On the Consistent Path Problem

Leonardo Lozano[1], David Bergman[†2], and J. Cole Smith[‡3]

[1]Operations, Business Analytics & Information Systems, University of Cincinnati
[2]Operations and Information Management, University of Connecticut
[3]Industrial Engineering, Clemson University

March 24, 2018

Abstract

The application of decision diagrams in combinatorial optimization has proliferated in the last decade. In recent years, authors have begun to investigate how to utilize not one, but a set of diagrams, to model constraints and objective function terms. Optimizing over a collection of decision diagrams, the problem we refer to as the consistent path problem (CPP), can be addressed by associating a network-flow model with each decision diagram, jointly linked through channeling constraints. A direct application of integer programming to the ensuing model has already shown to result in algorithms that provide orders-of-magnitude performance gains over classical methods. Lacking, however, is a careful study of dedicated solution methods designed to solve the CPP. This paper provides a detailed study of the CPP, including a discussion on complexity results and a complete polyhedral analysis. We propose a cut-generation algorithm which, under a structured ordering property, finds a cut, if one exists, through an application of the classical maximum flow problem, albeit in an exponentially sized network. We use this procedure to fuel a cutting-plane algorithm that is applied to unconstrained binary cubic optimization and a variant of the market split problem, resulting in a state-of-the-art algorithm for both.

Keywords. Networks/graphs:Flow algorithms; Networks/graphs: Generalized networks; Programming:Integer:Algorithms

1 Introduction

In this paper we study the *consistent path problem* (CPP). The CPP arises in the context of decision diagram-based optimization (Bergman et al. 2011, 2016b,a), where *binary decision diagrams* (BDDs) are used to represent the set of feasible solutions to a binary optimization problem, along with the objective function evaluation of those solutions.

The literature on BDD-based discrete optimization has focused on single-BDD solution approaches (Hoda et al. 2010, Hadžić and Hooker 2006, 2007, Hadžić et al. 2008, Bergman et al. 2011, 2016b,a). In recent years, however, authors have begun to study how a collection of BDDs can be used, concurrently and more compactly, to represent a problem. This idea was introduced by

*leolozano@uc.edu
†david.bergman@uconn.edu
‡jcsmith@clemson.edu

Bergman and Ciré (2016), who show that using multiple BDDs to represent a problem can yield compact models that improve upon single-BDD optimization techniques for the maximum independent set and unconstrained binary quadratic optimization problems. This study was extended to three classes of non-linear discrete optimization problems by Bergman and Ciré (2018), where state-space decomposition models were formulated to construct a collection of BDDs over which optimization is concurrently performed. We define the CPP as the problem of optimizing over a collection of BDDs.

The use of multiple decision diagrams to represent the feasible set for discrete optimization problems also arises in *constraint programming*. Certain global constraints can be modeled as decision diagrams, and the ensuing model reduces to finding a single common path among the set of decision diagrams (see, for example, Perez and Régin (2015)). This idea was also used to model and solve market split problems (Hadžić et al. 2009).

Despite the generality of this approach—namely, to transform a combinatorial optimization problem into a collection of BDDs—research on algorithms for optimizing over a collection of BDDs is lacking. For a single BDD, optimizing over all solutions reduces to a simple shortest or longest path problem in a directed acyclic graph, and hence takes time linear in the size of the BDD. This can also be viewed as an integer programming problem in a lifted space, where each BDD arc corresponds to a flow variable. It is well known that this network-flow reformulation yields a convex-hull relaxation (in a lifted space), and that the network-flow reformulation can be solved efficiently.

Contrastingly, we prove in this paper that the CPP is NP-hard, even if there are only two BDDs in the collection defining the CPP instance. However, if the variables associated with the layers of the BDDs have a certain common ordering property (which we later formally define as being *order associated* (OA)), we prove that optimizing over the collection of BDDs can be implemented in polynomial time with respect to the size of the BDDs of the collection, if the number of BDDs is considered constant.

This paper also presents a cutting-plane algorithm for the CPP where the component BDDs are OA. In particular, we propose to use the network-flow reformulation of the CPP introduced by Bergman and Ciré (2016, 2018) to formulate the problem as a binary programming problem, where the variables that are common among the BDDs are connected through linear channeling constraints. The ensuing optimization problem is computationally challenging. In order to efficiently solve this reformulation, we present a general class of cutting planes that can be used to tighten the LP relaxation of the CPP for the case in which the component BDDs are OA. Namely, for any collection of BDDs that are OA, we show that one can separate a fractional point of the network-flow reformulation by solving a maximum flow problem in the *intersection BDD* of all the component BDDs. This therefore characterizes, in its entirety, the convex hull of feasible points in the lifted network-flow reformulation.

This cutting-plane procedure requires exponential time and memory, since the intersection BDD grows exponentially with the number of BDDs. In order to cope with the prohibitive growth, we propose a cutting-plane algorithm that computes intersection BDDs for subsets of the component BDDs and generates cutting planes based on these. A thorough experimental evaluation indicates that this solution approach can provide significant computational speedups through the application to two classes of optimization problems—the *unconstrained binary cubic optimization problem* (UBCP) and the *market multisplit problem* (MMP), a generalization of the market split problem (Cornuéjols and Dawande 1999, Hadžić et al. 2009). For the UBCP, our results indicate that the network model reformulation without cutting planes requires longer computational times than standard integer programming (IP) models, but with the addition of the cutting planes described in this paper, the BDD-based model is orders-of-magnitude faster than a standard IP model. With

respect to the MMP, we show that the network-flow reformulation is already much faster than the natural IP model, and that the addition of our prescribed cutting planes can further improve algorithmic efficiency.

We note here that Becker et al. (2005) considers the similar, but distinct, problem of taking a fractional point in the original problem space, and finding a cutting plane in that space, over a single BDD. Becker et al. (2005) show that the separation problem they consider can be solved by a sequence of shortest-path problems with Lagrangean relaxation techniques. In this paper we do not seek cutting planes in the space of the original problem variables, but rather seek a cutting plane over the network-flow reformulation.

In summary, the main contributions of this paper are as follows:

1. A detailed analysis of the complexity of the CPP;
2. A description of the convex hull of integer points to the network-flow reformulation for the CPP if the component BDDs are OA; and
3. A cutting-plane algorithm for the CPP for the case in which the component BDDs are OA.

The remainder of the paper is organized as follows. Section 2 provides notation to be used throughout the paper, defines the problem classes studied, and formally defines the CPP. Section 3 presents complexity results concerning the CPP. Section 4 describes a cutting-plane algorithm for the CPP through the design of valid cuts for a network-flow reformulation. Section 5 reports on an experimental evaluation of the proposed cutting-plane algorithm on the UBCP and MMP. We conclude in Section 6 with ideas for future work.

