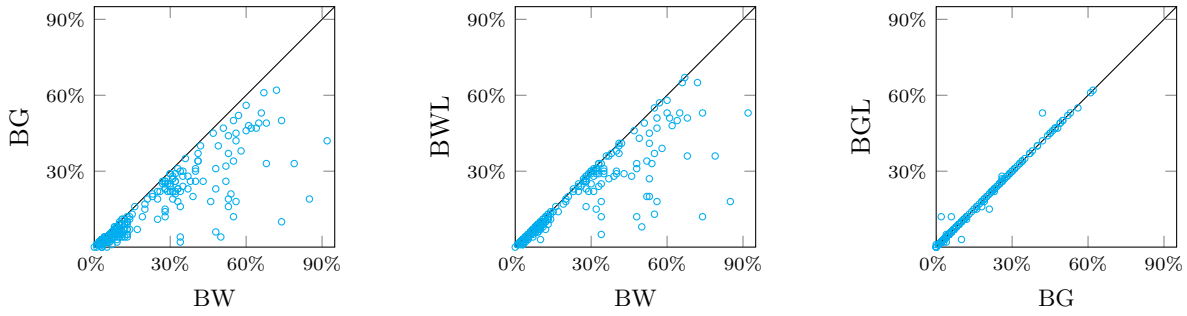


Figure 6.7: Pairwise root gap comparison for our four variants over the SOC-CC instances: BW vs. BG (left), BW vs. BWL (middle), and BG vs. BGL (right).

The last two plots in Figure 6.7 illustrate the effectiveness of the lifting procedure when using the combinatorial (middle) and the general (right) BDD flow cuts. We can see that BWL has smaller gaps than BW and its average root gap is closer BG than to BW (see Table 6.1). However, there is no clear root gap improvement when using the lifting procedure over the general BDD cuts. This is not a surprise since BG is able to separate all points outside each BDD solution set. Nonetheless, we can see the effectiveness of our lifting procedure later on, where BGL solves 2 instances more than BG, adds fewer cuts, and has the smallest average final gap across all variants (see Table 6.1).

We observed a similar behavior when we compare our four variant in the SOC-K data set, as shown in Figure 6.8. In this case, BG achieves a consistently smaller root gap than BW. Similarly, we can see that in most instances the lifting procedure has a positive impact over the combinatorial BDD flow cuts (middle plot). Lastly, there is no root gap difference when applying the lifting procedure over the general BDD flow cuts. However, there is a huge difference in overall performance when comparing BGL and BG, as shown in Table 6.2 and Figure 6.5. Since BGL adds on average five times fewer valid inequalities than BG, the former is more effective solving each problem.

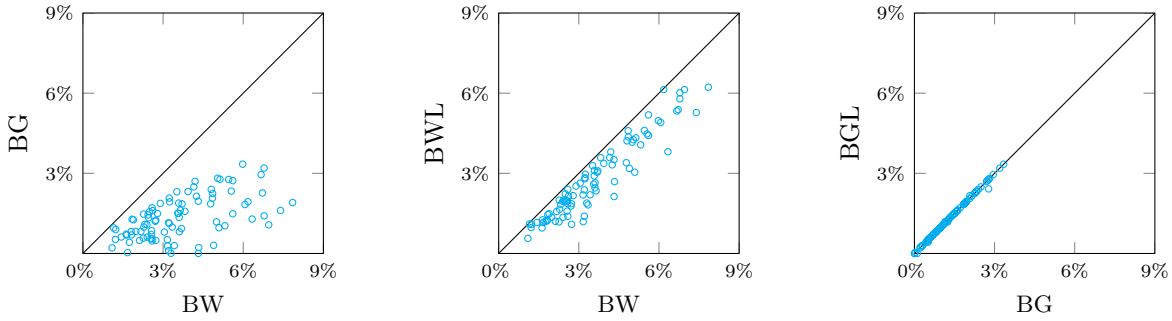


Figure 6.8: Root gap comparison between our techniques for SOC-K dataset.

Lastly, Figure 6.9 compares our combinatorial cut-and-lift to the cover cut alternatives. We can see that our combinatorial BDD flow cuts perform worse than the cover cuts (left plot), while the techniques are comparable when enhanced by their respective lifting procedures (middle plot). Our BDD lifting is slightly better than the continuous lifting over the cover inequalities (right plot). Moreover, the BDD lifting over for our stronger cuts (i.e., BGL) achieves tighter root relaxations when compare to CL (Figure 6.6, right plot).

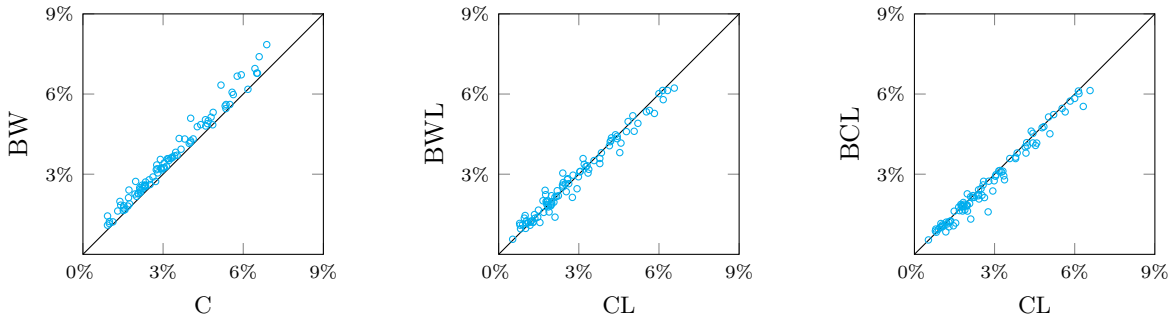


Figure 6.9: Root gap comparison: our techniques vs. existing approaches for SOC-K instances.

6.7 Conclusions

We introduce novel lifting and cutting-plane procedures for binary programs that leverage their combinatorial structure via a binary decision diagram (BDD) encoding of their constraints. Our lifting procedure relies on 0-1 disjunctions to lift valid inequalities and uses a BDD to efficiently compute the disjunctive sub-problems. This is an appealing property since the BDD needs to be constructed once

and can be used to lift any valid inequality by traversing all its arcs twice. In contrast, iterative lifting procedures based on continuous relaxation are often time consuming since they need to solve, in the best case, a linear programming model in each iteration.

While our combinatorial lifting can enhance any cutting-plane approach, we also propose a new cut generation algorithm based on an alternative network-flow representation of the BDD. Our cuts are derived from a max-flow variant over the BDD without the need of any additional information from the original problem. Moreover, we propose a more computational efficient alternative that generates cuts by solving a max-flow problem over the BDD directly, i.e., it does not required to construct a cut generator linear program. We show theoretical properties for both cutting-plane procedures—including the completeness of our stronger cuts—and compare our procedure to existing BDD-based cutting-planes in the literature.

BDDs give us the flexibility to apply our procedure to a wide range of non-linear problems. As a case study, we tested our procedure over second-order cone (SOC) inequalities. To do so, we introduce a novel BDD encoding for these constraints and compare the performance of our procedure against a state-of-the-art solver and existing cut-and-lift procedures for SOC knapsacks. The empirical results show that our technique solves 31 more instances, reducing the final gap by 42.6% and having a threefold decrease in run-time over the general chance-constrained instances. Also, our technique outperforms existing cut-and-lift methodologies for SOC knapsack problems by solving 17 more instances, achieving a 96.3% final gap reduction, and having comparable average solving time. We note that our procedure performs best when the solution set is small and the quadratic term of the SOC inequalities is predominant.

Chapter 7

Conclusions

This dissertation studies new methodologies to solve discrete optimization problems using decision diagrams (DDs). Our thesis states that DDs are effective tools to capture complex combinatorial structures that are not fully exploited by general-purpose solvers. Thus, combining DDs with existing technologies can achieve state-of-the-art performance on challenging optimization problems. To address this thesis, we study existing DD-based techniques and propose new procedures that integrate technologies from constraint programming (CP), artificial intelligence (AI), and integer programming (IP). We also consider optimization problems new to the DD community to further extend the applicability of DDs.

Chapter 4 focuses on the multi-commodity pickup-and-delivery traveling salesman problem (m-PDTSP), which generalizes several variants of the classical traveling salesman problem (TSP). We extend the DD literature on sequencing problems by introducing a novel encoding and compilation based on the capacity limitations of the problem. Our procedure leverages IP technologies by combining the DD relaxation with a linear programming formulation using Lagrangian duality. Also, our implementation considers inference and propagation techniques from the CP community to update the DD during search and infer infeasible assignments from the DD.

In Chapter 5, we introduce DD-based heuristics to the AI planning community by addressing the delete-free planning problems. We propose two DD relaxation techniques to compute admissible heuristics (i.e., dual bounds). Our DD encodings leverage different structural properties of the problem (i.e., sequencing and combinatorial structures) and, thus, provide informative heuristics for planning tasks where these properties are predominant. We enhance the DD relaxations using procedures from AI planning literature to identify landmarks and prune useless operators. We also relate our DD heuristics to admissible heuristics in the literature and present new research direction on how to combine DDs with AI planning techniques.

While the above chapters focus on a specific discrete optimization problem, Chapter 6 proposes new DD algorithms to solve general binary optimization problems. We present novel cut generation and lifting procedures based on DDs that can be integrated into any IP solver. Our combinatorial lifting is the first general-purpose DD-based algorithm to strengthen valid inequalities with strong theoretical guarantees. Specifically, we uncover conditions for when our iterative lifting returns facet-defining inequalities. This work also extends the literature on DD-based cutting plane procedures by presenting two novel cut generation algorithms that leverage a max-flow reformulation over the DD. While our techniques can be applied to any binary problem, we test them over second-order cone (SOC) inequalities. This work

introduces a DD encoding for SOC constraints and presents an extensive numerical study that shows how our techniques surpass state-of-the-art solvers.

7.1 Summary of Contributions

The main contributions of this dissertation include new algorithms and models based on DDs. We also contribute to different fields by solving challenging optimization problems with novel DD techniques. The following lists present a summary of the main contributions to the DD literature and the research areas of the problems addressed in this dissertation.

Contributions to DD Literature

1. We extend the use of DD Lagrangian bounds. Our implementation focuses on a pickup-and-delivery problem and studies the effects of penalizing different types of linear inequalities. We notice that it is beneficial to dualize constraints that are violated in the DD and have a strong linear programming formulation (e.g., assignment constraints).

2. We propose a general lifting procedure for binary optimization problems based on DDs. The algorithm is the first general lifting approach that leverages the combinatorial structure of a problem via DDs. Our technique relies on 0-1 disjunctions and relates to several lifting algorithms in the literature. We show that our approach returns a valid inequality and strictly increases the dimension of the induced face. Also, the lifted inequality removes a larger portion of the infeasible space than the original constraint. We provide sufficient conditions to obtain facet-defining inequalities when using our lifting mechanism iteratively.

3. We introduce two novel cut generation procedures based on a max-flow formulation over the DD. The first approach is based on a joint-capacity max-flow model that defines the convex hull of the feasibility set and, thus, can separate any point outside the convex hull. Our technique has several advantages with respect to existing DD-based cutting plane algorithms. For example, it does not require additional information from the feasibility set (e.g., interior points). Also, our cut generation linear program (CGLP) is bounded, thus, it avoids the need for normalization constraints.

   Our second cut generation technique relaxes the joint-capacity constraints of the first approach. By doing so, this procedure can efficiently generate valid inequalities solving a max-flow problem directly over the DD without the need to construct a CGLP. We show that this technique might not separate all infeasible fractional points and that reduced DD can remove more fractional points than unreduced DDs.

4. We present new DD models for a wide range of problems. Chapter 4 extends the DD literature for sequencing problems by considering capacity constraints and proposing a capacity-based construction procedure. In Chapter 5, we introduce the first DD encodings for delete-free AI planning and use a novel critical-path algorithm to obtain admissible heuristics. Lastly, Chapter 6 presents a DD encoding for second-order cone constraints, extending the literature of DDs for non-linear optimization. This chapter also presents a novel DD network flow formulation that can be integrated into any IP model.

Contributions to Other Areas

1. Our DD encoding for the m-PDTSP provides an efficient new alternative to model and solve pickup-and-delivery problems. We empirically show that DD relaxations can outperform state-of-the-art techniques, such as Benders decomposition and branch-and-cut. Moreover, our DD Lagrangian bounds significantly improve the performance of the DD relaxation, closing 27 instances in the literature. Thus, our DD Lagrangian approach is the current state-of-the-art for the m-PDTSP.

2. We introduce a new family of admissible heuristics to the AI planning community based on DDs. By relating our heuristics to other techniques in the AI planning literature, we provide new research directions that combine DDs with AI planning methodologies. This work also presents alternative uses of DDs for AI planning problems, such as extracting delete-free plans, identifying landmarks, and pruning redundant operators.

3. We propose a novel combinatorial cut-and-lift procedure for general binary optimization problems. The technique can enhance IP solvers by leveraging a DD to strengthen inequalities and separate infeasible points. We show that our approach achieves state-of-the-art performance for two types of second-order cone (SOC) inequalities: normally distributed linear chance constraints and SOC knapsacks.

7.2 Future Works

This section presents general research directions based on the procedures and modeling techniques introduced in this dissertation.

Extending DD-based Lagrangian Bounds

In Chapter 4, we empirically showed that Lagrangian penalties can significantly enhance DD bounds for a pickup-and-delivery routing problem. DD-based Lagrangian bounds have also been studied for other sequencing problems with similar results (Bergman et al., 2015b; Hooker, 2019). However, this technique is still quite new to the DD community and has not been considered for other types of problems.

A possible research direction is to apply this technique to other combinatorial problems. For example, one alternative is to use Lagrangian relaxation in conjunction with the DD encodings for delete-free planning that we present in Chapter 5. A recent work in the AI planning community proposes Lagrangian relaxation as a mechanism to compute admissible heuristics and relates the approach to existing heuristics in the field (Pommerening et al., 2019). This work opens the possibility of combining DDs with other AI planning techniques via Lagrangian duality to compute new admissible heuristics.

There is limited knowledge on how to effectively create Lagrangian relaxations that benefit from DD and integer linear programming (ILP) formulations. In Chapter 4, we empirically showed that relaxing tour constraints was beneficial, but penalties based on precedence and capacity constraints provided negligible improvements. Thus, an interesting research direction is to theoretically and empirically study which type of Lagrangian relaxations are better suited for DDs.

Another research direction is to consider Lagrangian decomposition instead of a Lagrangian relaxation to generate DD dual bounds. Bergman and Cire (2016a) propose the idea of DD-based Lagrangian decomposition in the CP community to propagate information between multiple DDs. However, this idea

has not been studied for bound computation in the DD literature. It is known that Lagrangian decomposition provides bounds that are stronger or equal to bounds from Lagrangian relaxations (Guignard and Kim, 1987). Thus, exploring this procedure can lead to tighter DD-based Lagrangian bounds.

Lastly, Lagrangian methodologies allow us to expand the applicability of DDs to mixed-integer programming (MIP) problems. As presented in this dissertation, DDs can model complex combinatorial structures, but they cannot represent continuous decisions. Lagrangian relaxation and other decomposition methodologies can extend the usage DDs by, for example, tackling MIP problems in which one or multiple DDs model the discrete variables and linear programming techniques handle the continuous variables.

Cutting Planes for Sequencing Problems

Current DD-based cut generation procedures, including the techniques introduced in Chapter 6, focus on general problem structures represented by either linear or non-linear constraints (Becker et al., 2005; Tjandraatmadja and van Hoeve, 2019; Davarnia and van Hoeve, 2020). However, none of these works have considered one of the most notable applications of DDs: sequencing problems.

Developing DD-based cut generation procedures for a sequencing problem is not a trivial task since the variables used in the DD encoding and in an ILP formulation are different. DD encodings of sequencing problems consider integer variables representing the order of the elements. In contrast, most ILP formulations for sequencing problems employ binary variables to represent the assignment of an element to a position in the sequence. Thus, the existing DD cutting plane procedures cannot be directly implemented for these problems due to the variable discrepancy.

An interesting research direction is to develop specialized cutting plane algorithms for sequencing problems. This idea could be particularly beneficial for problems where standard ILP formulations are known to provide poor relaxations, e.g., the sequential ordering problem (Ascheuer et al., 2000) and the pickup-and-delivery problem tackled in Chapter 4.

Modeling Complex Combinatorial Problems

Most DD applications have focused on sequencing problems or combinatorial problems that can be represented with linear constraints (Bergman et al., 2016a). While there exist DD encodings for non-linear expressions, e.g., sub-modular objective functions (Bergman and Cire, 2018), quadratic constraints (Bergman and Lozano, 2020), and SOC inequalities (see Chapter 6), the literature on these topics is very limited. Moreover, these works have shown that DDs can achieve state-of-the-art performance when tackling non-linear optimization problems. Thus, a potential research direction is to develop efficient DD encoding for these complex combinatorial structures.

An alternative is to create DD encodings for non-linear constraints using a mechanism similar to the one that we propose for SOC constraints (see Chapter 6). Our approach decomposes the non-linear expression into multiple linear components that are later on used to identify infeasible assignments over the DD. A similar strategy could be used for other type of constraints, for example, a polynomial inequality $\sum_{i=1}^{k}(\boldsymbol{a}_i^\top \boldsymbol{x} - b_i)^i \leq d$ with $k \in \mathbb{Z}_+$.

Another possible research project is to extend the use of DDs to model probabilistic constraints. This idea has been studied for specific types of chance constraints where the parameters follow Bernoulli (Latour et al., 2017, 2019) and Gaussian distributions (see Chapter 6). A possible extension is to model chance constraints where the parameters follow a probability distribution with finite support. Naive ILP

models for these structures enumerate all possible parameter combinations as constraints, resulting in formulations with exponentially main constraints (Ahmed and Shapiro, 2008). An interesting research question is whether DDs can efficiently model and solve this problem.

Appendix A

Pickup-and-Delivery: Additional Results

A detailed comparison of all five approaches is shown in Tables A.1 to A.8, where each table presents the average run time and number of instances solved for each class. Tables A.1-A.5 show one instance per line, for each of the instances of Class 1. Tables A.6-A.7 and A.8 present the average run time and instances solved for each configuration (10 instances per configuration). The average is computed considering the time limits and omitting the infeasible instances. In addition, Tables A.1-A.7 include column *inf* that shows the number of infeasible instances for each configuration. We point out that there are no infeasible instances in Class 3 (Table A.8).

Table A.1: Computational results - instances from Class 1

				Average time (# of instances solved)							
name	n	m	C	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	inf
br17.10Q10max1	16	10	10	-	-	0.6	0.6	**0.5**	1.0	1.0	
br17.10Q10max5	16	10	10	73.1	25.0	2167.3	**0.3**	**0.3**	0.5	0.6	
br17.10Q15max5	16	10	15	**0.0**	**0.0**	0.1	0.5	0.5	0.8	0.8	
br17.10Q20max5	16	10	20	1.2	-	**0.5**	**0.5**	**0.5**	0.9	1.0	
br17.10Q25max5	16	10	25	-	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.10Q2max1	16	10	2	**0.0**	-	9.4	0.2	0.2	0.3	0.3	1
br17.10Q30max5	16	10	30	-	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.10Q35max5	16	10	35	-	-	**0.5**	**0.5**	**0.5**	0.9	1.0	
br17.10Q3max1	16	10	3	27.8	7.0	7200.0 (0)	**0.3**	**0.3**	0.5	0.5	
br17.10Q40max5	16	10	40	-	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.10Q45max5	16	10	45	-	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.10Q4max1	16	10	4	6867.9	2284.0	**0.2**	0.3	0.3	0.6	0.6	
br17.10Q500max1	16	10	500	1.6	-	0.6	**0.5**	**0.5**	0.9	0.9	
br17.10Q50max5	16	10	50	-	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.10Q5max1	16	10	5	0.1	**0.0**	0.2	0.4	0.4	0.8	0.8	
br17.10Q6max1	16	10	6	2.5	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.10Q7max1	16	10	7	-	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.10Q8max1	16	10	8	-	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.10Q9max1	16	10	9	-	-	**0.5**	**0.5**	**0.5**	0.9	0.9	
br17.12Q10max1	16	12	10	2040.0	-	**0.2**	0.3	0.4	0.6	0.6	
br17.12Q10max5	16	12	10	7200.0 (0)	191.0	2270.0	**0.2**	**0.2**	0.5	0.5	
br17.12Q15max5	16	12	15	0.5	1.0	**0.3**	0.3	0.3	0.6	0.6	
br17.12Q20max5	16	12	20	0.9	-	**0.2**	0.3	0.3	0.6	0.7	
br17.12Q25max5	16	12	25	-	-	**0.2**	0.3	0.3	0.6	0.6	
br17.12Q2max1	16	12	2	**0.0**	-	7.6	0.2	0.2	0.3	0.3	1
br17.12Q30max5	16	12	30	-	-	**0.2**	0.3	0.4	0.6	0.6	
br17.12Q35max5	16	12	35	-	-	**0.2**	0.3	0.3	0.6	0.7	
br17.12Q3max1	16	12	3	1820.4	57.0	7200.0 (0)	**0.2**	**0.2**	0.5	0.5	
br17.12Q40max5	16	12	40	-	-	**0.2**	0.3	0.3	0.6	0.6	
br17.12Q45max5	16	12	45	-	-	**0.2**	0.3	0.3	0.6	0.7	
br17.12Q4max1	16	12	4	3048.6	888.0	2363.6	**0.3**	**0.3**	0.5	0.5	
br17.12Q500max1	16	12	500	1.1	-	**0.2**	0.3	0.3	0.6	0.6	
br17.12Q50max5	16	12	50	-	-	**0.2**	0.3	0.3	0.6	0.7	
br17.12Q5max1	16	12	5	0.1	**0.0**	0.2	0.3	0.3	0.6	0.6	
br17.12Q6max1	16	12	6	1.1	-	**0.2**	0.3	0.3	0.6	0.6	
br17.12Q7max1	16	12	7	-	-	**0.2**	0.3	0.3	0.6	0.6	
br17.12Q8max1	16	12	8	-	-	**0.2**	0.3	0.4	0.6	0.6	
br17.12Q9max1	16	12	9	-	-	**0.2**	0.3	0.3	0.6	0.6	
ESC07Q10max1	7	6	10	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q10max5	7	6	10	**0.0**	**0.0**	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q15max5	7	6	15	**0.0**	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q20max5	7	6	20	**0.0**	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q25max5	7	6	25	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q2max1	7	6	2	**0.0**	-	**0.0**	0.1	0.1	0.1	0.1	1
ESC07Q30max5	7	6	30	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q35max5	7	6	35	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q3max1	7	6	3	**0.0**	**0.0**	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q40max5	7	6	40	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q45max5	7	6	45	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q4max1	7	6	4	0.2	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q500max1	7	6	500	0.1	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q50max5	7	6	50	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q5max1	7	6	5	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q6max1	7	6	6	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q7max1	7	6	7	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q8max1	7	6	8	-	-	**0.0**	0.1	0.1	0.2	0.2	
ESC07Q9max1	7	6	9	-	-	**0.0**	0.1	0.1	0.2	0.2	

Table A.2: Computational results - instances from Class 1 (cont.)

				Average time (# of instances solved)							
name	n	m	C	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	inf
ESC11Q10max1	11	3	10	**0.0**	-	1.1	0.2	0.2	0.3	0.3	
ESC11Q10max5	11	3	10	-	-	1.1	**0.2**	**0.2**	0.3	0.4	
ESC11Q15max5	11	3	15	**0.0**	-	1.1	0.2	0.2	0.3	0.3	
ESC11Q20max5	11	3	20	**0.0**	-	1.1	0.2	0.2	0.3	0.3	
ESC11Q25max5	11	3	25	-	-	1.1	**0.2**	**0.2**	0.3	0.3	
ESC11Q2max1	11	3	2	**0.0**	-	1.1	0.2	0.2	0.3	0.3	
ESC11Q30max5	11	3	30	-	-	1.1	**0.2**	**0.2**	0.3	0.3	
ESC11Q35max5	11	3	35	-	-	1.2	**0.2**	**0.2**	0.3	0.3	
ESC11Q3max1	11	3	3	**0.0**	-	1.1	0.2	0.2	0.4	0.3	
ESC11Q40max5	11	3	40	-	-	1.1	**0.2**	**0.2**	0.3	0.3	
ESC11Q45max5	11	3	45	-	-	1.2	**0.2**	**0.2**	0.3	0.3	
ESC11Q4max1	11	3	4	**0.0**	-	1.2	0.2	0.2	0.3	0.3	
ESC11Q500max1	11	3	500	**0.0**	-	1.1	0.2	0.2	0.4	0.3	
ESC11Q50max5	11	3	50	-	-	1.2	**0.2**	**0.2**	0.3	0.3	
ESC11Q5max1	11	3	5	-	-	1.2	**0.2**	**0.2**	0.3	0.3	
ESC11Q6max1	11	3	6	-	-	1.1	**0.2**	**0.2**	0.3	0.3	
ESC11Q7max1	11	3	7	-	-	1.1	**0.2**	**0.2**	0.3	0.3	
ESC11Q8max1	11	3	8	-	-	1.2	**0.2**	**0.2**	0.3	0.3	
ESC11Q9max1	11	3	9	-	-	1.1	**0.2**	**0.2**	0.3	0.3	
ESC12Q10max1	12	7	10	-	-	1.5	**0.2**	**0.2**	0.4	0.4	
ESC12Q10max5	12	7	10	**0.0**	-	1.5	0.1	0.2	0.2	0.2	1
ESC12Q15max5	12	7	15	**0.1**	1.0	7200.0 (0)	0.2	0.2	0.4	0.4	
ESC12Q20max5	12	7	20	**0.0**	-	1.5	0.2	0.2	0.4	0.4	
ESC12Q25max5	12	7	25	**0.0**	-	1.6	0.2	0.2	0.4	0.4	
ESC12Q2max1	12	7	2	-	-	**0.0**	0.1	0.1	0.2	0.2	1
ESC12Q30max5	12	7	30	-	-	1.5	**0.2**	0.3	0.4	0.4	
ESC12Q35max5	12	7	35	-	-	1.5	**0.2**	**0.2**	0.4	0.4	
ESC12Q3max1	12	7	3	**0.0**	-	1.4	0.1	0.1	0.2	0.2	1
ESC12Q40max5	12	7	40	-	-	1.6	**0.2**	**0.2**	0.4	0.4	
ESC12Q45max5	12	7	45	-	-	1.6	**0.2**	**0.2**	0.4	0.4	
ESC12Q4max1	12	7	4	0.3	**0.0**	7200.0 (0)	0.2	0.2	0.3	0.3	
ESC12Q500max1	12	7	500	**0.0**	-	1.6	0.2	0.2	0.4	0.4	
ESC12Q50max5	12	7	50	-	-	1.6	**0.2**	**0.2**	0.4	0.4	
ESC12Q5max1	12	7	5	**0.1**	1.0	7200.0 (0)	0.2	0.2	0.3	0.3	
ESC12Q6max1	12	7	6	**0.0**	-	1.6	0.3	0.2	0.4	0.4	
ESC12Q7max1	12	7	7	**0.0**	-	1.6	0.2	0.2	0.4	0.4	
ESC12Q8max1	12	7	8	-	-	1.6	**0.2**	**0.2**	0.4	0.4	
ESC12Q9max1	12	7	9	-	-	1.6	**0.2**	**0.2**	0.4	0.4	
ESC25Q10max1	25	9	10	-	-	227.3	21.3	**11.9**	12.3	13.0	
ESC25Q15max5	25	9	15	**0.1**	1.0	7200.3 (0)	28.3	8.1	9.6	9.5	
ESC25Q20max5	25	9	20	**0.1**	1.0	216.4	19.9	9.3	9.9	9.7	
ESC25Q25max5	25	9	25	-	-	215.7	22.7	**9.4**	10.6	9.7	
ESC25Q30max5	25	9	30	-	-	214.1	20.3	**9.4**	10.0	9.8	
ESC25Q35max5	25	9	35	-	-	214.3	20.3	**9.3**	9.9	9.8	
ESC25Q3max1	25	9	3	43.2	13.0	7200.3 (0)	17.8	757.5	**9.5**	12.0	
ESC25Q40max5	25	9	40	-	-	216.1	25.7	**10.3**	11.0	10.5	
ESC25Q45max5	25	9	45	-	-	215.2	22.3	**9.1**	10.0	9.5	
ESC25Q4max1	25	9	4	**4.5**	7.0	7200.3 (0)	19.6	13.0	10.1	9.9	
ESC25Q500max1	25	9	500	**0.1**	-	217.2	15.0	9.3	9.3	9.7	
ESC25Q50max5	25	9	50	-	-	214.5	20.2	**9.3**	9.9	9.7	
ESC25Q5max1	25	9	5	**0.1**	1.0	193.1	14.2	8.5	9.3	9.7	
ESC25Q6max1	25	9	6	**0.1**	-	394.1	15.3	9.2	9.3	9.7	
ESC25Q7max1	25	9	7	-	-	215.7	15.1	**9.2**	9.2	9.7	
ESC25Q8max1	25	9	8	-	-	216.5	16.1	**8.9**	9.3	9.7	
ESC25Q9max1	25	9	9	-	-	217.2	14.9	**9.3**	9.3	9.7	

Table A.3: Computational results - instances from Class 1 (cont.)