2 Preliminaries and Definitions

This section provides a description of the notation we use throughout the paper, which is summarized in Table 1.

2.1 Binary Optimization Problems

For the purpose of this paper, a *binary optimization problem* (BOP) is a triple $P = \langle X, R, f \rangle$. Set $X = \{x_1, \ldots, x_n\}$ are variables that take only binary values, R is a set of *relations* that dictate when an assignment of 0/1 values to the variables in X is permissible (this is typically given as a set of linear constraints $Ax \leq b$, assuming X is ordered as a vector $x = (x_1, \ldots, x_n)$), and $f : \mathbb{B}^n \to \mathbb{R}$ is an objective function, which we will assume without loss of generality is to be maximized. A *solution* $\tilde{x}$ is any assignment of 0/1 values to the variables in X. A solution is *feasible* if it satisfies each relation in R. We let $\mathbf{Sol}(P)$ be the set of all feasible solutions. Vector $x^* \in \mathbf{Sol}(P)$ is an *optimal solution* if $f(x^*) \geq f(\hat{x})$ for all $\hat{x} \in \mathbf{Sol}(P)$. The goal is to find an optimal solution and the optimal value.

The approach studied in this paper is broadly applicable to binary (and, more generally, discrete) optimization problems. We focus on the application to the following two problem classes.

UBCP: The (unconstrained) *binary polynomial optimization* problem consists of n variables and can be formulated as follows:

$$\max_{x \in \mathbb{B}^n} \sum_{s \in 2^{[n]}} w_s \cdot \prod_{i \in s} x_i.$$

Every subset $s \in 2^{[n]}$ (where we define $[a] := \{1, \ldots, a\}$, and 2^Y as the *power set* of all subsets of Y) is associated with a value w_s which, if all variables in s are set to 1, contributes w_s to the objective

Table 1: Relevant definitions and notation

Symbol	Explanation
X	Set of variables $\{x_1, \ldots, x_n\}$
$f(x)$	Objective function corresponding to solution x
$\mathbf{Sol}(P)$	All feasible solutions to problem P
N	Set of nodes for a BDD
A	Set of arcs for a BDD
$\ell(u)$	Layer corresponding to node u
L_j	Set of all nodes contained in layer j
$t(a)$	Tail of arc a
$h(a)$	Head of arc a
$d(a)$	Domain of arc a
$w(a)$	Weight of arc a
$\alpha(u)$	One-arc leaving node u
$\beta(u)$	Zero-arc leaving node u
$\mathrm{var}(L_j)$	x-variable corresponding to layer j
$\mathrm{i}(L_j)$	index of x-variable corresponding to layer j
$X(B)$	Set of variables decided by BDD B
$\omega(B)$	Width of BDD B
$s(p)$	Set of all solutions $x \in \mathbb{B}^n$ corresponding to path p
Ω_B	Set of all paths from $\mathbf{r}$ to $\mathbf{t}$ in BDD B
$\mathcal{S}(B)$	Set of all solutions $x \in \mathbb{B}^n$ corresponding to a path in Ω_B
$p^B(x)$	Unique path in BDD B that corresponds to solution $x \in \mathbb{B}^n$
$\mathcal{B}$	A collection of BDDs $\{B_1, \ldots, B_K\}$
$\mathcal{Q}$	A set of consistent paths $\{q(B_1), \ldots, q(B_K)\}$
$x(\mathcal{Q})$	Solution x corresponding to $\mathcal{Q}$

value. The problem seeks to find an assignment that maximizes the sum of the values w_s over those sets s where all variables are 1. For this paper we discuss the application to *unconstrained binary cubic optimization* (UBCP), where $w_s = 0$ for all s with $|s| > 3$, but note that the ideas can be extended to this more general setting.

MMP: The MMP is an extension of the *market split problem* (Cornuéjols and Dawande 1999), which is inspired by the situation in which a company seeks to divide a set of retailers with demand for multiple products into two divisions. The goal is to assign the retailers to the divisions in such a way that the first division controls γ percent of the company's demand for each product and the second division controls $(1-\gamma)$ percent. If a perfect split is not attainable, then the objective is to minimize the sum of the deviations from the initial goal.

The MMP generalizes the market split problem by considering multiple divisions instead of two. Let d denote the number of divisions, r be the number of retailers, and p be the number of products. Let a_{ij} be the demand for product i in retailer j and b_i be the desired market share for product i among all divisions (assumed throughout this paper to be constant from division to division). Let x_{jq} be a binary variable that takes a value of 1 if retailer j is assigned to division q and let s_{iq} denote the deviation from the goal for product i in division q. We formulate the MMP

as:

$$\max \quad \sum_{i=1}^{p}\sum_{q=1}^{d}|s_{iq}| \tag{MMP}$$

$$\text{s.t.} \quad \sum_{j=1}^{r} a_{ij}x_{jq} + s_{iq} = b_i \qquad \forall i \in \{1,\dots,p\}, \forall q \in \{1,\dots,d\} \tag{1a}$$

$$\sum_{q=1}^{d} x_{jq} = 1 \qquad \forall j \in \{1,\dots,r\} \tag{1b}$$

$$x_{jq} \in \{0,1\} \qquad \forall j \in \{1,\dots,r\}, \forall q \in \{1,\dots,d\} \tag{1c}$$

$$s_{iq} \text{ unrestricted} \qquad \forall i \in \{1,\dots,p\}, \forall q \in \{1,\dots,d\}. \tag{1d}$$

The objective function minimizes the sum of deviations from the goal. Constraints (1a) define the s-variables while constraints (1b) ensure that each retailer is assigned to exactly one division.

2.2 Binary Decision Diagrams

A *binary decision diagram* (BDD) $B = (m, N, \ell, A, d, \text{var})$ for a binary optimization problem is a *layered-acyclic digraph* composed of node set N and arc set A. Every node $u \in N$ is associated with a *layer* $\ell(u) \in \{1,\dots,m+1\}$, and these $m+1$ non-empty layers partition the nodes in N. Denote $L_j := \{u \in N : \ell(u) = j\}$ as the set of nodes in layer j. For convenience, in this paper we simply refer to a set L_j as layer j of the BDD. Layer L_1 has cardinality one, with the singleton node $\mathbf{r}$ called the *root*. Layer L_{m+1} has at most two nodes—the *true terminal* $\mathbf{t}$ and the (optionally present) *false terminal* $\mathbf{f}$. Let $|B| = |N|$ denote the *size* of B. Let the *width* of a layer be the number of nodes in that layer, denoted by $\omega(L_j) := |L_j|$. The *width* of a BDD, $\omega(B)$, is the maximum width of any layer (i.e., $\omega(B) = \max_j \omega(L_j)$).