				Average time (# of instances solved)							
name	n	m	C	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	inf
ESC47Q10max1	47	10	10	**10.0**	-	7200.1 (0)	7212.8 (0)	7211.7 (0)	124.8	148.4	
ESC47Q10max5	47	10	10	**61.3**	83.0	7200.1 (0)	7211.7 (0)	7210.2 (0)	155.5	151.7	
ESC47Q15max5	47	10	15	**10.0**	12.0	7200.1 (0)	7212.8 (0)	7211.3 (0)	161.0	161.0	
ESC47Q20max5	47	10	20	**9.9**	17.0	7200.1 (0)	7212.7 (0)	7211.7 (0)	160.8	162.8	
ESC47Q25max5	47	10	25	**9.9**	-	7200.1 (0)	7212.5 (0)	7208.9 (0)	160.4	163.6	
ESC47Q2max1	47	10	2	-	-	7200.0 (0)	**3.3**	4.0	5.9	7.8	1
ESC47Q30max5	47	10	30	**9.9**	-	7200.1 (0)	7212.4 (0)	7212.3 (0)	143.0	149.7	
ESC47Q35max5	47	10	35	-	-	7200.1 (0)	7212.6 (0)	7213.0 (0)	162.3	163.6	
ESC47Q3max1	47	10	3	61.0	**40.0**	7200.1 (0)	7211.6 (0)	7210.0 (0)	155.5	152.6	
ESC47Q40max5	47	10	40	**10.0**	-	7200.1 (0)	7212.9 (0)	7211.6 (0)	158.9	163.4	
ESC47Q45max5	47	10	45	-	-	7200.1 (0)	7212.8 (0)	7212.0 (0)	158.1	162.8	
ESC47Q4max1	47	10	4	**11.5**	12.0	7200.1 (0)	7212.5 (0)	7212.2 (0)	163.8	165.9	
ESC47Q500max1	47	10	500	**10.0**	-	7200.1 (0)	7212.4 (0)	7212.4 (0)	163.1	167.4	
ESC47Q50max5	47	10	50	**10.0**	-	7200.1 (0)	7212.6 (0)	7213.2 (0)	156.7	164.6	
ESC47Q5max1	47	10	5	**10.0**	-	7200.1 (0)	7212.8 (0)	7212.2 (0)	165.0	166.5	
ESC47Q6max1	47	10	6	**9.9**	-	7200.1 (0)	7212.9 (0)	7212.8 (0)	162.9	167.7	
ESC47Q7max1	47	10	7	**10.0**	-	7200.1 (0)	7212.9 (0)	7213.2 (0)	162.5	168.8	
ESC47Q8max1	47	10	8	**10.0**	-	7200.1 (0)	7210.7 (0)	7211.9 (0)	164.6	167.8	
ESC47Q9max1	47	10	9	**10.0**	-	7200.1 (0)	7210.8 (0)	7210.6 (0)	162.5	167.7	
p43.1Q1000max1	42	9	1000	**1.9**	-	7200.0 (0)	7207.6 (0)	7208.7 (0)	7219.9 (0)	121.0	
p43.1Q1000max5	42	9	1000	-	-	7200.0 (0)	7209.5 (0)	7208.6 (0)	7225.2 (0)	120.6	
p43.1Q100max5	42	9	100	**1.9**	-	7200.0 (0)	7209.6 (0)	7208.7 (0)	7225.2 (0)	119.8	
p43.1Q10max1	42	9	10	**1.9**	-	7200.0 (0)	7209.7 (0)	7208.9 (0)	626.5	120.2	
p43.1Q10max5	42	9	10	7200.0 (0)	7200.0 (0)	7200.0 (0)	7209.7 (0)	7208.7 (0)	292.6	**287.9**	
p43.1Q12max1	42	9	12	-	-	7200.0 (0)	7209.3 (0)	7208.4 (0)	2305.7	**120.0**	
p43.1Q13max1	42	9	13	-	-	7200.0 (0)	7209.5 (0)	7208.7 (0)	3630.0	**119.4**	
p43.1Q14max1	42	9	14	-	-	7200.0 (0)	7209.4 (0)	7208.6 (0)	5229.2	**120.2**	
p43.1Q15max1	42	9	15	-	-	7200.0 (0)	7209.5 (0)	7208.5 (0)	7225.6 (0)	**120.2**	
p43.1Q15max5	42	9	15	7200.0 (0)	7200.0 (0)	7200.0 (0)	7209.9 (0)	7208.7 (0)	7225.5 (0)	**129.3**	
p43.1Q16max1	42	9	16	-	-	7200.0 (0)	7209.6 (0)	7208.7 (0)	7225.3 (0)	**119.9**	
p43.1Q17max1	42	9	17	-	-	7200.0 (0)	7210.0 (0)	7208.7 (0)	7225.0 (0)	**120.4**	
p43.1Q18max1	42	9	18	-	-	7200.0 (0)	7209.7 (0)	7208.6 (0)	7225.0 (0)	**118.9**	
p43.1Q19max1	42	9	19	-	-	7200.0 (0)	7210.0 (0)	7208.8 (0)	7225.0 (0)	**119.8**	
p43.1Q20max1	42	9	20	**1.9**	-	7200.0 (0)	7209.5 (0)	7208.8 (0)	7224.6 (0)	121.1	
p43.1Q20max5	42	9	20	-	-	7200.0 (0)	7209.6 (0)	7208.8 (0)	7225.8 (0)	**120.0**	
p43.1Q25max5	42	9	25	-	-	7200.0 (0)	7209.8 (0)	7208.5 (0)	7225.0 (0)	**120.5**	
p43.1Q2max1	42	9	2	7200.0 (0)	7200.0 (0)	7200.0 (0)	7200.3 (0)	7209.8 (0)	7225.2 (0)	7227.9 (0)	
p43.1Q30max1	42	9	30	**2.0**	-	7200.0 (0)	7209.5 (0)	7208.5 (0)	7224.9 (0)	128.8	
p43.1Q30max5	42	9	30	-	-	7200.0 (0)	7209.6 (0)	7208.7 (0)	7223.1 (0)	**119.8**	
p43.1Q35max5	42	9	35	-	-	7200.0 (0)	7210.1 (0)	7209.0 (0)	7225.6 (0)	**121.1**	
p43.1Q3max1	42	9	3	7200.0 (0)	7200.0 (0)	7200.0 (0)	7209.7 (0)	7208.6 (0)	**450.8**	909.7	
p43.1Q40max5	42	9	40	**1.9**	-	7200.0 (0)	7209.5 (0)	7208.4 (0)	7227.2 (0)	121.0	
p43.1Q45max5	42	9	45	-	-	7200.0 (0)	7209.7 (0)	7208.6 (0)	7225.7 (0)	**120.1**	
p43.1Q4max1	42	9	4	7200.0 (0)	7200.0 (0)	7200.0 (0)	7209.4 (0)	7209.0 (0)	508.6	**225.1**	
p43.1Q500max1	42	9	500	**1.9**	-	7200.0 (0)	7210.1 (0)	7208.3 (0)	7225.1 (0)	120.6	
p43.1Q50max5	42	9	50	-	-	7200.0 (0)	7209.8 (0)	7208.6 (0)	7226.1 (0)	**120.4**	
p43.1Q55max5	42	9	55	-	-	7200.0 (0)	7209.6 (0)	7208.5 (0)	7224.9 (0)	**120.7**	
p43.1Q5max1	42	9	5	-	-	7200.0 (0)	7209.5 (0)	7208.8 (0)	2354.5	**130.4**	
p43.1Q60max5	42	9	60	**1.9**	-	7200.0 (0)	7209.5 (0)	7208.6 (0)	7224.5 (0)	121.0	
p43.1Q65max5	42	9	65	-	-	7200.0 (0)	7209.6 (0)	7208.6 (0)	7224.6 (0)	**120.9**	
p43.1Q6max1	42	9	6	-	-	7200.0 (0)	7210.0 (0)	7208.9 (0)	4356.5	**122.6**	
p43.1Q70max5	42	9	70	-	-	7200.0 (0)	7209.6 (0)	7208.2 (0)	7225.5 (0)	**120.5**	
p43.1Q75max5	42	9	75	-	-	7200.0 (0)	7209.9 (0)	7208.7 (0)	7225.4 (0)	**121.1**	
p43.1Q7max1	42	9	7	-	-	7200.0 (0)	7209.6 (0)	7209.0 (0)	241.6	**120.5**	
p43.1Q80max5	42	9	80	**1.9**	-	7200.0 (0)	7209.9 (0)	7208.2 (0)	7225.6 (0)	120.5	
p43.1Q8max1	42	9	8	-	-	7200.0 (0)	7209.6 (0)	7208.8 (0)	246.9	**121.3**	
p43.1Q90max5	42	9	90	-	-	7200.0 (0)	7209.6 (0)	7208.9 (0)	7225.2 (0)	**120.3**	
p43.1Q9max1	42	9	9	-	-	7200.0 (0)	7209.8 (0)	7208.8 (0)	570.8	**120.2**	

Table A.4: Computational results - instances from Class 1 (cont.)

				Average time (# of instances solved)							
name	n	m	C	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	inf
p43.2Q1000max1	42	20	1000	7200.4 (0)	-	7200.0 (0)	7207.7 (0)	7206.7 (0)	7221.6 (0)	**1086.8**	
p43.2Q100max5	42	20	100	7201.6 (0)	-	7200.0 (0)	7207.9 (0)	7207.6 (0)	7223.0 (0)	**1030.6**	
p43.2Q10max1	42	20	10	7201.9 (0)	7200.0 (0)	7200.0 (0)	7207.7 (0)	7206.6 (0)	7222.8 (0)	**1072.5**	
p43.2Q10max5	42	20	10	-	-	7200.0 (0)	7206.8 (0)	7207.5 (0)	7220.9 (0)	7221.4 (0)	
p43.2Q15max5	42	20	15	-	-	7200.0 (0)	7207.8 (0)	7208.0 (0)	7223.6 (0)	7221.9 (0)	
p43.2Q17max1	42	20	17	-	-	7200.0 (0)	7207.5 (0)	7207.3 (0)	7221.6 (0)	**1077.2**	
p43.2Q18max1	42	20	18	-	-	7200.0 (0)	7207.4 (0)	7207.7 (0)	7221.7 (0)	**1037.6**	
p43.2Q19max1	42	20	19	-	-	7200.0 (0)	7207.7 (0)	7206.6 (0)	7220.4 (0)	**1034.5**	
p43.2Q20max1	42	20	20	7201.5 (0)	-	7200.0 (0)	7207.7 (0)	7206.8 (0)	7220.3 (0)	**1035.7**	
p43.2Q20max5	42	20	20	-	-	7200.0 (0)	7207.4 (0)	7207.0 (0)	7223.3 (0)	**1028.1**	
p43.2Q25max5	42	20	25	-	-	7200.0 (0)	7207.5 (0)	7207.1 (0)	7223.0 (0)	**1034.0**	
p43.2Q2max1	42	20	2	-	-	7200.0 (0)	7200.1 (0)	7200.1 (0)	7211.9 (0)	7214.4 (0)	
p43.2Q30max1	42	20	30	7201.6 (0)	-	7200.0 (0)	7207.7 (0)	7206.8 (0)	7221.4 (0)	**1105.5**	
p43.2Q30max5	42	20	30	-	-	7200.0 (0)	7207.5 (0)	7206.6 (0)	7223.3 (0)	**1036.4**	
p43.2Q35max5	42	20	35	-	-	7200.0 (0)	7207.4 (0)	7207.1 (0)	7225.5 (0)	**1034.0**	
p43.2Q3max1	42	20	3	-	-	7200.0 (0)	7207.5 (0)	7207.7 (0)	7220.4 (0)	7221.9 (0)	
p43.2Q40max5	42	20	40	7200.7 (0)	7200.0 (0)	7200.0 (0)	7207.6 (0)	7207.0 (0)	7223.9 (0)	**1099.4**	
p43.2Q45max5	42	20	45	-	-	7200.0 (0)	7207.9 (0)	7206.7 (0)	7223.2 (0)	**1031.5**	
p43.2Q4max1	42	20	4	-	-	7200.0 (0)	7207.6 (0)	7207.7 (0)	7221.9 (0)	7222.2 (0)	
p43.2Q500max1	42	20	500	-	-	7200.0 (0)	7207.8 (0)	7207.3 (0)	7221.6 (0)	**1104.6**	
p43.2Q50max5	42	20	50	-	-	7200.0 (0)	7207.5 (0)	7207.2 (0)	7224.6 (0)	**1032.2**	
p43.2Q5max1	42	20	5	-	-	7200.0 (0)	7207.0 (0)	7207.6 (0)	7221.7 (0)	7222.3 (0)	
p43.2Q60max5	42	20	60	7201.6 (0)	-	7200.0 (0)	7207.5 (0)	7206.8 (0)	7223.9 (0)	**1094.8**	
p43.2Q6max1	42	20	6	-	-	7200.0 (0)	7206.8 (0)	7208.6 (0)	7222.2 (0)	7222.2 (0)	
p43.2Q7max1	42	20	7	-	-	7200.0 (0)	7207.6 (0)	7208.4 (0)	7223.3 (0)	7222.0 (0)	
p43.2Q80max5	42	20	80	7201.7 (0)	-	7200.0 (0)	7207.7 (0)	7206.9 (0)	7221.9 (0)	**1029.0**	
p43.2Q8max1	42	20	8	-	-	7200.0 (0)	7207.5 (0)	7206.7 (0)	7222.7 (0)	**1531.8**	
p43.2Q9max1	42	20	9	-	-	7200.0 (0)	7207.6 (0)	7207.5 (0)	7222.2 (0)	**1036.7**	
p43.3Q1000max1	42	37	1000	7200.3 (0)	-	142.0	7205.4 (0)	7205.7 (0)	7222.4 (0)	**65.2**	
p43.3Q100max5	42	37	100	7200.5 (0)	-	1483.3	7205.0 (0)	7204.2 (0)	7217.5 (0)	**65.1**	
p43.3Q10max1	42	37	10	7200.5 (0)	7200.0 (0)	7200.0 (0)	7206.0 (0)	7204.3 (0)	7216.0 (0)	**2713.4**	
p43.3Q10max5	42	37	10	-	-	7200.0 (0)	7201.1 (0)	7200.3 (0)	7210.2 (0)	7208.9 (0)	
p43.3Q15max5	42	37	15	-	-	7200.0 (0)	7206.6 (0)	7205.0 (0)	7214.6 (0)	7214.4 (0)	
p43.3Q20max1	42	37	20	7201.3 (0)	-	140.7	7205.2 (0)	7204.3 (0)	7217.7 (0)	**65.3**	
p43.3Q20max5	42	37	20	-	-	7200.0 (0)	7205.2 (0)	7205.0 (0)	7215.6 (0)	7214.9 (0)	
p43.3Q25max5	42	37	25	-	-	7200.0 (0)	7205.1 (0)	7206.2 (0)	7215.8 (0)	**2981.6**	
p43.3Q2max1	42	37	2	-	-	7200.0 (0)	**1.0**	**1.0**	1.6	1.6	1
p43.3Q30max1	42	37	30	7201.1 (0)	-	137.9	7205.2 (0)	7204.8 (0)	7215.8 (0)	**59.2**	
p43.3Q30max5	42	37	30	-	-	**552.6**	7205.0 (0)	7205.8 (0)	7217.7 (0)	853.1	
p43.3Q35max5	42	37	35	-	-	489.0	7205.0 (0)	7204.4 (0)	7218.6 (0)	**120.3**	
p43.3Q3max1	42	37	3	-	-	7200.0 (0)	**1.9**	2.1	3.5	4.0	1
p43.3Q40max5	42	37	40	7200.5 (0)	7200.0 (0)	**60.1**	7205.0 (0)	7205.3 (0)	7217.2 (0)	60.8	
p43.3Q45max5	42	37	45	-	-	215.1	7205.0 (0)	7204.4 (0)	7219.1 (0)	65.1	
p43.3Q4max1	42	37	4	-	-	7200.0 (0)	7200.6 (0)	7200.2 (0)	7208.9 (0)	7208.9 (0)	
p43.3Q500max1	42	37	500	-	-	142.3	7205.2 (0)	7205.8 (0)	7222.6 (0)	64.9	
p43.3Q50max5	42	37	50	-	-	84.8	7205.4 (0)	7204.9 (0)	7218.1 (0)	64.4	
p43.3Q5max1	42	37	5	-	-	7200.0 (0)	7200.5 (0)	7200.6 (0)	7209.9 (0)	7209.1 (0)	
p43.3Q60max5	42	37	60	7200.9 (0)	-	1479.6	7205.5 (0)	7204.9 (0)	7218.9 (0)	65.1	
p43.3Q6max1	42	37	6	-	-	7200.0 (0)	7205.1 (0)	7200.1 (0)	7214.2 (0)	7214.8 (0)	
p43.3Q7max1	42	37	7	-	-	7200.0 (0)	7204.8 (0)	7205.9 (0)	7214.7 (0)	7214.2 (0)	
p43.3Q80max5	42	37	80	7200.8 (0)	-	1410.0	7205.0 (0)	7204.7 (0)	7217.4 (0)	63.2	
p43.3Q8max1	42	37	8	-	-	7200.0 (0)	7204.8 (0)	7205.1 (0)	7215.8 (0)	7215.2 (0)	
p43.3Q9max1	42	37	9	-	-	7200.0 (0)	7205.0 (0)	7204.9 (0)	7215.6 (0)	7215.6 (0)	

Table A.5: Computational results - instances from Class 1 (cont.)