Each arc $a \in A$ is directed from node $t(a)$ to node $h(a)$, where $t(a), h(a) \in N$ are denoted as the *tail* and *head* of arc a. For now, we will assume that $\ell(h(a)) = \ell(t(a)) + 1$, so that each arc connects nodes in adjacent layers. Every node $u \notin \{\mathbf{t}, \mathbf{f}\}$ has (at most) two out-directed arcs denoted by $\alpha(u)$ and $\beta(u)$, distinguished by their *arc domains*. Arc $\alpha(u)$ has $d(\alpha(u)) = 1$ and arc $\beta(u)$ has $d(\beta(u)) = 0$. Any arc a with $d(a) = 1$ is referred to as a *one-arc* and any arc a with $d(a) = 0$ is referred to as a *zero-arc*. Additionally, each arc a has a *weight* $w(a)$.

When used to represent the set of solutions to a binary optimization problem P on variable set $X = \{x_1,\dots,x_n\}$ each of the first m layers $L_1,\dots,L_m$ is associated with a unique variable in X (note, $m \le n$) which is denoted by $\text{var}(L_j)$. The index of the x-variable $\text{var}(L_j)$ is denoted by $\text{i}(L_j)$. Let $X(B) \subseteq X$ be the set of variables that are associated with the layers in B, i.e., $X(B) = \cup_{j=1}^{m}\text{var}(L_j)$.

The association between variables and layers dictate a connection between the paths in B and the solutions to P. Namely, let $p = (a_1,\dots,a_m)$ be a path from $\mathbf{r}$ to $\mathbf{t}$ (i.e., $t(a_1) = \mathbf{r}, h(a_m) = \mathbf{t}$, and for $j = 1,\dots,m-1,\ h(a_j) = t(a_{j+1})$). We let $s(p) = \{x \in \mathbb{B}^n : \text{var}(L_j) = d(a_j),\ \forall a_j \in p\}$, so that $s(p)$ denotes the set of 0/1 assignments to the variables in X for which for each $j \in \{1,\dots,m\}$, the variable $\text{var}(L_j)$ associated with layer L_j is fixed to the value $d(a_j)$.

Let $\mathcal{S}(B) = \{s(p) : p \in \Omega_B\}$, where Ω_B is the set of all arc-directed $\mathbf{r}$-to-$\mathbf{t}$ paths in B. We define the *weight* of a path $w(p)$ to be the sum of the weights of the arcs in p: $w(p) = \sum_{j=1}^{m} w(a_j)$. Note that a solution $x \in \mathbb{B}^n$ corresponds to exactly one path in B, and we let $p^B(x)$ be that path; however, a path in B can correspond to many solutions.

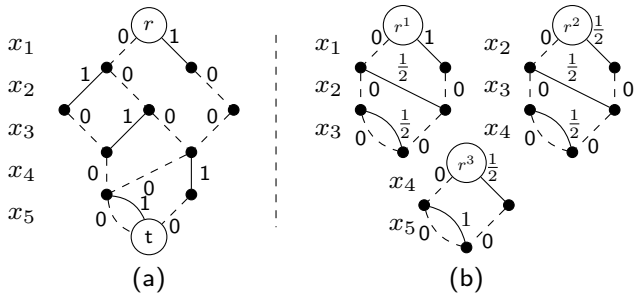

Figure 1: (a) Single BDD and (b) collection of BDDs for example BOP.

If (i) $\mathcal{S}(B) = \mathbf{Sol}(P)$ and (ii) for every $x \in \mathbf{Sol}(P)$, $f(x) = w\left(p^B(x)\right)$, we say that B is an *exact* BDD for P. For many combinatorial optimization problems the size of any exact BDD will be exponential in size. It is therefore of interest to represent the solution set of P as a collection of BDDs $\mathcal{B}$, where we say that $\mathcal{B}$ is an *exact BDD decomposition* of P if $\bigcap_{B\in\mathcal{B}} \mathcal{S}(B) = \mathbf{Sol}(P)$ and for every $x \in \mathbf{Sol}(P)$, $f(x) = \sum_{B\in\mathcal{B}} \left(w\left(p^B(x)\right)\right)$. Let $X(\mathcal{B}) = \cup_{B\in\mathcal{B}} X(B)$.

When it is necessary to distinguish between BDDs we will use a superscript B on elements of B (for example, the root node will be denoted by $\mathbf{r}^B$).

Finally, we formally define the OA property for a BDD collection. A BDD collection is said to satisfy the OA property if there exists an indexing of variables $x_1, \ldots, x_n$ such that $\mathsf{i}(L_j^B) < \mathsf{i}(L_{j+1}^B)$, $\forall B \in \mathcal{B}$ and for all $j = 1, \ldots, m^B$. As a consequence of this property, for each BDD $B \in \mathcal{B}$, an r^B–t^B path encounters variables in $X(B)$ corresponding to the layers of B in increasing order of the variable indices.

2.3 The Consistent Path Problem

Given a collection of BDDs $\mathcal{B} = \{B_1, \ldots, B_K\}$ and a set of variables X, with each BDD defined on variables $X(B_k) \subseteq X$ for $k = 1, \ldots, K$, a *consistent path collection* (CPC) is a set of paths $\mathcal{Q} = \{q(B_1), \ldots, q(B_K)\}$, one per BDD, for which $\cap_{k=1}^K s(q(B_k)) \neq \emptyset$. Note that if, for every variable $x_i \in X$, $x_i \in \cup_{k=1}^K X(B_k)$, then $|\cap_{k=1}^K s(q(B_k))| \in \{0, 1\}$ and so a CPC $\mathcal{Q}$ defines at most one setting of each variable in X. We denote that single solution as $x(\mathcal{Q})$. If $\mathcal{B}$ is an exact BDD decomposition of P then for every CPC $\mathcal{Q}$, $f(x(\mathcal{Q})) = \sum_{k=1}^K \sum_{a\in q(B_k)} w(a)$. The *consistent path problem* (CPP) seeks a CPC of largest total weight.

To illustrate, consider the BOP $\mathcal{P}'$ (adapted from Bergman and Ciré (2016)) defined by $\max f(x) = \sum_{i=1}^5 x_i$ subject to $x_1 + x_2 \leq 1, x_1 + x_3 \leq 1, x_2 + x_3 \leq 1, x_2 + x_4 \leq 1, x_3 + x_4 \leq 1$, and $x_4 + x_5 \leq 1$. The optimal solution value is 2. Figure 1(a) (where the false terminal node is omitted) depicts a single BDD for this BOP. The variables corresponding to each layer appear on the left of the BDD and dashed/solid arcs correspond to setting variables on that layer to 0/1. For a single BDD, the problem of optimizing over the solutions reduces to a longest-path computation which, since a BDD is a directed acyclic graph, can be done in time proportional to the number of nodes.