				Average time (# of instances solved)							
name	n	m	C	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	inf
p43.4Q1000max1	42	50	1000	7200.1 (0)	-	7200.0 (0)	7.6	6.5	11.0	9.1	
p43.4Q100max5	42	50	100	7200.2 (0)	-	7200.0 (0)	11.1	6.0	13.1	8.5	
p43.4Q10max1	42	50	10	7200.9 (0)	7200.0 (0)	7200.0 (0)	2.9	2.4	4.7	4.1	
p43.4Q10max5	42	50	10	-	-	7200.0 (0)	0.8	0.8	1.3	1.3	1
p43.4Q15max5	42	50	15	-	-	7200.0 (0)	0.8	0.8	1.4	1.4	1
p43.4Q20max1	42	50	20	7200.1 (0)	-	7200.0 (0)	5.5	5.4	8.9	7.7	
p43.4Q20max5	42	50	20	-	-	7200.0 (0)	0.9	0.9	1.6	1.6	1
p43.4Q25max5	42	50	25	-	-	7200.0 (0)	4.8	3.2	7.9	5.4	
p43.4Q2max1	42	50	2	-	-	7200.0 (0)	0.8	0.8	1.3	1.3	1
p43.4Q30max1	42	50	30	7200.2 (0)	-	7200.0 (0)	5.6	5.4	9.0	7.8	
p43.4Q30max5	42	50	30	-	-	7200.0 (0)	6.8	4.9	9.9	7.2	
p43.4Q35max5	42	50	35	-	-	7200.0 (0)	6.5	5.4	10.2	7.8	
p43.4Q3max1	42	50	3	-	-	7200.0 (0)	0.8	0.8	1.3	1.3	1
p43.4Q40max5	42	50	40	7200.0 (0)	12.0	7200.0 (0)	6.6	5.4	10.7	7.8	
p43.4Q45max5	42	50	45	-	-	7200.0 (0)	6.7	5.4	11.3	7.8	
p43.4Q4max1	42	50	4	-	-	7200.0 (0)	0.8	0.8	1.3	1.3	1
p43.4Q500max1	42	50	500	-	-	7200.0 (0)	5.7	5.4	9.1	7.8	
p43.4Q50max5	42	50	50	-	-	7200.0 (0)	6.8	5.4	11.3	7.8	
p43.4Q5max1	42	50	5	-	-	7200.0 (0)	0.8	0.8	1.3	1.3	1
p43.4Q60max5	42	50	60	-	-	7200.0 (0)	8.9	5.4	11.4	7.8	
p43.4Q6max1	42	50	6	-	-	7200.0 (0)	0.8	0.8	1.4	1.4	1
p43.4Q7max1	42	50	7	-	-	7200.0 (0)	0.9	0.9	1.5	1.5	1
p43.4Q80max5	42	50	80	-	-	7200.0 (0)	8.8	5.4	11.4	7.7	
p43.4Q8max1	42	50	8	-	-	7200.0 (0)	0.9	1.0	1.7	1.8	1
p43.4Q9max1	42	50	9	-	-	7200.0 (0)	1.2	1.1	2.1	1.8	1

Table A.6: Computational results - instances from Class 2

				Average time (# of instances solved)							
name	n	m	C	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	inf
n10m5Q10	9	5	10	**0.0**	**0.0**	**0.0**	0.1	0.1	0.2	0.2	
n10m5Q15	9	5	15	**0.0**	**0.0**	**0.0**	0.1	0.1	0.2	0.2	
n10m5Q20	9	5	20	-	**0.0**	**0.0**	0.1	0.1	0.2	0.2	
n10m5Q25	9	5	25	-	-	**0.0**	0.1	0.1	0.2	0.2	
n10m5Q30	9	5	30	-	-	**0.0**	0.1	0.1	0.2	0.2	
n10m5Q500	9	5	500	**0.0**	-	**0.0**	0.1	0.1	0.2	0.2	
n10m10Q10	9	10	10	**0.0** (3)	**0.0** (3)	**0.0** (3)	0.1 (3)	0.1 (3)	0.2 (3)	0.2 (3)	7
n10m10Q15	9	10	15	**0.0** (9)	**0.0** (9)	18.5 (9)	0.1 (9)	0.1 (9)	0.2 (9)	0.2 (9)	1
n10m10Q20	9	10	20	**0.0**	**0.0**	**0.0**	0.1	0.1	0.2	0.2	
n10m10Q25	9	10	25	-	**0.0**	**0.0**	0.1	0.1	0.2	0.2	
n10m10Q30	9	10	30	-	**0.0**	**0.0**	0.1	0.1	0.2	0.2	
n10m10Q500	9	10	500	**0.0**	-	**0.0**	0.1	0.1	0.2	0.2	
n10m15Q10	9	15	10	-	-	-	-	-	-	-	10
n10m15Q15	9	15	15	**0.0 (1)**	**0.0 (1)**	**0.0** (1)	0.1 (1)	0.1 (1)	0.2 (1)	0.2 (1)	9
n10m15Q20	9	15	20	**0.0 (4)**	**0.0 (4)**	**0.0** (4)	0.1 (4)	0.1 (4)	0.2 (4)	0.2 (4)	6
n10m15Q25	9	15	25	**0.0 (6)**	**0.0 (6)**	**0.0** (6)	0.1 (6)	0.1 (6)	0.2 (6)	0.2 (6)	4
n10m15Q30	9	15	30	**0.0 (8)**	**0.0 (8)**	**0.0** (8)	0.1 (8)	0.1 (8)	0.2 (8)	0.2 (8)	2
n10m15Q500	9	15	500	**0.0**	-	**0.0**	0.1	0.1	0.2	0.2	
n15m5Q10	14	5	10	0.4	1.0	0.9	**0.3**	**0.3**	0.6	0.6	
n15m5Q15	14	5	15	**0.3**	1.0	0.8	0.4	0.4	0.6	0.6	
n15m5Q20	14	5	20	-	1.0	0.8	**0.4**	**0.4**	0.6	0.7	
n15m5Q25	14	5	25	-	-	0.8	**0.4**	**0.4**	0.6	0.7	
n15m5Q30	14	5	30	-	-	0.8	**0.4**	**0.4**	0.6	0.7	
n15m5Q500	14	5	500	**0.3**	-	0.8	0.4	0.4	0.6	0.7	
n15m10Q10	14	10	10	1801.4 (3)	1.0 (3)	1.0 (3)	**0.2 (3)**	**0.2** (3)	0.4 (3)	0.4 (3)	7
n15m10Q15	14	10	15	0.4 (9)	1.0 (9)	0.8 (9)	**0.2 (9)**	**0.2** (9)	0.4 (9)	0.4 (9)	1
n15m10Q20	14	10	20	0.3	**0.0**	720.5 (9)	0.2	0.2	0.4	0.4	
n15m10Q25	14	10	25	0.2	**0.0**	0.7	0.2	0.2	0.4	0.4	
n15m10Q30	14	10	30	-	**0.0**	0.7	0.2	0.2	0.4	0.4	
n15m10Q500	14	10	500	**0.2**	-	0.7	0.2	0.2	0.4	0.4	
n15m15Q10	14	15	10	-	-	-	-	-	-	-	10
n15m15Q15	14	15	15	1.9 (6)	1.0 (6)	4108.5 (3)	**0.2 (6)**	**0.2** (6)	0.4 (6)	0.4 (6)	4
n15m15Q20	14	15	20	1.0 (8)	**0.0** (8)	636.6 (8)	0.2 (8)	0.2 (8)	0.4 (8)	0.4 (8)	2
n15m15Q25	14	15	25	0.1	**0.0**	0.2	0.2	0.2	0.4	0.4	
n15m15Q30	14	15	30	0.0	**0.0**	0.2	0.2	0.2	0.4	0.4	
n15m15Q500	14	15	500	**0.0**	-	0.2	0.2	0.2	0.4	0.4	
n20m5Q10	19	5	10	2.6	**1.0**	2.9	2.1	2.2	3.1	3.2	
n20m5Q15	19	5	15	0.3	**0.0**	2.1	2.2	2.3	3.2	3.3	
n20m5Q20	19	5	20	-	**0.0**	2.2	2.2	2.3	3.2	3.2	
n20m5Q25	19	5	25	-	-	**2.2**	2.2	2.3	3.2	3.3	
n20m5Q30	19	5	30	-	-	**2.2**	2.2	2.3	3.2	3.3	
n20m5Q500	19	5	500	**0.2**	-	2.2	2.2	2.3	3.2	3.3	
n20m10Q10	19	10	10	1831.6 (6)	1806.0 (6)	1031.4 (6)	**1.2** (7)	1.2 (7)	1.9 (7)	2.0 (7)	3
n20m10Q15	19	10	15	66.7	31.0	723.1 (9)	**1.4**	**1.4**	2.3	2.2	
n20m10Q20	19	10	20	53.1	**1.0**	2.1	1.5	1.6	2.5	2.4	
n20m10Q25	19	10	25	53.1	**1.0**	1.7	1.6	1.6	2.5	2.4	
n20m10Q30	19	10	30	-	**1.0**	1.6	1.6	1.6	2.6	2.4	
n20m10Q500	19	10	500	53.2	-	1.9	1.9	**1.8**	2.7	2.8	
n20m15Q10	19	15	10	-	-	-	-	-	-	-	10
n20m15Q15	19	15	15	5305.1 (3)	3399.0 (6)	4594.8 (3)	**0.4** (8)	**0.4** (8)	0.7 (8)	0.7 (8)	2
n20m15Q20	19	15	20	3072.6 (6)	910.0 (9)	722.7 (9)	**0.6**	**0.6**	1.1	1.1	
n20m15Q25	19	15	25	172.1	6.0	721.6 (9)	**0.8**	**0.8**	1.4	1.4	
n20m15Q30	19	15	30	114.4	2.0	2.1	**0.9**	**0.9**	1.5	1.4	
n20m15Q500	19	15	500	116.9	-	1.1	1.0	**0.9**	1.5	1.4	

Table A.7: Computational results - instances from Class 2 (cont.)

				Average time (# of instances solved)							
name	n	m	C	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$	inf
n25m5Q10	24	5	10	**1.9**	10.0	6.0	7.3	8.3	9.6	10.3	
n25m5Q15	24	5	15	**0.8**	2.0	5.8	8.0	8.2	10.2	10.4	
n25m5Q20	24	5	20	**0.8**	2.0	6.0	8.3	8.7	11.2	11.6	
n25m5Q25	24	5	25	-	-	**6.0**	8.2	8.9	10.8	11.6	
n25m5Q30	24	5	30	-	-	**6.0**	8.3	8.5	10.8	11.6	
n25m5Q500	24	5	500	**0.8**	-	6.0	8.2	8.6	11.2	11.7	
n25m10Q10	24	10	10	3684.2 (6)	2004.0 (7)	1606.2 (7)	19.6 (9)	19.0 (9)	**11.1** (9)	18.7 (9)	1
n25m10Q15	24	10	15	137.4	67.0	**6.9**	11.5	7.0	9.3	8.3	
n25m10Q20	24	10	20	13.5	**5.0**	6.4	10.4	5.9	10.7	7.5	
n25m10Q25	24	10	25	13.6	**4.0**	5.7	11.9	5.9	9.8	7.5	
n25m10Q30	24	10	30	-	**4.0**	6.2	12.8	6.1	10.4	7.9	
n25m10Q500	24	10	500	13.6	-	**6.2**	11.9	6.6	10.8	8.2	
n25m15Q10	24	15	10	-	7200.0	4806.1 (1)	5.4 (3)	**5.0** (3)	5.5 (3)	6.4 (3)	7
n25m15Q15	24	15	15	5786.3 (2)	3167.0 (6)	4323.8 (4)	26.9	19.6	**13.7**	17.9	
n25m15Q20	24	15	20	3804.2 (5)	1385.0 (9)	7.3	14.0	**5.2**	8.6	6.5	
n25m15Q25	24	15	25	1387.4 (9)	59.0	6.1	22.3	**5.2**	10.2	6.4	
n25m15Q30	24	15	30	565.2	-	5.9	25.1	**5.0**	11.1	6.6	
n25m15Q500	24	15	500	371.5	-	5.5	27.7	**4.8**	13.2	6.3	

Table A.8: Computational results - instances from Class 3

				Average time (# of instances solved)						
name	n	m	C	$\mathcal{BE}$	$\mathcal{CU}$	$\mathcal{CP}$	$\mathcal{M}^{\mathcal{C}}$	$\mathcal{M}^{\mathcal{T}}$	$\mathcal{M}^{\mathcal{C}}_{\beta}$	$\mathcal{M}^{\mathcal{T}}_{\beta}$
m5Q5	12	5	5	**0.0**	**0.0**	0.9	0.1	0.1	0.2	0.2
m5Q10	12	5	10	**0.0**	**0.0**	**0.0**	0.1	0.1	0.2	0.2
m5Q15	12	5	15	**0.0**	**0.0**	**0.0**	0.1	0.1	0.2	0.2
m5Q20	12	5	20	**0.0**	-	**0.0**	0.1	0.1	0.2	0.2
m5Q25	12	5	25	**0.0**	-	**0.0**	0.1	0.1	0.2	0.2
m5Q30	12	5	30	**0.0**	-	**0.0**	0.1	0.1	0.2	0.2
m5Q500	12	5	500	0.1	-	**0.0**	0.1	0.1	0.2	0.2
m10Q5	22	10	5	1.8	2.0	52.8	**0.8**	0.9	1.4	1.6
m10Q10	22	10	10	86.8	165.0	9.9	**2.2**	**2.2**	3.1	3.2
m10Q15	22	10	15	61.5	30.0	5.6	2.8	**2.5**	3.7	3.5
m10Q20	22	10	20	**1.7**	2.0	4.1	3.1	2.7	4.1	3.9
m10Q25	22	10	25	**1.4**	2.0	4.1	3.1	2.8	4.3	4.0
m10Q30	22	10	30	**1.4**	2.0	4.2	3.1	2.9	4.4	4.1
m10Q500	22	10	500	**1.4**	-	3.9	3.1	2.9	4.5	4.1
m15Q5	32	15	5	2005.8 (4)	2529.0 (9)	6557.9 (1)	5034.6 (6)	6616.0 (3)	**1878.5** (9)	6355.2 (3)
m15Q10	32	15	10	6523.2 (6)	6493.0 (1)	**368.1**	3523.1 (6)	2893.6 (8)	864.0	1877.4 (9)
m15Q15	32	15	15	4124.0 (8)	3284.0 (6)	**60.1**	3051.3 (6)	708.5	1435.2 (9)	351.9
m15Q20	32	15	20	917.5	269.0	**20.0**	2016.8 (9)	56.5	251.3	31.6
m15Q25	32	15	25	118.2	40.0	**15.2**	1365.0	34.9	186.4	27.8
m15Q30	32	15	30	100.7	43.0	**15.1**	1643.7 (9)	29.7	172.1	27.2
m15Q500	32	15	500	99.4	-	**14.6**	2176.7 (9)	29.6	222.2	27.1

Appendix B

AI Planning: Additional Results

B.1 Average Optimality Gaps

Table B.1 presents the average optimality gap for each tested heuristic in the initial states. The average is computed over the 314 instances where the cost-optimal plans are known.