Figure 1b displays a collection of BDDs, each of which encodes the information required by the respective constraints in the system. Observe that this collection satisfies the OA property. An example of a consistent path would be the dashed→solid→dashed path directed out of r^1, the solid→dashed→dashed path directed out of r^2, and the dashed→dashed path directed out of r^3. The first path dictates that $x_1 = 0, x_2 = 1, x_3 = 0$, the second path dictates that $x_2 = 1, x_3 = 0, x_4 = 0$, and the third path dictates that $x_4 = 0, x_5 = 0$. Since these variable assignments are consistent among the paths, this is a consistent path collection and corresponds to a feasible

solution $x' = (0, 1, 0, 0, 0)$ to $\mathcal{P}'$. Additionally, the sum of the weights of the paths $(0 + \frac{1}{2} + 0) + (\frac{1}{2} + 0 + 0) + (0 + 0) = 1$ corresponds to the objective value of x'.

An optimal solution to the CPP consists of the solid→dashed→dashed path directed out of r^1, the dashed→dashed→solid path directed out of r^2, and the solid→dashed path directed out of r^3. These paths coincide on their assignment of values to the problem variables $(1, 0, 0, 1, 0)$ and is of maximum total length among all paths $(1 + 0 + 0) + (0 + 0 + \frac{1}{2}) + (\frac{1}{2} + 0) = 2$, which corresponds to the optimal solution and optimal value to the problem, respectively.

3 Complexity Results

We first explore the complexity of the following problem, which we call the CPCD. Let $\mathcal{B} = \{B_1, \ldots, B_K\}$ be a collection of BDDs. The CPCD poses the question: "does there exist a consistent path collection of weight greater than or equal to D?"

To explore the complexity of CPCD, we will describe a transformation from the independent set problem described in Garey and Johnson (1979). Let $G = (V, E)$ be a graph with vertices $V = \{v_1, \ldots, v_n\}$ and edges $E \subseteq \binom{V}{2}$. An *independent set* is a subset $W \subseteq V$ satisfying that, for every edge $e \in E$, $|e \cap W| \leq 1$. We denote the DISP as the decision problem that asks the question: "does there exist an independent set in G of cardinality κ, where κ is a given positive integer constant value?"

Theorem 1. *Problem CPCD is NP-complete in the strong sense, even if the BDD collection in $\mathcal{B}$ is OA.*

Proof. To establish that CPCD belongs to NP, we can generate a series of K paths, one for each BDD, as a polynomial-sized guess and then check in time proportional to $\sum_{k=1}^{K} m^k$ that (a) the paths are consistent and (b) the total cost of those paths does not exceed D.

To show that CPCD is NP-complete, we perform a transformation from DISP. Let G be a DISP graph. We construct a collection of BDDs $\mathcal{B}$ consisting of one BDD per edge in G, for which $X(\mathcal{B}) = \{x_1, \ldots, x_n\}$. Setting variable $x_i = 1$ indicates that we include v_i in the independent set, and $x_i = 0$ indicates that we exclude v_i from the independent set.

We order the nodes in each edge $e = \{v_i, v_j\} \in E$ such that $i < j$ and construct a BDD $B(e)$ consisting of five nodes and six arcs. The layers are as follows: $L_1^{B(e)} = \left\{\mathbf{r}^{B(e)}\right\}, L_2^{B(e)} = \left\{u_1^{B(e)}, u_2^{B(e)}\right\}$, and $L_3^{B(e)} = \left\{\mathbf{t}^{B(e)}, \mathbf{f}^{B(e)}\right\}$, with $\text{var}(L_1^{B(e)}) = x_i$ and $\text{var}(L_2^{B(e)}) = x_j$; hence, the constructed BDDs indeed satisfy the OA property. The arc set is $A = \left\{a_1^{B(e)}, \ldots, a_6^{B(e)}\right\}$, with

$$
\begin{aligned}
a_1^{B(e)} &= \left(\mathbf{r}^{B(e)}, u_1^{B(e)}\right), \quad a_2^{B(e)} = \left(\mathbf{r}^{B(e)}, u_2^{B(e)}\right) \\
a_3^{B(e)} &= \left(u_1^{B(e)}, \mathbf{f}^{B(e)}\right), \quad a_4^{B(e)} = \left(u_1^{B(e)}, \mathbf{t}^{B(e)}\right) \\
a_5^{B(e)} &= \left(u_2^{B(e)}, \mathbf{t}^{B(e)}\right), \quad a_6^{B(e)} = \left(u_2^{B(e)}, \mathbf{t}^{B(e)}\right).
\end{aligned}
$$

The arc domains are 1 for the odd-indexed arcs, and 0 otherwise. Each BDD is generated sequentially, with the arc weights generated according to the following rules. The weights of the one-arcs are $w(a) = 1$ if $\text{var}(t(a))$ has not appeared in a previous BDD (noting that $\text{var}(t(a))$ is the vertex associated with the layer that the arc a is directed out of), and $w(a) = 0$ otherwise. Also, $w(a) = 0$ for all zero-arcs. Finally, set the CPCD target objective D equal to κ.

A depiction of the BDD for an edge e is shown in Figure 2, where a solid/dashed arc corresponds to an arc with arc-domain 0/1, and where we assume that both x_i and x_j appear for the first time in the BDD collection. (The superscripts $B(e)$ are omitted in this figure for simplicity.)

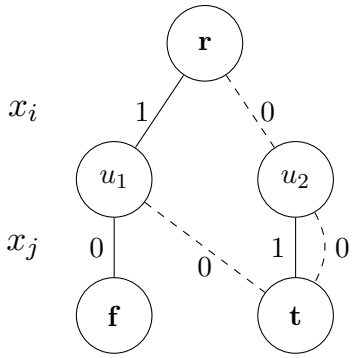

Figure 2: BDD for edge e

First, consider a DISP instance having a solution with objective value $W \geq \kappa$. For every BDD $B(e)$ in $\mathcal{B}$ with $e = \{v_i, v_j\}$, choose the path consisting of arcs:

- $a_1^{B(e)}$ and $a_4^{B(e)}$ if $v_i \in W$ and $v_j \notin W$;
- $a_2^{B(e)}$ and $a_5^{B(e)}$ if $v_i \notin W$ and $v_j \in W$; and
- $a_2^{B(e)}$ and $a_6^{B(e)}$ if $v_i \notin W$ and $v_j \notin W$,

recalling that at most one of v_i and v_j can belong to W because it is an independent set. Each such path ends at a true terminal and satisfies the consistency restrictions on the path collection. Moreover, this collection of paths encounters a total of $|W| \geq D = \kappa$ arcs having a weight of one. Therefore, the transformed consistent paths provide a solution to the CPCD instance.

Now, consider a CPCD instance having a solution given by consistent path collection $\mathcal{Q}'$ where $f(x(\mathcal{Q}')) \geq \kappa$. Since every path in $\mathcal{Q}'$ is directed from a root node $\mathbf{r}^{B(e)}$ to a true terminal node $\mathbf{t}^{B(e)}$, there is no edge $e = \{v_i, v_j\}$ for which $x(\mathcal{Q}')_i = x(\mathcal{Q}')_j = 1$. Therefore, the set $\tilde{V}(Q') = \{v_j : x(\mathcal{Q}')_j = 1\}$ forms an independent set.

Furthermore, we claim that $f(x(\mathcal{Q}')) = \sum_{j=1}^{n} x(\mathcal{Q}')_i$. Suppose $x(\mathcal{Q}')_i = 0$. Then, for each e with $v_i \in e$, the BDD $B(e)$ must select a path where the arc chosen on the layer associated with v_i is a zero-arc. Since each arc with arc domain 0 has weight 0, the contribution to $f(x(\mathcal{Q}'))$ for each of these variables is 0. On the other hand if $x(\mathcal{Q}')_i = 1$, then for each e with $v_i \in e$, the BDD $B(e)$ must select a path where the arc chosen on the layer associated with v_i is a one-arc. Exactly one arc with arc domain 1 has weight 1, and so the contribution to $f(x(\mathcal{Q}'))$ for each of these variables is 1. Solution $\mathcal{Q}'$ thus has an objective of at least $\kappa = D$, and therefore is a solution to CPCD.

Therefore, the CPCD instance is a yes-instance if and only if the original DISP instance is a yes-instance. Because the transformation was polynomial in size and all numerical data is binary, we have that the CPCD is strongly NP-complete. This completes the proof. □ □

A natural question that stems from the prior proof regards the CPCD complexity when the number of BDDs in the instance is fixed. The following theorem shows that even when there are just two BDDs in the collection, the CPCD is still NP-complete. Importantly, as we show later, we can no longer insist that the problem remains NP-hard for a fixed number of BDDs if the BDD collection satisfies the OA property.

Theorem 2. *The CPCD is strongly NP-complete, even for $K = 2$.*

Proof. Because we have already shown that CPCD belongs to NP, we prove this result via transformation from 3SAT, which is defined as follows (Garey and Johnson 1979).

> Consider a set of clauses $\mathcal{C}$ on a set $\mathcal{V} = \{v_1, \ldots, v_n\}$ of n binary variables (which can either take values of true or false). Each clause contains three literal values, each of which is a true or false value for one particular variable. Does there exist a *truth assignment* of binary values to the variables in $\mathcal{V}$, i.e., an assignment of true or false values for every variable such that every clause has a literal that matches some value in the assignment?

Let $\mathcal{C} = \{C_1, \ldots, C_\eta\}$. Let the indices of the variable elements included in C_j be given by $\text{ele}(C_j) = \{\ell_{1j}, \ell_{2j}, \ell_{3j}\}$, and define $\text{val}(\ell_{ij}) = \texttt{true}$ if a true value for $v_{\ell_{ij}}$ is a literal in the ith element of C_j and $\text{val}(\ell_{ij}) = \texttt{false}$ if a false value of $v_{\ell_{ij}}$ is a literal in the ith element of C_j. It is also convenient to define $M_i = \{\alpha_{i1}, \ldots, \alpha_{i|M_i|}\}$ as an ordered set of clause indices that involve 3SAT variable v_i either as a true or a false literal. Note that without loss of generality, we assume that $|M_i| \geq 2$ for each $i = 1, \ldots, n$.

The transformation creates a pair of BDDs over variables y_{ij}, for all $i = 1, \ldots, n$ and $j \in M_i$. The first BDD is designed to ensure that the values of y_{ij} are consistent, i.e., for each $i = 1, \ldots, n$, we must have that $y_{ij'} = y_{ij''}$ for all pairs of j' and j'' such that $j' \in M_i$ and $j'' \in M_i$. The second BDD is designed to ensure that every clause is satisfied by at least one assignment of the variables associated with the clause. All arc weights equal zero in both BDDs, and the objective target $D = 0$ for the CPCD instance.

For the first BDD, we begin by defining the root node r^1. The algorithm below constructs this BDD in layers, one corresponding to each BDD variable y_{ij}. Steps 1, 2a, and 2b are initialization steps.

Step 1. Create nodes u^1_{11T} and u^1_{11F}. The first node represents the event that $y_{1\alpha_{11}}$ is assigned to true, and the second is analogously defined for setting $y_{1\alpha_{11}}$ to be false. Create a one-arc from r^1 to u^1_{11T} and a zero-arc from r^1 to u^1_{11F}. These arcs correspond to $y_{1\alpha_{11}}$. If $|M_i| > 2$ then go to Step 2a, and otherwise ($|M_i| = 2$) go to Step 2b.

Step 2a. Create nodes u^1_{12T}, u^1_{12F}, and u^1_{12X}. The first two nodes are defined as above for the second clause that includes v_1. The third denotes that an inconsistency has been found at some point in the path, whereby some variable was assigned a value of true for one clause but false for another. Create a one-arc from u^1_{11T} to u^1_{12T}, a zero-arc from u^1_{11F} to u^1_{12F}, a zero-arc from u^1_{11T} to u^1_{12X}, and a one-arc from u^1_{11F} to u^1_{12X}. These arcs correspond to $y_{1\alpha_{12}}$. Set $i = 1$ and $j = 3$. If $j < |M_i|$ then go to Step 4, and otherwise go to Step 5.

Step 2b. Create nodes u^1_1 and u^1_{1X}. The first node represents the event that $y_{1\alpha_{11}} = y_{1\alpha_{12}}$, and the second is defined for the case in which $y_{1\alpha_{11}} \neq y_{1\alpha_{12}}$. Create a one-arc from u^1_{11T} to u^1_1 and a zero-arc from u^1_{11F} to u^1_1. Also, create a zero-arc from u^1_{11T} to u^1_{1X} and a one-arc from u^1_{11F} to u^1_{1X}. These arcs correspond to $y_{1\alpha_{12}}$. Set $i = 2$ and $j = 1$, and go to Step 3.

Step 3. Create nodes u^1_{i1T}, u^1_{i1F}, and u^1_{i1X}. Create a one-arc from u^1_{i-1} to u^1_{i1T}, a zero-arc from u^1_{i-1} to u^1_{i1F}, a zero-arc from $u^1_{i-1,X}$ to u^1_{i1X}, and a one-arc from $u^1_{i-1,X}$ to u^1_{i1X}. These arcs correspond to $y_{i\alpha_{i1}}$. Set $j = 2$. If $|M_i| > 2$ then go to Step 4; otherwise, go to Step 5.