Table B.1: Average optimality gap.

domain	LP	$\mathcal{M}_2$	$\mathcal{M}_4$	$\mathcal{M}_8$	$\mathcal{M}_{16}$	$\mathcal{M}_{32}$	$\mathcal{M}_{64}$	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
bar11	0.73	0.64	0.57	0.54	0.51	0.49	**0.48**	0.76	0.76	0.76	0.74	0.74	0.74
bar14	**0.13**	0.69	0.56	0.47	0.40	0.32	0.26	0.24	0.24	0.19	0.19	0.19	0.19
chi14	0.22	0.80	0.77	0.73	0.70	0.66	0.65	0.33	0.30	0.26	0.24	0.21	**0.15**
ele11	**0.48**	0.78	0.77	0.77	0.77	0.74	0.68	0.61	0.61	0.61	0.61	0.61	0.61
flo11	**0.05**	0.83	0.81	0.81	0.80	0.79	0.78	0.27	0.27	0.27	0.27	0.26	0.26
flo14	**0.07**	0.82	0.80	0.79	0.78	0.77	0.77	0.29	0.29	0.29	0.28	0.28	0.28
ged14	0.88	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	1.00	1.00	1.00	1.00	1.00	1.00
nom11	**0.00**	0.70	0.63	0.56	0.48	0.41	0.37	0.02	0.02	0.01	0.01	0.01	**0.00**
ope11	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	1.00	1.00	1.00	1.00	1.00	1.00
par11	**0.13**	0.78	0.72	0.66	0.60	0.52	0.46	0.18	0.18	0.18	0.18	0.18	0.18
par14	**0.11**	0.77	0.71	0.65	0.59	0.51	0.44	0.15	0.15	0.15	0.15	0.15	0.15
peg11	**0.43**	0.61	0.58	0.56	0.56	0.53	0.46	0.99	0.99	0.95	0.93	0.92	0.92
sca11	**0.04**	0.79	0.77	0.76	0.75	0.74	0.74	0.04	0.04	0.04	0.04	0.04	0.04
sok11	**0.05**	0.47	0.44	0.40	0.37	0.35	0.32	0.21	0.19	0.18	0.17	0.17	0.17
tra11	0.94	0.19	0.19	0.19	0.18	0.18	**0.18**	0.99	0.99	0.99	0.99	0.99	0.99
tra14	0.86	0.40	0.39	0.38	0.38	0.37	**0.37**	0.97	0.97	0.97	0.97	0.97	0.97
vis11	**0.03**	0.70	0.64	0.61	0.58	0.55	0.51	0.06	0.05	0.05	0.04	0.04	0.04
vis14	**0.02**	0.85	0.83	0.82	0.80	0.79	0.77	0.04	0.04	0.04	0.04	0.04	0.03
woo11	**0.12**	0.73	0.70	0.67	0.63	0.58	0.54	0.24	0.20	0.17	0.16	0.15	0.14

B.2 Solving Time and States Evaluated

Table B.2 shows the average solving time and states evaluated for all our relaxed MDD heuristics. The average values are calculated over the instances where all relaxed MDD methods solve the task. Column "#" shows the number of instances where all approaches found a cost-optimal plan, i.e., the number of instances used to compute the average performance metrics. Similarly, Table B.3 shows the average performance metrics for the relaxed BDD heuristics.

Table B.2: Overall MDD Performance.

		Average Time (sec)						Average # States Evaluated					
domain	#	$\mathcal{M}_2$	$\mathcal{M}_4$	$\mathcal{M}_8$	$\mathcal{M}_{16}$	$\mathcal{M}_{32}$	$\mathcal{M}_{64}$	$\mathcal{M}_2$	$\mathcal{M}_4$	$\mathcal{M}_8$	$\mathcal{M}_{16}$	$\mathcal{M}_{32}$	$\mathcal{M}_{64}$
bar11	0	-	-	-	-	-	-	-	-	-	-	-	-
bar14	0	-	-	-	-	-	-	-	-	-	-	-	-
chi14	4	331.3	320.7	**313.2**	317.5	339.8	377.7	694,820.5	685,161.8	675,891.8	671,269.8	671,269.3	**671,269.3**
ele11	7	**116.4**	252.4	543.3	797.4	815.1	948.9	16,271.4	16,181.4	15,626.0	10,800.9	5,852.6	**3,399.3**
flo11	0	-	-	-	-	-	-	-	-	-	-	-	-
flo14	0	-	-	-	-	-	-	-	-	-	-	-	-
ged14	20	**1.9**	4.1	8.6	18.5	40.8	89.2	**9.1**	**9.1**	**9.1**	**9.1**	**9.1**	**9.1**
nom11	5	260.8	61.0	5.7	0.6	**0.4**	**0.4**	160,914.8	22,354.8	1,257.0	38.8	7.0	**2.2**
ope11	20	**0.3**	0.5	1.1	2.6	6.1	14.8	**1.0**	**1.0**	**1.0**	**1.0**	**1.**	**1.0**
par11	0	-	-	-	-	-	-	-	-	-	-	-	-
par14	0	-	-	-	-	-	-	-	-	-	-	-	-
peg11	17	**15.1**	29.6	51.4	78.3	119.0	170.2	6,629.7	5,542.1	4,255.1	2,770.9	1,909.2	**1,140.4**
sca11	1	**0.1**	**0.1**	0.2	**0.1**	**0.1**	**0.1**	357.0	**348.0**	**348.0**	**348.0**	**348.0**	**348.0**
sok11	16	**62.0**	66.3	81.0	103.6	146.5	236.4	9,481.3	6,731.4	5,391.7	3,877.4	2,783.6	**1,952.5**
tra11	1	16.6	**14.3**	15.8	22.3	27.7	43.6	1,525.0	656.0	449.0	321.0	266.0	**197.0**
tra14	1	352.4	362.0	419.5	485.0	477.6	**345.5**	49,690.0	24,330.0	13,600.0	8,718.0	4,105.0	**1,258.0**
vis11	9	118.9	89.1	**84.4**	96.9	101.6	140.6	130,189.4	84,190.7	50,985.2	29,250.3	14,804.4	**9,031.7**
vis14	3	370.2	277.3	**234.2**	235.8	265.5	305.2	442,995.3	268,824.7	145,634.3	84,368.3	45,795.7	**23,254.0**
woo11	2	508.9	296.4	163.6	97.2	47.8	**24.4**	371,424.5	125,073.5	30,653.5	8,376.5	2,305.0	**594.0**
Average		78.4	**71.8**	91.3	118.4	138.5	184.3	68,466.8	47,240.4	37,259.6	32,195.5	29,087.1	**27,488.2**

Table B.3: Overall BDD Performance.

		Average Time (sec)						Average # States Evaluated					
domain	#	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$	$\mathcal{B}_2$	$\mathcal{B}_4$	$\mathcal{B}_8$	$\mathcal{B}_{16}$	$\mathcal{B}_{32}$	$\mathcal{B}_{64}$
bar11	0	-	-	-	-	-	-	-	-	-	-	-	-
bar14	14	**10.4**	22.8	37.7	70.3	109.9	117.1	32,398.7	30,821.4	27,411.1	21,713.3	14,215.2	**5,653.9**
chi14	18	448.8	369.7	230.9	122.8	**114.7**	123.9	367,694.7	291,537.8	180,355.6	96,305.4	82,250.7	**74,784.5**
ele11	15	**17.0**	30.8	60.1	109.1	239.5	514.2	45,027.3	43,549.5	42,218.9	42,054.8	42,103.2	**41,686.9**
flo11	4	86.7	87.1	**86.6**	119.1	154.0	235.8	173,597.0	116,613.3	67,609.5	44,016.0	26,286.3	**19,634.5**
flo14	1	**89.5**	128.2	179.1	197.6	175.0	230.1	210,121.0	178,847.0	126,276.0	58,064.0	38,774.0	**22,563.0**
ged14	20	**1.4**	2.1	3.7	5.2	10.4	21.8	2,110.9	2,110.9	2,110.9	2,110.9	2,110.9	**2,110.9**
nom11	16	**0.8**	0.9	1.0	1.1	1.5	2.2	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
ope11	20	**0.1**	**0.1**	**0.1**	0.2	0.3	0.5	**2.0**	**2.0**	**2.0**	**2.0**	**2.0**	**2.0**
par11	3	**19.3**	31.9	51.0	83.9	146.9	273.3	**566.0**	**566.0**	566.7	567.7	569.3	570.3
par14	6	**24.2**	43.3	70.5	128.0	234.7	459.4	**475.8**	476.0	478.7	477.8	477.7	479.0
peg11	17	**14.4**	19.6	29.8	51.5	95.6	188.4	56,778.7	54,782.5	51,625.1	51,078.6	**51,078.6**	**51,078.6**
sca11	5	**28.4**	49.3	96.4	213.7	418.7	626.6	37,136.6	36,877.2	37,130.4	37,498.8	36,169.4	**31,590.2**
sok11	18	17.5	9.2	**7.2**	8.4	11.9	15.1	20,372.4	6,527.0	2,932.3	1,906.6	1,436.1	**1,008.1**
tra11	0	-	-	-	-	-	-	-	-	-	-	-	-
tra14	0	-	-	-	-	-	-	-	-	-	-	-	-
vis11	16	**0.5**	0.7	1.0	0.6	1.1	1.8	2,350.3	2,011.3	1,290.2	414.5	391.4	**343.6**
vis14	16	0.5	0.3	**0.2**	**0.2**	**0.2**	0.3	1,878.4	713.7	189.6	84.9	21.9	**12.9**
woo11	13	108.0	14.3	**9.6**	16.1	29.6	30.5	209,047.0	18,810.2	8,182.2	6,205.8	5,943.4	**3,829.8**
Average		55.8	45.9	**39.8**	44.4	71.3	118.1	64,365.0	42,306.2	29,470.8	20,436.1	18,122.5	**16,329.0**

Appendix C

Cut Generation and Lifting: Additional Results

C.1 Experiments Comparing Different BDD Widths

Table C.1 presents the average performance for CPLEX and our four alternatives (i.e, BW, BWL, BG, and BGL) with three different maximum width values, $\mathcal{W} \in \{2000, 3000, 4000\}$, over the SOC-CC instances. The table shows the number of instances solved, average root gap, and average final gap for all techniques. Our four alternatives with $\mathcal{W} \in \{2000, 3000, 4000\}$ each outperform CPLEX. $\mathcal{W} = 4000$ achieves the best overall performance across the four combinatorial cut-and-lift alternatives.

Table C.1: Average performance of all techniques for different BDD widths for SOC-CC.

	Width	# Solve	Root Gap	Final Gap
CPLEX		137	20.82%	7.75%
BW	2000	137	20.31%	7.35%
	3000	**140**	20.11%	**7.25%**
	4000	139	**20.01%**	**7.25%**
BWL	2000	149	16.61%	6.09%
	3000	**150**	16.03%	6.04%
	4000	**150**	**15.68%**	**5.89%**
BG	2000	160	14.93%	5.21%
	3000	159	14.05%	4.87%
	4000	**166**	**13.47%**	**4.59%**
BGL	2000	160	14.91%	5.00%
	3000	**168**	14.03%	4.66%
	4000	**168**	**13.44%**	**4.43%**

Similarly, Table C.2 presents the average performance over the SOC-K instances for CPLEX, our four alternatives (i.e, BW, BWL, BG, and BGL), and cover cuts with BDD lifting (i.e., BCL) with three different maximum width values, $\mathcal{W} \in \{2000, 3000, 4000\}$. Overall, $\mathcal{W} = 4000$ achieves the best or comparable performance across the five BDD approaches.

Table C.2: Average performance of all techniques for different BDD widths for SOC-K.

	Width	# Solve	Root Gap	Final Gap
CPLEX		70	2.84%	0.39%
BCL	2000	**71**	2.72%	0.29%
	3000	70	2.69%	0.28%
	4000	70	**2.67%**	**0.27%**
BW	2000	66	3.64%	**0.52%**
	3000	**67**	3.63%	**0.52%**
	4000	64	**3.63%**	0.53%
BWL	2000	74	2.84%	0.27%
	3000	72	**2.80%**	0.28%
	4000	**74**	**2.80%**	**0.26%**
BG	2000	75	1.62%	0.16%
	3000	**78**	1.45%	**0.15%**
	4000	76	**1.34%**	0.16%
BGL	2000	83	1.61%	0.04%
	3000	87	1.44%	0.02%
	4000	**88**	**1.33%**	**0.01%**

C.2 Average Performance Comparison for Knapsack Chance Constraints

Table C.3 shows the number of instances solved, average root gap, and average final gap for each n, m, and Ω combination with $\mathcal{W} = 4000$. Similarly, Table C.4 shows the average number of nodes in the branch-and-bound search and the average run time for the instances that all techniques solved to optimality. Column # shows the number of instances used to compute the average results.

Table C.3: Gap comparison across all techniques for SOC-K instances.

			# Instances Solved to Optimality								Av. Root Gap (%)								Av. Final Gap (%)							
n	m	Ω	CPLEX	C	CL	BCl	BW	BWL	BG	BGL	CPLEX	C	CL	BCl	BW	BWL	BG	BGL	CPLEX	C	CL	BCl	BW	BWL	BG	BGL
100	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	0.9	1.8	1.4	1.4	2.0	1.4	**0.5**	**0.5**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	2.4	2.9	2.2	1.9	3.2	2.0	0.5	**0.4**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	3.7	3.9	2.8	2.2	4.4	2.6	**0.3**	**0.3**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	20	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	1.9	2.8	2.4	2.3	3.2	2.5	**1.1**	**1.1**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	4.5	5.0	4.5	4.4	5.4	4.3	**1.5**	**1.5**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	3	**5**	**5**	**5**	6.1	6.3	5.9	5.5	7.1	5.4	**1.5**	**1.5**	**0.0**	**0.0**	**0.0**	**0.0**	0.4	**0.0**	**0.0**	**0.0**
125	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	0.7	1.4	0.9	1.0	1.8	1.2	**0.7**	**0.7**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	1.8	2.2	1.6	1.5	2.5	1.6	**0.8**	**0.8**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	2.7	2.8	2.0	1.7	3.3	1.8	**0.9**	**0.9**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	20	1	**5**	**5**	**5**	**5**	4	**5**	**5**	**5**	2.1	2.9	2.6	2.5	3.1	2.7	**1.9**	**1.9**	**0.0**	**0.0**	**0.0**	**0.0**	0.1	**0.0**	**0.0**	**0.0**
		3	1	2	1	1	1	3	4	**5**	4.0	4.3	4.1	4.1	4.7	4.1	**2.4**	**2.4**	1.0	0.9	0.8	0.8	1.2	0.4	0.1	**0.0**
		5	0	0	0	0	0	1	2	**5**	5.8	6.0	5.6	5.5	6.5	5.6	**2.7**	**2.7**	2.5	2.5	2.0	1.6	2.8	1.6	0.5	**0.0**
150	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	0.5	1.1	0.9	0.9	1.3	1.1	**0.7**	**0.7**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	1.6	1.9	1.5	1.4	2.1	1.5	**0.9**	**0.9**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
		5	**5**	**5**	**5**	**5**	3	**5**	**5**	**5**	2.7	2.8	2.4	2.3	3.2	2.6	**1.3**	**1.3**	**0.0**	**0.0**	**0.0**	**0.0**	0.2	**0.0**	**0.0**	**0.0**
	20	1	4	4	4	4	3	4	4	**5**	1.6	2.3	1.9	1.9	2.5	2.2	**1.6**	**1.6**	0.1	0.1	0.2	0.1	0.3	0.1	0.2	**0.0**
		3	0	0	0	0	0	1	1	**3**	3.3	3.7	3.3	3.1	3.9	3.3	**2.1**	**2.1**	1.3	1.3	1.0	0.7	1.6	0.8	0.6	**0.1**
		5	0	0	0	0	0	0	0	**5**	4.7	5.1	4.5	4.3	5.3	4.4	2.6	**2.5**	2.1	2.3	1.8	1.6	2.7	1.6	1.5	**0.0**

Table C.4: Nodes and time comparison for SOC-K instances.