Step 4. Create nodes u^1_{ijT}, u^1_{ijF}, and u^1_{ijX}. Create a one-arc from $u^1_{i,j-1,T}$ to u^1_{ijT}, a zero-arc from $u^1_{i,j-1,F}$ to u^1_{ijF}, a zero-arc from $u^1_{i,j-1,T}$ to u^1_{ijX}, and a one-arc from $u^1_{i,j-1,F}$ to u^1_{ijX}. Also, create parallel zero- and one-arcs that both go from $u^1_{i,j-1,X}$ to u^1_{ijX}. These arcs correspond to $y_{i\alpha_{ij}}$. Increment j by one. If $j < |M_i|$ then repeat Step 4; otherwise, go to Step 5.

Step 5. Create nodes u^1_i and u^1_{iX}. Create a one-arc from u^1_{ijT} to u^1_i and a zero-arc from u^1_{ijF} to u^1_i. Also, create a zero-arc from u^1_{ijT} to u^1_{iX}, a one-arc from u^1_{ijF} to u^1_{iX}, and parallel zero- and one-arcs that both go from u^1_{ijX} to u^1_{iX}. These arcs correspond to $y_{i\alpha_{ij}}$. If $i = n$ then go to Step 6. Otherwise, increment i by one, set $j = 1$, and go to Step 3.

Step 6. Relabel u^1_n as t^1, relabel u^1_{nX} as f^1, and terminate.

For the second BDD, we define the root node r^2 and construct the BDD by linking subnetworks corresponding to each clause C_j, $j = 1, \ldots, \eta$. The subnetwork for C_j forces a path to either choose at least one decision corresponding to a literal that satisfies C_j or forces the path to use arcs that lead only to the failure state as before. Nodes u^2_{hjT}, for $h = 1, 2, 3$ and $j = 1, \ldots, \eta$, represent the situation in which all clauses $C_1, \ldots, C_j$ have been satisfied by our variable assignments and that, in particular, C_j was satisfied by one of the first h elements of the clause. Nodes u^2_{hjF}, for $h = 1, 2$ and $j = 1, \ldots, \eta$, represent the situation in which all clauses $C_1, \ldots, C_{j-1}$ have been satisfied by our variable assignments, but C_j has not been satisfied by one of the first h elements of the clause. Any node of the form u_{hjX} corresponds to an infeasible assignment due to at least one of the clauses $\{C_1, \ldots, C_j\}$. The subnetworks are built as follows.

Subnetwork 1. Build layers as follows.

- Create a layer with nodes u^2_{11T} and u^2_{11F}. Incoming arcs to these nodes are associated with $y_{\ell_{11}1}$. If $\text{val}(\ell_{11}) = \texttt{true}$, then create a one-arc from r^2 to u^2_{11T} and a zero-arc from r^2 to u^2_{11F}. Otherwise, create a zero-arc from r^2 to u^2_{11T} and a one-arc from r^2 to u^2_{11F}.
- Create a layer with nodes u^2_{21T} and u^2_{21F}. Incoming arcs to these nodes are associated with $y_{\ell_{21}1}$. Create parallel zero- and one-arcs from u^2_{11T} to u^2_{21T}. If $\text{val}(\ell_{21}) = \texttt{true}$, then create a one-arc from u^2_{11F} to u^2_{21T} and a zero-arc from u^2_{11F} to u^2_{21F}. Otherwise, create a zero-arc from u^2_{11F} to u^2_{21T} and a one-arc from u^2_{11F} to u^2_{21F}.
- Create a layer with nodes u^2_{31T} and u^2_{31X}. Incoming arcs to these nodes are associated with $y_{\ell_{31}1}$. Create parallel zero- and one-arcs from u^2_{21T} to u^2_{31T}. If $\text{val}(\ell_{31}) = \texttt{true}$, then create a one-arc from u^2_{21F} to u^2_{31T} and a zero-arc from u^2_{21F} to u^2_{31X}. Otherwise, create a zero-arc from u^2_{21F} to u^2_{31T} and a one-arc from u^2_{21F} to u^2_{31X}.

Subnetwork $j = 2, \ldots, \eta$. This construction mirrors that for subnetwork 1, except that it starts from $u^2_{3,j-1,T}$ and contains nodes u^2_{hjX} at each layer of the BDD.

- Create a layer with nodes u^2_{1jT}, u^2_{1jF}, and u^2_{1jX}. Incoming arcs to these nodes are associated with $y_{\ell_{1j}j}$. Create parallel zero- and one-arcs from $u^2_{3,j-1,X}$ to u^2_{1jX}. If $\text{val}(\ell_{1j}) = \texttt{true}$, then create a one-arc from $u^2_{3,j-1,T}$ to u^2_{1jT} and a zero-arc from $u^2_{3,j-1,T}$ to u^2_{1jF}. Otherwise, create a zero-arc from $u^2_{3,j-1,T}$ to u^2_{1jT} and a one-arc from $u^2_{3,j-1,T}$ to u^2_{1jF}.
- Create a layer with nodes u^2_{2jT}, u^2_{2jF}, and u^2_{2jX}. Incoming arcs to these nodes are associated with $y_{\ell_{2j}j}$. Create parallel zero- and one-arcs from u^2_{1jT} to u^2_{2jT}, and parallel zero- and one-arcs from u^2_{1jX} to u^2_{2jX}. If $\text{val}(\ell_{2j}) = \texttt{true}$, then create a one-arc from u^2_{1jF} to u^2_{2jT} and a zero-arc from u^2_{1jF} to u^2_{2jF}. Otherwise, create a zero-arc from u^2_{1jF} to u^2_{2jT} and a one-arc from u^2_{1jF} to u^2_{2jF}.
- Create a layer with nodes u^2_{3jT} and u^2_{3jX}. Incoming arcs to these nodes are associated with $y_{\ell_{3j}j}$. Create parallel zero- and one-arcs from u^2_{2jT} to u^2_{3jT}, and parallel zero- and one-arcs from u^2_{2jX} to u^2_{3jX}. If $\text{val}(\ell_{31}) = \texttt{true}$, then create a one-arc from u^2_{2jF} to u^2_{3jT} and a zero-arc from u^2_{2jF} to u^2_{3jX}. Otherwise, create a zero-arc from u^2_{2jF} to u^2_{3jT} and a one-arc from u^2_{2jF} to u^2_{3jX}.

To finish construction of this BDD, we relabel $u^2_{3\eta T}$ as t^2 and $u^2_{3\eta X}$ as f^2. Observe that the size of both BDDs is polynomial in terms of the size of the 3SAT instance, because both BDDs contain $O(\eta)$ nodes and no additional numerical data. Figure 3 shows an example transformation for a

3SAT instance with the following clauses $C_1 = (v_1 \text{ true } \vee v_2 \text{ false } \vee v_3 \text{ true})$, $C_2 = (v_1 \text{ false } \vee v_2 \text{ true } \vee v_3 \text{ false})$, and $(v_1 \text{ false } \vee v_2 \text{ true } \vee v_3 \text{ true})$.

With the transformation to a two-BDD CPCD instance now complete, we show that the original 3SAT instance is a yes-instance if and only if the CPCD instance is a yes-instance. Consider a yes-instance to 3SAT with solution $\bar{v}$. In the BDDs, set solution $\bar{y}_{ij} = \bar{v}_i$ for all $i = 1, \ldots, n$ and $j \in M_i$. In the first BDD, this choice results in a path that starts at r^1, and for each $i = 1, \ldots, n-1$, visits nodes u^1_{ijT}, $j = 1, \ldots, |M_i| - 1$, followed by u^1_i if $\bar{v}_i = 1$ and visits nodes u^1_{ijF}, $j = 1, \ldots, |M_i| - 1$ followed by u^1_i, if $\bar{v}_i = 0$. The same pattern holds for $i = n$, except the path ends in t^1 in both cases. In the second BDD, the path starts at r^2, and for each $j = 1, \ldots, \eta - 1$ visits nodes in the following order: u^2_{1jT}, u^2_{2jT}, u^2_{3jT} if clause j is satisfied by the first literal; u^2_{1jF}, u^2_{2jT}, u^2_{3jT} if clause j is satisfied by the second literal; and u^2_{1jF}, u^2_{2jF}, u^2_{3jT} if clause j is satisfied by the third literal. The same pattern holds for $j = \eta$, except that the terminating node is t^2 in all three cases. Because there exists a pair of paths in the two BDDs that are consistent and terminate at t^1 and t^2, the transformed solution guarantees a solution to CPCD as well, verifying it as a yes-instance.

Now, suppose that the CPCD instance has a solution, and that this solution corresponds to a set of variables $\bar{y}$. By construction, $\bar{y}_{ij'} \neq \bar{y}_{ij''}$ is impossible if $j' \in M_i$ and $j'' \in M_i$, or else the solution would correspond to a path in the first BDD that terminates at f^1. Thus, consider the 3SAT solution $\bar{v}_i = \bar{y}_{ij}$ for any $j \in M_i$. The path in the second BDD could not have visited any node u^2_{hjX}, and therefore must encounter all nodes u^2_{3jT} for $j = 1, \ldots, \eta - 1$ and then t^2. Visiting these nodes verifies that each each clause is satisfied (by the first clause element if u^2_{1jT} is visited; by the second clause element if u^2_{1jF} and u^2_{2jT} are visited; and by the third clause element otherwise). Therefore, the 3SAT instance has a solution given by $\bar{v}$, verifying that the 3SAT instance is a yes-instance as well.

Because no numerical data is used in the transformation, the proof establishes that CPCD is NP-complete even when there are only two BDDs. This completes the proof. □ □

The two prior theorems show that CPCD is NP-complete even when the OA property holds when the number of BDDs is arbitrary, and that the CPCD is NP-complete even when there are just two BDDs. However, the proof of Theorem 2 does not create an OA collection. In fact, it turns out that the CPP is substantially easier when both the OA property holds and the number of BDDs is fixed. We now show that the CPP defined over collections of BDDs that are OA can be solved in polynomial time, for fixed K.

Theorem 3. *Let $\mathcal{B} = \{B_1, \ldots, B_K\}$ be a collection of BDDs. Deciding if a consistent path collection of weight greater than or equal to D exists is polynomial in the number of nodes in the BDDs for fixed K if the BDDs in $\mathcal{B}$ are OA.*

Before proceeding with the proof, we discuss the notion of *intersection* of BDDs (where, for the remainder of this section we assume the BDDs are OA), which will be critical for the proof and for the design of valid inequalities in the next section. Given two BDDs B_1 and B_2, an *intersection* BDD, denoted by $B_1 \cap B_2$, is one in which (a) $X(B_1 \cap B_2) = X(B_1) \cup X(B_2)$, (b) $\mathbf{Sol}(B_1 \cap B_2) = \mathbf{Sol}(B_1) \cap \mathbf{Sol}(B_2)$, and (c) $\forall p \in \mathbf{Sol}(B_1 \cap B_2), w^{B_1 \cap B_2}(p) = w^{B_1}(p^{B_1}(x)) + w^{B_2}(p^{B_2}(x))$. It is well known that there exists an intersection BDD of width $\omega(B_1) \cdot \omega(B_2)$ and that such a BDD can be found in time proportional to $O(n\omega(B_1) \cdot \omega(B_2))$ (Bryant 1986, 1992, Lai et al. 1993).

For example, consider Figure 4, which illustrates an intersection BDD. The two BDDs on the left are intersected to construct the BDD on the right. Any **r**–**t** path in the right BDD corresponds to a CPC, which consists of the paths in both BDDs on the left associated with the path in the BDD intersection.

An intersection BDD can be constructed by processing the layers one-by-one through a concurrent top-down traversal of the BDDs, by associating a pair of nodes, one from each original BDD, with every node created in the intersection BDD. We first construct the root node of the intersection BDD, and associated it with the two root nodes of the original BDDs. Then, having constructed layer L_j of the intersection BDD, we proceed to create layer L_{j+1} by expanding the nodes in the intersection BDD one-by-one. For any node uv in layer L_j of the intersection BDD, which is associated with nodes u and v, respectively, in layer j of the original BDDs B_1 and B_2 respectively, we examine the outgoing arcs of u and v. Suppose that the zero-arc exiting u in B_1 (v in B_2) connects to u_0 (v_0). Also, suppose that the one-arc exiting u in B_1 (v in B_2) connects to u_1 (v_1). A new node u_0v_0 is created in layer L_{j+1} of the intersection BDD, a zero-arc is added that connects uv to u_0v_0, and the weight of the arc is $w_{\beta(u)} + w_{\beta(v)}$. Similarly, we create a new node u_1v_1 in layer L_{j+1} of the intersection BDD, add a one-arc to the intersection BDD that connects uv to u_1v_1, and set the weight of that arc to $w_{\alpha(u)} + w_{\alpha(v)}$. After creating all such nodes in L_{j+1}, those nodes that have common labels are *merged* into a single node along with all incoming arcs. Note that any node and arc in an intersection BDD for $\mathcal{B} = \{B_1, \ldots, B_K\}$ is associated with exactly K nodes and arcs from the component BDDs.

This construction assumes that $X(B_1) = X(B_2)$. If not, we can add a new layer associated with every variable in $X(B_2) \setminus X(B_1)$ to B_1 (and can likewise do the same for missing variables in B_2) without changing $\mathbf{Sol}(B)$. Suppose we wish to add variable x_j to BDD B between layers L_j and L_{j+1}. Create an additional set of nodes L of size $|L_{j+1}|$, each associated with a single node in L_{j+1}. For every $v \in L_{j+1}$, let v be the corresponding node in L. For every arc $a = (u, v)$ in the original BDD with $u \in L_j$ and $v \in L_{j+1}$, redirect the arc so that $h(a) = \mathsf{v}$. Keep all other attributes of the arcs unchanged. Then, for every new node v, create two arcs $\alpha(\mathsf{v})$ and $\beta(\mathsf{v})$ that are directed from v to v, each with zero arc weight. The set of solutions remains unchanged and the size of the BDDs grows in size by $O(\max\{|N^{B_1}|, |N^{B_2}|\})$, a polynomial in the size of the BDDs.