				Av. Nodes Explore								Av. Time (sec)							
n	m	Ω	#	CPLEX	C	CL	BCl	BW	BWL	BG	BGL	CPLEX	C	CL	BCl	BW	BWL	BG	BGL
100	10	1	5	2,187.4	9,493.2	3,407.0	3,808.4	13,278.6	5,055.4	190.8	**169.8**	3.3	2.7	**1.4**	8.9	16.9	9.3	149.4	34.0
		3	5	15,911.0	38,018.4	18,615.8	13,374.4	77,135.4	11,973.4	126.6	**100.2**	9.4	12.1	**6.6**	11.2	51.2	11.2	188.0	52.3
		5	5	56,023.8	73,318.4	34,887.8	11,064.8	197,240.2	24,390.4	181.8	**170.8**	32.3	24.7	13.2	**10.8**	109.3	15.1	431.5	61.1
	20	1	5	12,093.2	84,851.6	36,812.6	37,478.6	137,687.4	56,573.2	2,324.8	**1,116.4**	**19.1**	38.7	19.3	33.9	127.1	45.4	477.1	165.3
		3	5	328,741.6	577,413.8	349,153.2	299,747.8	1,216,771.8	289,571.6	2,545.6	**1,301.0**	375.6	324.8	209.8	179.7	836.9	**176.7**	690.6	246.4
		5	3	796,297.7	1,030,628.0	1,073,455.0	431,804.0	1,648,576.7	655,462.0	1,119.7	**729.0**	970.4	725.8	753.4	**332.2**	1344.0	474.2	1318.0	445.3
125	10	1	5	**2,557.4**	26,109.4	5,254.4	6,706.8	59,136.2	22,296.4	3,862.2	4,874.4	3.3	6.9	**2.4**	15.7	41.8	20.1	111.2	33.5
		3	5	16,810.2	89,336.0	35,721.0	28,278.2	190,739.6	39,481.2	8,113.8	**4,436.2**	**11.8**	23.8	12.8	22.1	88.9	25.3	435.4	71.8
		5	5	157,561.0	301,041.4	253,127.8	87,198.2	1,290,400.0	89,929.6	6,137.0	**4,415.0**	105.5	102.1	84.8	45.9	417.1	**42.8**	863.8	85.1
	20	1	4	243,540.3	1,134,037.8	628,233.8	584,492.3	2,016,914.3	568,772.5	120,848.5	**90,177.3**	357.6	564.6	327.2	332.0	1038.2	288.9	422.5	**237.8**
		3	1	445,816.0	902,386.0	476,674.0	513,874.0	2,206,133.0	1,047,598.0	79,141.0	**29,092.0**	645.0	668.2	**340.7**	431.7	1726.1	663.5	1098.4	631.8
		5	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
150	10	1	5	**1,835.4**	39,164.8	14,878.4	18,218.4	61,288.8	33,738.0	17,672.8	8,190.4	**2.9**	10.0	5.2	23.3	44.7	28.8	101.8	38.3
		3	5	267,752.2	1,156,660.6	1,084,202.4	400,828.6	1,886,360.8	399,942.4	56,666.0	**36,921.0**	183.1	389.2	348.5	146.3	764.2	136.9	358.0	**94.1**
		5	3	394,755.0	877,686.3	423,235.0	337,257.7	2,718,139.7	658,492.3	**87,523.7**	108,409.0	252.5	262.0	157.4	**132.2**	1092.1	218.6	566.7	182.8
	20	1	3	**338,981.0**	2,736,707.7	975,963.7	1,014,534.7	3,681,778.3	2,585,156.0	586,007.0	418,882.0	567.4	1456.4	635.3	632.5	2108.4	1296.5	839.5	**339.8**
		3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

C.3 Average Performance Comparison for General Chance Constraints

Tables C.5-C.7 show the number of instances solved, average root gap, and average final gap for each n, m, Ω, and t combination, with $\mathcal{W} = 4000$. Similarly, Tables C.8-C.10 show the average number of nodes in the branch-and-bound search and the average run time for the instances that all techniques solved to optimality. Column # shows the number of instances used to compute the average results.

Table C.5: Gap comparison across all techniques for instances with $t = 0.1$

			# Instances Solved to Optimality					Av. Root Gap (%)					Av. Final Gap (%)				
n	m	Ω	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL
75	10	1	**5**	**5**	**5**	**5**	**5**	7.6	10.0	7.1	**3.8**	**3.8**	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	47.6	45.6	22.7	16.2	**15.5**	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	**5**	56.1	48.0	20.2	18.0	**17.1**	0.0	0.0	0.0	0.0	0.0
	20	1	**5**	**5**	**5**	**5**	**5**	54.7	56.5	54.9	**47.0**	47.1	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	83.0	76.4	27.9	**26.2**	28.5	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	**5**	81.3	31.8	9.4	**8.5**	9.4	0.0	0.0	0.0	0.0	0.0
100	10	1	**5**	**5**	**5**	**5**	**5**	6.1	7.5	6.7	**5.4**	**5.4**	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	34.3	34.0	27.6	**23.3**	**23.3**	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	**5**	40.8	39.6	26.8	22.5	**22.4**	0.0	0.0	0.0	0.0	0.0
	20	1	0	0	0	**2**	1	27.1	29.1	27.0	**23.5**	**23.5**	17.7	18.1	13.5	**5.3**	7.5
		3	**5**	**5**	**5**	**5**	**5**	65.3	63.0	48.2	**46.0**	**46.0**	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	**5**	67.5	60.0	42.6	**39.8**	39.9	0.0	0.0	0.0	0.0	0.0
125	10	1	**4**	**4**	**4**	**4**	**4**	4.4	5.5	4.8	**4.1**	**4.1**	**0.3**	0.4	0.4	**0.3**	0.4
		3	0	0	0	0	0	33.8	33.6	30.6	**29.5**	**29.5**	25.4	23.5	18.1	16.5	**15.0**
		5	0	0	**1**	**1**	**1**	38.0	36.3	32.0	**31.1**	**31.1**	27.9	24.5	13.2	12.5	**9.7**
	20	1	0	0	0	0	0	19.6	20.5	20.0	**18.7**	**18.7**	16.7	17.7	16.7	**15.3**	15.4
		3	0	0	**1**	**1**	**1**	58.0	58.0	53.7	49.7	**49.5**	53.7	51.9	32.4	25.9	**25.2**
		5	0	1	1	**2**	**2**	56.2	55.0	43.3	**40.4**	**40.5**	49.4	21.5	10.2	7.2	**6.1**
Total/Average			59	60	62	**65**	64	43.4	39.5	28.1	**25.2**	25.3	10.6	8.8	5.8	4.6	**4.4**

Table C.6: Gap comparison across all techniques for instances with $t = 0.2$

			# Instances Solved to Optimality					Av. Root Gap (%)					Av. Final Gap (%)				
n	m	Ω	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL
75	10	1	**5**	**5**	**5**	**5**	**5**	2.9	4.5	3.6	**2.0**	**2.0**	0.0	0.0	0.0	0.0	0.0
		3	2	4	5	**5**	**5**	10.1	10.7	9.2	**5.4**	**5.4**	1.4	0.3	**0.0**	**0.0**	**0.0**
		5	1	1	3	**5**	**5**	13.1	13.8	11.5	**7.0**	**7.0**	4.5	3.9	1.2	**0.0**	**0.0**
	20	1	2	2	**3**	**3**	**3**	11.6	13.9	11.6	**7.8**	**7.8**	3.2	3.8	2.6	1.4	**0.9**
		3	0	0	0	0	1	31.6	32.4	31.0	**22.1**	**22.1**	25.6	25.6	23.6	14.1	**13.7**
		5	0	0	0	1	1	34.3	34.7	31.7	**22.6**	**22.6**	27.2	28.0	24.8	14.2	**13.9**
100	10	1	**5**	**5**	**5**	**5**	**5**	2.6	3.8	3.1	**2.4**	**2.4**	0.0	0.0	0.0	0.0	0.0
		3	0	1	1	2	**4**	8.3	8.8	8.0	6.3	**6.2**	4.4	4.2	3.3	1.5	**1.0**
		5	0	0	0	0	0	12.4	12.8	12.1	**10.5**	**10.5**	9.0	9.6	8.7	6.9	**6.6**
	20	1	0	0	1	2	**1**	7.8	9.3	8.7	**6.6**	**6.6**	3.7	4.5	3.9	**2.2**	2.4
		3	0	0	0	0	0	26.4	26.7	25.9	**22.6**	**22.6**	23.2	24.2	23.6	19.6	**19.4**
		5	0	0	0	0	0	32.8	32.9	32.0	**28.5**	**28.5**	30.0	30.9	29.5	25.6	**25.5**
125	10	1	**5**	**5**	**5**	**5**	**5**	**1.6**	2.8	2.6	2.4	2.4	0.0	0.0	0.0	0.0	0.0
		3	0	0	0	0	0	7.9	8.1	7.9	**7.6**	**7.6**	5.6	6.0	5.7	5.4	**5.2**
		5	0	0	0	0	0	9.7	9.8	9.5	**8.9**	**8.9**	7.5	8.0	7.9	7.4	**7.4**
	20	1	0	0	0	0	0	**5.6**	6.5	6.3	5.7	5.7	3.7	4.4	4.2	3.7	**3.6**
		3	0	0	0	0	0	23.4	23.5	23.2	21.8	**21.8**	22.0	22.7	22.5	20.7	**20.7**
		5	0	0	0	0	0	26.8	26.8	26.5	**24.5**	**24.5**	25.4	26.1	25.8	23.5	**23.5**
Total/Average			20	23	28	33	**35**	14.9	15.7	14.7	**11.9**	**11.9**	10.9	11.2	10.4	8.1	**8.0**

Table C.7: Gap comparison across all techniques for instances with $t = 0.3$

			# Instances Solved to Optimality					Av. Root Gap (%)					Av. Final Gap (%)				
n	m	Ω	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL
75	10	1	**5**	**5**	**5**	**5**	**5**	1.2	2.7	1.8	**0.9**	**0.9**	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	3.6	4.4	3.4	**1.8**	**1.8**	0.0	0.0	0.0	0.0	0.0
		5	4	4	**5**	**5**	**5**	4.3	5.0	4.0	**2.1**	**2.1**	0.3	0.4	**0.0**	**0.0**	**0.0**
	20	1	**5**	**5**	**5**	**5**	**5**	3.7	5.5	4.4	**2.3**	2.4	0.0	0.0	0.0	0.0	0.0
		3	0	0	1	**4**	**4**	7.7	9.0	7.6	**4.3**	**4.3**	3.1	3.5	2.2	0.6	**0.2**
		5	0	0	0	3	**4**	8.8	9.8	7.9	**4.2**	4.8	4.8	5.6	3.6	**0.0**	0.2
100	10	1	**5**	**5**	**5**	**5**	**5**	**1.0**	2.1	1.7	1.2	1.2	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	2.3	3.1	2.8	**2.1**	**2.1**	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	**5**	2.9	3.6	3.5	**2.6**	**2.6**	0.0	0.0	0.0	0.0	0.0
	20	1	4	3	4	**5**	**5**	3.3	4.6	4.0	**3.1**	**3.1**	0.2	0.4	0.1	**0.0**	**0.0**
		3	0	0	0	**1**	**1**	6.8	7.5	6.8	**5.4**	**5.4**	4.2	4.8	4.0	2.7	**2.6**
		5	0	0	0	0	0	8.5	9.1	8.6	**6.7**	**6.7**	6.4	6.6	6.1	4.6	**4.4**
125	10	1	**5**	**5**	**5**	**5**	**5**	**0.3**	1.1	1.0	0.9	0.9	0.0	0.0	0.0	0.0	0.0
		3	**5**	**5**	**5**	**5**	**5**	**1.6**	2.2	2.1	1.9	1.9	0.0	0.0	0.0	0.0	0.0
		5	**5**	**5**	**5**	**5**	**5**	**1.8**	2.3	2.1	2.0	2.0	0.0	0.0	0.0	0.0	0.0
	20	1	**5**	4	**5**	**5**	**5**	**1.7**	2.8	2.6	2.4	2.4	**0.0**	0.1	**0.0**	**0.0**	**0.0**
		3	0	0	0	0	0	4.8	5.4	5.2	**4.5**	**4.5**	3.3	3.9	3.5	**3.0**	**3.0**
		5	0	0	0	0	0	7.5	8.0	7.8	**6.9**	**6.9**	6.3	6.7	6.5	5.7	**5.6**
Total/Average			58	56	60	68	**69**	4.0	4.9	4.3	**3.1**	**3.1**	1.6	1.8	1.4	**0.9**	**0.9**

Table C.8: Nodes and time comparison for instances solved with $t = 0.1$

				Av. Nodes Explore					Av. Time (sec)				
n	m	Ω	#	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL
75	10	1	5	12,184.8	16,966.8	5,472.6	544.0	**490.6**	22.8	18.9	**10.3**	49.6	17.6
		3	5	56,780.0	14,815.8	1,003.6	**344.8**	411.6	54.2	16.7	**4.7**	17.1	5.6
		5	5	48,045.0	3,253.4	250.0	278.6	**226.0**	52.5	7.5	**3.6**	10.3	3.7
	20	1	5	202,903.6	215,680.0	146,082.8	27,599.2	**26,056.8**	1,184.5	630.9	526.2	454.0	**309.0**
		3	5	13,220.0	2,036.0	7.0	**4.6**	5.6	58.3	26.1	**11.2**	32.7	11.3
		5	5	11,943.8	23.4	**2.0**	4.4	**2.0**	59.8	15.7	9.2	15.6	**9.0**
100	10	1	5	225,098.4	417,678.4	251,483.8	**79,218.2**	90,030.0	557.1	488.9	367.5	192.1	**152.5**
		3	5	887,680.0	844,533.2	104,978.8	99,247.0	**54,076.8**	1,648.1	1,003.0	185.6	281.8	**115.3**
		5	5	645,651.2	246,868.2	22,797.0	**14,450.2**	18,390.8	1,039.0	372.1	**53.5**	88.8	55.1
	20	1	0	-	-	-	-	-	-	-	-	-	-
		3	5	351,411.2	45,446.2	7,104.4	**5,298.8**	5,562.4	1,733.5	245.2	49.9	108.3	**49.6**
		5	5	270,318.6	10,668.0	2,959.6	**2,201.6**	2,292.8	1,177.4	70.6	30.9	55.4	**30.8**
125	10	1	4	426,022.3	1,342,855.3	935,676.0	**833,522.3**	990,455.3	1,039.1	1,117.8	959.8	**816.5**	910.5
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
	20	1	0	-	-	-	-	-	-	-	-	-	-
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
Average				262,604.9	263,402.1	123,151.5	**88,559.5**	99,000.1	718.9	334.5	184.4	176.9	**139.2**

Table C.9: Nodes and time comparison for instances solved with $t = 0.2$

				Av. Nodes Explore					Av. Time (sec)				
n	m	Ω	#	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL
75	10	1	5	12,121.4	17,446.4	8,475.4	1,150.2	**1,417.8**	20.8	18.4	**13.8**	42.6	21.3
		3	2	467,592.5	533,607.0	181,152.0	34,845.5	**34,824.5**	1,851.3	1,502.2	797.6	146.0	**104.5**
		5	1	465,102.0	501,715.0	260,340.0	**118,609.0**	140,669.0	1,418.2	852.0	489.3	461.8	**451.4**
	20	1	2	188,805.5	359,856.0	126,528.5	**11,088.5**	15,135.0	1,361.3	1,313.8	524.6	317.3	**195.0**
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
100	10	1	5	95,393.2	248,544.4	151,204.4	56,917.2	**30,007.2**	209.1	257.0	168.9	98.0	**63.5**
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
	20	1	0	-	-	-	-	-	-	-	-	-	-
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
125	10	1	5	**49,518.0**	312,285.4	254,152.8	310,771.6	206,355.0	**112.0**	237.6	225.3	244.7	184.6
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
	20	1	0	-	-	-	-	-	-	-	-	-	-
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
Average				213,088.8	328,909.0	163,642.2	88,897.0	**71,401.4**	828.8	696.8	369.9	218.4	**170.1**

Table C.10: Nodes and time comparison for instances solved with $t = 0.3$

				Av. Nodes Explore					Av. Time (sec)				
n	m	Ω	#	CPLEX	BW	BWL	BG	BGL	CPLEX	BW	BWL	BG	BGL
75	10	1	5	1,749.0	4,934.2	1,579.0	**606.6**	627.8	**4.7**	11.6	9.5	33.7	20.0
		3	5	105,041.6	316,139.6	88,043.4	3,903.8	**2,959.4**	249.8	422.3	108.3	137.5	**91.0**
		5	4	106,289.8	243,311.0	47,106.5	2,933.0	**1,837.0**	214.8	268.4	81.4	118.6	**57.6**
	20	1	5	46,164.6	84,968.0	54,357.6	6,022.0	**4,415.4**	236.1	349.8	206.0	158.5	**82.3**
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
100	10	1	5	4,668.2	12,624.4	12,106.0	**2,334.0**	3,958.8	**11.0**	20.8	21.7	31.4	24.8
		3	5	98,319.6	234,446.4	245,079.2	43,458.2	**33,129.8**	245.3	373.8	379.2	127.6	**91.3**
		5	5	325,560.8	742,981.0	629,685.2	**137,986.0**	164,530.4	1,081.2	1,147.9	1,441.6	**388.6**	432.2
	20	1	3	73,317.0	103,117.7	112,111.3	48,466.3	**43,590.0**	550.6	307.5	337.8	209.8	**178.7**
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
125	10	1	5	**620.6**	7,116.8	5,396.6	6,666.6	3,225.8	**2.5**	19.2	18.6	23.6	18.8
		3	5	**115,889.0**	291,387.6	303,281.4	137,235.4	129,738.6	301.2	472.1	446.9	237.0	**195.3**
		5	5	**244,464.4**	408,045.8	394,664.0	299,365.8	258,533.4	874.4	645.0	598.8	626.4	**416.9**
	20	1	4	**156,326.3**	674,932.5	420,752.8	385,045.5	253,781.3	**1,279.3**	1,992.8	1,284.0	1,329.8	1,361.9
		3	0	-	-	-	-	-	-	-	-	-	-
		5	0	-	-	-	-	-	-	-	-	-	-
Average				106,534.2	260,333.7	192,846.9	89,501.9	**75,027.3**	420.9	502.6	411.2	285.2	**247.6**

Bibliography

S. Ahmed and A. Shapiro. Solving chance-constrained stochastic programs via sampling and integer programming. In *State-of-the-art decision-making tools in the information-intensive age*, pages 261–269. Informs, 2008.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., USA, 1993. ISBN 013617549X.

S. B. Akers. Binary decision diagrams. *IEEE Transactions on computers*, C-27(6):509–516, 1978.

J. Amilhastre, H. Fargier, A. Niveau, and C. Pralet. Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. *International Journal on Artificial Intelligence Tools*, 23(04):1460015, 2014.

H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *Proceedings of the Principles and Practice of Constraint Programming*, pages 118–132. Springer, 2007.

C. Artigues. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154 – 159, 2017.

N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, 17(1): 61–84, 2000.

A. Atamtürk and A. Bhardwaj. Network design with probabilistic capacities. *Networks*, 71(1):16–30, 2018.

A. Atamtürk and V. Narayanan. The submodular knapsack polytope. *Discrete Optimization*, 6(4): 333–344, 2009.

A. Atamtürk and V. Narayanan. Conic mixed-integer rounding cuts. *Mathematical programming*, 122 (1):1–20, 2010.

A. Atamtürk, L. F. Muller, and D. Pisinger. Separation and extension of cover inequalities for conic quadratic knapsack constraints with generalized upper bounds. *INFORMS Journal on Computing*, 25 (3):420–431, 2013.

R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal methods in system design*, 10(2-3):171–206, 1997.

E. Balas. Facets of the knapsack polytope. *Mathematical programming*, 8(1):146–164, 1975.

E. Balas. *Disjunctive Programming*. Springer, 2018.

E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming*, 58(1-3):295–324, 1993.

E. Balas, M. Fischetti, and W. R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68(1):241–265, 1995.

E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.

R. Baldacci, A. Mingozzi, and R. Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(3):356–371, 2012.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

J. Beasley and B. Cao. A dynamic programming based algorithm for the crew scheduling problem. *Computers & Operations Research*, 25(78):567 – 582, 1998.

J. E. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer. BDDs in a branch and cut framework. In *Proceedings of the International Workshop on Experimental and Efficient Algorithms*, pages 452–463. Springer, 2005.

M. Behle. Binary decision diagrams and integer programming. *Ph.D. Thesis*, 2007.

M. Behle. On threshold BDDs and the optimal variable ordering problem. *Journal of Combinatorial Optimization*, 16(2):107–118, 2008.

M. Behle and F. Eisenbrand. 0/1 vertex and facet enumeration with BDDs. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 158–165. SIAM, 2007.

J. Benders. Partitioning procedures for solving mixed-variables programming problems . 1962.

D. Bergman and A. A. Cire. Decomposition based on decision diagrams. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 45–54. Springer, 2016a.

D. Bergman and A. A. Cire. Multiobjective optimization by decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 86–95. Springer, 2016b.

D. Bergman and A. A. Cire. Theoretical insights and algorithmic tools for decision diagram-based optimization. *Constraints*, 21(4):533–556, 2016c.

D. Bergman and A. A. Cire. On finding the optimal BDD relaxation. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 41–50. Springer, 2017.

D. Bergman and A. A. Cire. Discrete nonlinear optimization by state-space decompositions. *Management Science*, 64(10):4700–4720, 2018.

D. Bergman and L. Lozano. Decision diagram decomposition for quadratically constrained binary optimization. *INFORMS Journal on Computing*, 2020.

D. Bergman, W.-J. van Hoeve, and J. N. Hooker. Manipulating MDD relaxations for combinatorial optimization. In *Proceedings of the International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 20–35. Springer, 2011.

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. In *Proceedings of the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 34–49. Springer, 2012.

D. Bergman, A. A. Cire, W.-J. v. Hoeve, and J. N. Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014a.

D. Bergman, A. A. Cire, A. Sabharwal, H. Samulowitz, V. Saraswat, and W.-J. van Hoeve. Parallel combinatorial optimization with decision diagrams. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 351–367. Springer, 2014b.

D. Bergman, A. A. Cire, and W. van Hoeve. MDD propagation for sequence constraints. *Journal of Artificial Intelligence Research*, 50:697–722, 2014c.

D. Bergman, A. A. Cire, W.-J. van Hoeve, and T. Yunes. BDD-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, 2014d.

D. Bergman, A. A. Cire, and W.-J. van Hoeve. Improved constraint propagation via lagrangian decomposition. In *International Conference on Principles and Practice of Constraint Programming*, pages 30–38. Springer, 2015a.

D. Bergman, A. A. Cire, and W.-J. van Hoeve. Lagrangian bounds from decision diagrams. *Constraints*, 20(3):346–361, 2015b.

D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. *Decision Diagrams for Optimization*. Springer International Publishing, 1 edition, 2016a. ISBN 978-3-319-42849-9.

D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016b.

D. Bergman, M. Bodur, C. Cardonha, and A. A. Cire. Network models for multiobjective discrete optimization. *arXiv preprint arXiv:1802.08637*, 2018.

D. Bergman, C. Cardonha, and S. Mehrani. Binary decision diagrams for bin packing with minimum color fragmentation. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 57–66. Springer, 2019.

D. P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 2017.

C. Betz and M. Helmert. Planning with h^+ in theory and practice. In *Proceedings of the Annual German Conference on Artificial Intelligence*, pages 9–16. Springer, 2009.

A. Bhardwaj. *Binary conic quadratic knapsacks*. PhD thesis, UC Berkeley, 2015.

R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. Mixed-integer programming: A progress report. In *The sharpest cut: the impact of Manfred Padberg and his work*, pages 309–325. SIAM, 2004.

A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90 (1-2):281–300, 1997.

B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on computers*, 45(9):993–1002, 1996.

B. Bonet and J. Castillo. A complete algorithm for generating landmarks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 315–318, 2011.

B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning*, pages 360–372. Springer, 1999.

B. Bonet and M. Helmert. Strengthening landmark heuristics via hitting sets. In *Proceedings of the European Conference on Artificial Intelligence*, pages 329–334, 2010.

K. E. Booth, T. T. Tran, G. Nejat, and J. C. Beck. Mixed-integer and constraint programming techniques for mobile robot task planning. *IEEE Robotics and Automation Letters*, 1(1):500–507, 2016.

R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 100(8):677–691, 1986.

R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69 (1-2):165–204, 1994.

Q. Cappart, E. Goutierre, D. Bergman, and L.-M. Rousseau. Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1443–1451, 2019.

B. Cardoen, E. Demeulemeester, and J. Beliën. Operating room planning and scheduling: A literature review. *European journal of operational research*, 201(3):921–932, 2010.

W. M. Carlyle, J. O. Royset, and R. Kevin Wood. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. *Networks: An International Journal*, 52(4):256–270, 2008.

M. P. Castro, C. Piacentini, A. A. Cire, and J. C. Beck. Relaxed decision diagrams for cost-optimal classical planning. In *Proceedings of the Workshop on Heuristics and Search for Domain-Independent Planning*, pages 50–58, 2018.

M. P. Castro, C. Piacentini, A. A. Cire, and J. C. Beck. Relaxed BDDs: An admissible heuristic for delete-free planning based on a discrete relaxation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 77–85, 2019.

M. P. Castro, A. A. Cire, and J. C. Beck. An MDD-based Lagrangian approach to the multi-commodity pickup-and-delivery TSP. *INFORMS Journal on Computing*, 32(2):263–278, 2020a.

M. P. Castro, A. A. Cire, and J. C. Beck. A combinatorial cut-and-lift procedure with an application to 0-1 second-order conic programming. *Mathematical Programming, Series B (Under Review)*, 2020b.

M. P. Castro, C. Piacentini, A. A. Ciré, and J. C. Beck. Solving delete free planning with relaxed decision diagram based heuristics. *Journal of Artificial Intelligence Research*, 67:607–651, 2020c.

K. C. Cheng and R. H. Yap. Maintaining generalized arc consistency on ad hoc r-ary constraints. In *International conference on principles and practice of constraint programming*, pages 509–523. Springer, 2008.

K. C. Cheng and R. H. Yap. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2):265–304, 2010.

A. A. Ciré and J. N. Hooker. The separation problem for binary decision diagrams. In *ISAIM*, 2014.

A. A. Cire and W. J. van Hoeve. MDD propagation for disjunctive scheduling. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 11–19, 2012.

A. A. Cire and W.-J. van Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61(6):1411–1428, 2013.

A. A. Cire, A. Diamant, T. Yunes, and A. Carrasco. A network-based formulation for scheduling clinical rotations. *Production and Operations Management*, 28(5):1186–1205, 2019.

M. C. Cohen, P. W. Keller, V. Mirrokni, and M. Zadimoghaddam. Overcommitment in cloud services: Bin packing with chance constraints. *Management Science*, 65(7):3255–3271, 2019.

M. Conforti, G. Cornujols, and G. Zambelli. *Integer Programming*. Springer International Publishing, 2014.

J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Ann.Oper.Res.*, 153 (1):29–46, 2007.

J.-F. Cordeau, G. Laporte, J.-Y. Potvin, and M. W. Savelsbergh. Transportation on demand. In *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 429 – 466. Elsevier, 2007.

G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1): 3–44, 2008.

A. B. Corrêa, F. Pommerening, and G. Francès. Relaxed decision diagrams for delete-free planning. In *Proceedings of the Workshop on Constraints and AI Planning*, pages 1–2, 2018.

S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.

D. Davarnia and W.-J. van Hoeve. Outer approximation for integer nonlinear programs via decision diagrams. *Mathematical Programming*, Feb 2020. ISSN 1436-4646.

D. de Uña, G. Gange, P. Schachte, and P. J. Stuckey. Compiling CP subproblems to MDDs and d-DNNFs. *Constraints*, 24(1):56–93, 2019.

E. W. Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1): 269–271, 1959.

I. Dumitrescu, S. Ropke, J.-F. Cordeau, and G. Laporte. The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2): 269, 2008.

S. Edelkamp, P. Kissmann, and Á. Torralba. Symbolic A* search with pattern databases and the merge-and-shrink abstraction. In *Proceedings of the European Conference on Artificial Intelligence*, pages 306–311, 2012.

D. Espinoza, R. Fukasawa, and M. Goycoolea. Lifting, tilting and fractional programming revisited. *Operations research letters*, 38(6):559–563, 2010.

D. G. Espinoza. *On linear programming, integer programming and cutting planes*. PhD thesis, Georgia Institute of Technology, 2006.

R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management science*, 50(12):1861–1871, 2004.

A. Frangioni. Generalized bundle methods. *SIAM Journal on Optimization*, 13(1):117–156, 2002.

N. Frohner and G. R. Raidl. Towards improving merging heuristics for binary decision diagrams. In *International Conference on Learning and Intelligent Optimization*, pages 30–45. Springer, 2019.

G. Gange, P. J. Stuckey, and R. Szymanek. MDD propagators with explanation. *Constraints*, 16(4):407, 2011.

G. Gange, P. J. Stuckey, and P. Van Hentenryck. Explaining propagators for edge-valued decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 340–355. Springer, 2013.

Á. García-Olaya, S. Jiménez, and C. Linares López. The 2011 International planning competition. Technical Report, 2011.

A. Gefen and R. I. Brafman. The minimal seed set problem. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 319–322, 2011.

F. Geißer, T. Keller, and R. Mattmüller. Abstractions for planning with state-dependent action costs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 140–148, 2016.

A. M. Geoffrion. Lagrangian relaxation and its uses in integer programming. *Mathematical Programming Study 2*, pages 82–114, 1974.

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

R. E. Gomory. Some polyhedra related to combinatorial problems. *Linear Algebra and its Applications*, 2(4):451 – 558, 1969. ISSN 0024-3795.

J. E. González, A. A. Cire, A. Lodi, and L.-M. Rousseau. Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem. *Constraints*, pages 1–24, 2020.

L. Gouveia and P. Pesneau. On extended formulations for the precedence constrained asymmetric traveling salesman problem. *Networks*, 48(2):77–89, 2006.

L. Gouveia and M. Ruthmair. Load-dependent and precedence-based models for pickup and delivery problems. *Computers & Operations Research*, 63:56–71, 2015.

M. Guignard and S. Kim. Lagrangian decomposition: A model yielding stronger Lagrangian bounds. *Mathematical programming*, 39(2):215–228, 1987.

C. Guo, M. Bodur, D. M. Aleman, and D. R. Urbach. Logic-based benders decomposition and binary decision diagram based approaches for stochastic distributed operating room scheduling. *arXiv preprint arXiv:1907.13265*, 2019.

L. Gurobi Optimization. Gurobi optimizer reference manual, 2020.

T. Hadzic and J. Hooker. Postoptimality analysis for integer programming using binary decision diagrams. In *GICOLAG Workshop (Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry), Vienna. Technical report, Carnegie Mellon University*, 2006.

T. Hadzic and J. N. Hooker. Cost-bounded binary decision diagrams for 0-1 programming. In *Proceedings of the International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 84–98. Springer, 2007.

T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. *Proceedings of the International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems*, pages 131–138, 2004.

T. Hadzic, J. N. Hooker, B. OSullivan, and P. Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 448–462. Springer, 2008a.

T. Hadzic, J. N. Hooker, and P. Tiedemann. Propagating separable equalities in an MDD store. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 318–322. Springer, 2008b.

T. Hadzic, E. O'Mahony, B. O'Sullivan, and M. Sellmann. Enhanced inference for the market split problem. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 716–723. IEEE, 2009.

P. L. Hammer, E. L. Johnson, and U. N. Peled. Facet of regular 0–1 polytopes. *Mathematical Programming*, 8(1):179–206, 1975.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

P. Haslum, B. Bonet, H. Geffner, et al. New admissible heuristics for domain-independent planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 5, pages 9–13, 2005.

P. Haslum, J. K. Slaney, and S. Thiébaux. Minimal landmarks for optimal delete-free planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 353–357, 2012.

U.-U. Haus and C. Michini. Compact representations of all members of an independence system. *Annals of Mathematics and Artificial Intelligence*, 79(1-3):145–162, 2017.

U.-U. Haus, C. Michini, and M. Laumanns. Scenario aggregation using binary decision diagrams for stochastic programs with endogenous uncertainty. *arXiv preprint arXiv:1701.04055*, 2017.

M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1):196–210, 1962.

M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical programming*, 1(1):6–25, 1971.

M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: what's the difference anyway? In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 162–169, 2009.

M. Helmert, G. Röger, et al. How good is almost perfect?. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 8, pages 944–949, 2008.

I. T. Hernádvölgyi. Solving the sequential ordering problem with automatically generated lower bounds. In *Operations Research Proceedings 2003*, pages 355–362. Springer, 2004.

H. Hernández-Pérez and J.-J. Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139, 2004.

H. Hernández-Pérez and J.-J. Salazar-González. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196(3):987–995, 2009.

S. Hoda, W.-J. Van Hoeve, and J. N. Hooker. A systematic approach to MDD-based constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 266–280. Springer, 2010.

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

C. Holland, J. Levis, R. Nuggehalli, B. Santilli, and J. Winters. UPS optimizes delivery routes. *Interfaces*, 47(1):8 – 23, 2017.

J. N. Hooker. Decision diagrams and dynamic programming. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 94–110. Springer, 2013.

J. N. Hooker. Job sequencing bounds from decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 565–578. Springer, 2017.

J. N. Hooker. Improved job sequencing bounds from decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 268–283. Springer, 2019.

G. Horn, J. Maschler, G. Raidl, and E. Rönnberg. A*-based construction of decision diagrams for a prize-collecting scheduling problem. *Algorithms and Complexity Group, TU Wien, Technical Report AC-TR-18-011, 2018a. Copyright The publishers will keep this document online on the Internet-or its possible replacement–for a period of*, 25, 2018.

M. Horn and G. R. Raidl. Decision diagram based limited discrepancy search for a job sequencing problem. *Computer Aided Systems Theory-EUROCAST*, 2019.

A. Hosseininasab and W.-J. van Hoeve. Exact multiple sequence alignment by synchronized decision diagrams. *INFORMS Journal on Computing*, 2019.

IBM. *ILOG CPLEX Studio 12.9 Manual*, 2019.

T. Imai and A. Fukunaga. A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. In *Proceedings of the European Conference on Artificial Intelligence*, pages 459–464, 2014.

T. Imai and A. Fukunaga. On a practical, integer-linear programming model for delete-free tasks and its use as a heuristic for cost-optimal planning. *Journal of Artificial Intelligence Research*, 54:631–677, 2015.

S. Joung and S. Park. Lifting of probabilistic cover inequalities. *Operations Research Letters*, 45(5): 513–518, 2017.

R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

E. Karpas and C. Domshlak. Optimal search with inadmissible heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2012.

B. Kell and W.-J. Van Hoeve. An MDD approach to multidimensional bin packing. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 128–143. Springer, 2013.

B. Kell, A. Sabharwal, and W.-J. van Hoeve. BDD-guided clause generation. In L. Michel, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 215–230, Cham, 2015. Springer International Publishing.

T. Keller, F. Pommerening, J. Seipp, F. Geißer, and R. Mattmüller. State-dependent cost partitionings for Cartesian abstractions in classical planning. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 3161–3169, 2016.

E. Keyder, S. Richter, and M. Helmert. Sound and complete landmarks for And/Or graphs. In *Proceedings of the European Conference on Artificial Intelligence*, volume 215, pages 335–340, 2010.

J. Kinable, A. A. Cire, and W.-J. van Hoeve. Hybrid optimization methods for time-dependent sequencing problems. *European Journal of Operational Research*, 259(3):887–897, 2017.

D. Kowalczyk and R. Leus. A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching. *INFORMS Journal on Computing*, 30(4):768–782, 2018.

A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, 3(10):497–520, 1960.

A. L. Latour, B. Babaki, A. Dries, A. Kimmig, G. Van den Broeck, and S. Nijssen. Combining stochastic constraint optimization and probabilistic programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 495–511. Springer, 2017.

A. L. D. Latour, B. Babaki, and S. Nijssen. Stochastic constraint propagation for mining probabilistic networks. In *International Joint Conference on Artificial Intelligence*, 2019.

C.-Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

C. Lemaréchal. *An extension of davidon methods to non differentiable problems*, pages 95–109. Springer Berlin Heidelberg, Berlin, Heidelberg, 1975.

A. N. Letchford and J.-J. Salazar-González. Stronger multi-commodity flow formulations of the (capacitated) sequential ordering problem. *European Journal of Operational Research*, 251(1):74–84, 2016.

C. Liu, D. M. Aleman, and J. C. Beck. The senior transportation problem. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 412–428, 2018.

R. Liu, X. Xie, V. Augusto, and C. Rodriguez. Heuristic algorithms for a vehicle routing problem with simultaneous delivery and pickup and time windows in home health care. *European Journal of Operational Research*, 230(3):475 – 486, 2013.

M. Löbbing and I. Wegener. The number of knight's tours equals 33,439,123,484,294counting with binary decision diagrams. *the electronic journal of combinatorics*, 3(1):R5, 1996.

M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear algebra and its applications*, 284(1-3):193–228, 1998.

A. Lodi. Mixed integer programming computation. In *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer, 2010.

A. Lodi, M. Tanneau, and J. P. Vielma. Disjunctive cuts for mixed-integer conic optimization. *arXiv preprint arXiv:1912.03166*, 2019.

Q. Louveaux and L. A. Wolsey. Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1 (3):173–207, Oct 2003.

L. Lozano and J. C. Smith. A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs. *Mathematical Programming*, pages 1–24, 2018.

L. Lozano, D. Bergman, and J. C. Smith. On the consistent path problem. *Optimization Online e-prints*, 2018.

J. Maschler and G. R. Raidl. Multivalued decision diagrams for a prize-collecting sequencing problem. In *Proceedings of PATAT*, pages 375–397, 2018.

S.-i. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th international Design Automation Conference*, pages 272–277, 1993.

D. R. Morrison, E. C. Sewell, and S. H. Jacobson. Solving the pricing problem in a branch-and-price algorithm for graph coloring using zero-suppressed binary decision diagrams. *INFORMS Journal on Computing*, 28(1):67–82, 2016.

S. Nadarajah and A. A. Cire. Network-based approximate linear programming for discrete optimization. *Available at SSRN 3081898*, 2017.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, USA, 1988. ISBN 047182819X.

M. Nishino, N. Yasuda, S.-i. Minato, and M. Nagata. BDD-constrained search: A unified approach to constrained shortest path problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

R. J. ONeil and K. Hoffman. Decision diagrams for solving traveling salesman problems with pickup and delivery in real time. *Operations Research Letters*, 47(3):197–201, 2019.

M. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical programming*, 5(1): 199–215, 1973.

M. W. Padberg. A note on zero-one programming. *Operations Research*, 23(4):833–837, 1975.

S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1):21–51, 2008.

G. Perez and J.-C. Régin. Improving GAC-4 for table and MDD constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 606–621. Springer, 2014.

G. Perez and J.-C. Régin. Efficient operations on MDDs for building constraint programming models. In *International Joint Conference on Artificial Intelligence*, 2015a.

G. Perez and J.-C. Régin. Relations between MDDs and tuples and dynamic modifications of MDDs based constraints. *arXiv preprint arXiv:1505.02552*, 2015b.

G. Perez and J.-C. Régin. Constructions and in-place operations for MDDs based constraints. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 279–293. Springer, 2016.

G. Perez and J.-C. Régin. MDDs are efficient modeling tools: An application to some statistical constraints. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 30–40. Springer, 2017a.

G. Perez and J.-C. Régin. MDDs: sampling and probability constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 226–242. Springer, 2017b.

G. Perez and J.-C. Régin. Soft and cost MDD propagators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017c.

G. Perez and J.-C. Régin. Parallel algorithms for operations on multi-valued decision diagrams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

M. Perregaard and E. Balas. Generating cuts from multiple-term disjunctions. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 348–360. Springer, 2001.

C. Piacentini, M. P. Castro, A. A. Cire, and J. C. Beck. Linear and integer programming-based heuristics for cost-optimal numeric planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6254–6261, 2018.

N. Ploskas, C. Laughman, A. U. Raghunathan, and N. V. Sahinidis. Heat exchanger circuitry design by decision diagrams. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 461–471. Springer, 2019.

F. Pommerening and M. Helmert. Optimal planning for delete-free tasks with incremental LM-cut. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 363–367, 2012.

F. Pommerening, G. Röger, M. Helmert, and B. Bonet. LP-based heuristics for cost-optimal planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 226–234, 2014.

F. Pommerening, G. Röger, M. Helmert, H. Cambazard, L.-M. Rousseau, and D. Salvagnin. Lagrangian decomposition for optimal cost partitioning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 338–347, 2019.

J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the European Conference on Planning*, pages 174–182, 2001.

A. U. Raghunathan, D. Bergman, J. Hooker, T. Serra, and S. Kobori. Seamless multimodal transportation scheduling. *arXiv preprint arXiv:1807.09676*, 2018.

I. Rodríguez-Martín and J. J. Salazar-González. A hybrid heuristic approach for the multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *Journal of Heuristics*, 18(6):849–867, 2012.

M. Römer, A. A. Cire, and L.-M. Rousseau. A local search framework for compiling relaxed decision diagrams. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 512–520. Springer, 2018.

F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.

P. Roy, G. Perez, J.-C. Régin, A. Papadopoulos, F. Pachet, and M. Marchini. Enforcing structure on temporal sequences: the allen constraint. In *International conference on principles and practice of constraint programming*, pages 786–801. Springer, 2016.

S. Sanner and D. McAllester. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *International Joint Conference on Artificial Intelligence*, volume 2005, pages 1384–1390, 2005.

M. W. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation science*, 29 (1):17–29, 1995.

T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools*, 3(02):187–207, 1994.

A. Şen, A. Atamtürk, and P. Kaminsky. A conic integer optimization approach to the constrained assortment problem under the mixed multinomial logit model. *Operations Research*, 66(4):994–1003, 2018.

T. Serra and J. Hooker. Compact representation of near-optimal integer programming solutions. *Mathematical Programming*, pages 1–34, 2019.

T. Serra, A. U. Raghunathan, D. Bergman, J. Hooker, and S. Kobori. Last-mile scheduling under uncertainty. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 519–528. Springer, 2019.

S. Singh and I. Chana. A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of grid computing*, 14(2):217–264, 2016.

A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *1990 IEEE International conference on computer-aided design. Digest of Technical Papers*, pages 92–95. IEEE, 1990.

S. Subbarayan. Efficient reasoning for nogoods in constraint solvers with BDDs. In *International Symposium on Practical Aspects of Declarative Languages*, pages 53–67. Springer, 2008.

C. Tjandraatmadja. Decision diagram relaxations for integer programming. 2018.

C. Tjandraatmadja and W.-J. van Hoeve. Target cuts from relaxed decision diagrams. *INFORMS Journal on Computing*, 31(2):285–301, 2019.

Á. Torralba, C. Linares López, and D. Borrajo. Symbolic merge-and-shrink for cost-optimal planning. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 2394–2400, 2013.

Á. Torralba, C. Linares López, and D. Borrajo. Abstraction heuristics for symbolic bidirectional search. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 3272–3278, 2016.

P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

C. Van de Panne and W. Popp. Minimum-cost cattle feed under probabilistic protein constraints. *Management Science*, 9(3):405–430, 1963.

P. van den Bogaerdt and M. de Weerdt. Multi-machine scheduling lower bounds using decision diagrams. *Operations Research Letters*, 46(6):616–621, 2018.

W.-J. van Hoeve. Graph coloring lower bounds from decision diagrams. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 405–418. Springer, 2020.

H. Verhaeghe, C. Lecoutre, and P. Schaus. Compact-MDD: Efficiently filtering (s) MDD constraints with reversible sparse bit-sets. In *International Joint Conference on Artificial Intelligence*, pages 1383–1389, 2018.

H. Verhaeghe, C. Lecoutre, and P. Schaus. Extending compact-diagram to basic smart multi-valued variable diagrams. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 581–598. Springer, 2019.

J. P. Vielma, S. Ahmed, and G. L. Nemhauser. A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs. *INFORMS Journal on Computing*, 20(3):438–450, 2008.

J. P. Vielma, I. Dunning, J. Huchette, and M. Lubin. Extended formulations in mixed integer conic quadratic programming. *Mathematical Programming Computation*, 9(3):369–418, 2017.

I. Wegener. *Branching programs and binary decision diagrams: theory and applications*, volume 4. SIAM, 2000.

L. A. Wolsey. Technical note - facets and strong valid inequalities for integer programs. *Operations Research*, 24(2):367–372, 1976. doi: 10.1287/opre.24.2.367.

L. A. Wolsey and G. L. Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

Y. Xue and W.-J. van Hoeve. Embedding decision diagrams into generative adversarial networks. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 616–632. Springer, 2019.

L. Zhu and R. Givan. Landmark extraction via planning graph propagation. In *Proceedings of the International Conference on Automated Planning and Scheduling, Doctoral Consortium*, pages 156–160, 2003.