Proof. (Proof of Theorem 3.)

Consider the iterative intersection of the BDDs in $\mathcal{B}$, $(((B_1 \cap B_2) \cdots) \cap B_m)$. The time to construct this intersection is proportional to $O(n \cdot \omega(B_1) \cdots \omega(B_K))$, resulting in a single BDD with width at most $\omega(B_1) \cdots \omega(B_K)$. Let W be the maximum width of any BDD in $\mathcal{B}$. The size of the intersection BDD is $O(n \cdot W^K) = O(n^{K+1})$, which can be found in time $O(n^K)$. The longest-weight path in the intersection BDD can then be found in time linear in the size of the resulting BDD. By the properties of intersection BDDs, this will correspond to the largest-weight consistent path in $\mathcal{B}$. □ □

4 A Cutting-Plane Algorithm

A natural approach to solving the CPP is to reformulate the problem via a network-flow lifting (Bergman and Ciré 2016, 2018). For every $a \in A^{B_1} \cup \cdots \cup A^{B_K}$ define variable y_a to indicate if arc a is chosen in the CPC. Let $\delta^+(u)/\delta^-(u)$ be the arcs directed out of/into node u, respectively. Recalling that, for each BDD B_k and for each layer j in B_k, $\text{var}(L_j^{B_k})$ corresponds to a variable x_i,

consider the network-flow reformulation of the CPP defined over $\mathcal{B}$ in NFR:

$$\begin{aligned}
\max \quad & \sum_{a \in A^{B_1} \cup \cdots \cup A^{B_K}} w_a y_a && && \text{(NFR)}\\
\text{s.t.} \quad & \sum_{a \in \delta^+(\mathbf{r}^{B_k})} y_a = \sum_{a \in \delta^-(\mathbf{t}^{B_k})} y_a = 1 && \forall k \in \{1, \ldots, K\} &\\
& \sum_{a \in \delta^+(u)} y_a = \sum_{a \in \delta^-(u)} y_a && \forall k \in \{1, \ldots, K\}, \forall u \in N^{B_k} \setminus \left\{\mathbf{r}^{B_k}, \mathbf{t}^{B_k}\right\} &\\
& x_{\mathsf{i}\left(L_j^{B_k}\right)} = \sum_{u \in L_j^{B_k}} \sum_{a \in \delta^+(u): d(a)=1} y_a && \forall k \in \{1, \ldots, K\}, \forall j \in \left\{1, \ldots, m^{B_k}\right\} &\\
& y_a \in \{0,1\} && \forall a \in A^{B_1} \cup \cdots \cup A^{B_K} &\\
& x_i \in \{0,1\} && \forall i \in \{1, \ldots, n\}. &
\end{aligned}$$

The variables y_a, together with the flow constraints, enforce that a CPC is selected. The original problem variables are forced to equate to the sum of the one-arcs on the layers that correspond to that variable across all BDDs, which provides the linking between the constraints.

We now describe a convex hull model for NFR formulated in the space of the x- and y-variables, assuming that the OA property is satisfied. Let $\hat{B}$ be the intersection BDD of all BDDs in $\mathcal{B}$. Let $\bar{a}$ be an artificial arc for which $t(\bar{a}) = \mathbf{t}^{\hat{B}}$ and $h(\bar{a}) = \mathbf{r}^{\hat{B}}$. Let $A^+ = A^{\hat{B}} \cup \{\bar{a}\}$ and let $\Lambda(a')$ be the set of all arcs $a \in A^{\hat{B}}$ associated with $a' \in A^{B_1} \cup \cdots \cup A^{B_K}$. Define flow variables v_a for every $a \in A^+$ and let (x', y') be fixed values for the NFR problem variables. We define the following max-flow optimization problem:

$$\begin{aligned}
\hat{z}(y') = \max \quad & v_{\bar{a}} && && \text{(MFP}(y')\text{)}\\
\text{s.t.} \quad & \sum_{a \in \delta^+(u)} v_a - \sum_{a \in \delta^-(u)} v_a = 0 && \forall u \in N^{\hat{B}} && \text{(2a)}\\
& \sum_{a \in \Lambda(a')} v_a \leq y'_{a'} && \forall a' \in A^{B_1} \cup \cdots \cup A^{B_K} && \text{(2b)}\\
& v_a \geq 0 && \forall a \in A^+. && \text{(2c)}
\end{aligned}$$

Now consider the dual of MFP(y'). For any $a \in A^{\hat{B}}$, let $\hat{\Lambda}(a)$ be the set of arcs $a' \in A^{B_1} \cup \cdots \cup A^{B_K}$ associated with arc a (note that $|\hat{\Lambda}(a)| = K$). Let $\boldsymbol{\pi}$ denote the dual variables associated with constraints (2a) and let $\boldsymbol{\lambda}$ be the dual variables associated with constraints (2b). The dual of MFP(y') is given by:

$$\begin{aligned}
\min \quad & \sum_{a' \in A^{B_1} \cup \cdots \cup A^{B_K}} y'_{a'} \lambda_{a'} && && \text{(3a)}\\
\text{s.t.} \quad & \pi_{\mathbf{t}^{\hat{B}}} - \pi_{\mathbf{r}^{\hat{B}}} \geq 1 && && \text{(3b)}\\
& \pi_{t(a)} - \pi_{h(a)} + \sum_{a' \in \hat{\Lambda}(a)} \lambda_{a'} \geq 0 && \forall a \in A^{\hat{B}} && \text{(3c)}\\
& \boldsymbol{\pi} \text{ unrestricted} && && \text{(3d)}\\
& \boldsymbol{\lambda} \geq \mathbf{0}. && && \text{(3e)}
\end{aligned}$$

Let Ψ be the set of extreme points of the polyhedron defined by constraints (3b)–(3e) and define

$$\Psi' = \{\boldsymbol{\lambda} \mid \exists \boldsymbol{\pi} \text{ such that } (\boldsymbol{\lambda}, \boldsymbol{\pi}) \in \Psi\} \tag{4}$$

as the projection of Ψ onto the λ-variables. Note that Ψ and Ψ' do not depend on y'.

Theorem 4. *Define formulation CH-NFR as the linear programming relaxation of NFR extended with the following inequalities:*

$$\sum_{a' \in A^{B_1} \cup \cdots \cup A^{B_K}} \lambda_{a'} y_{a'} \geq 1 \quad \forall \boldsymbol{\lambda} \in \Psi'. \tag{5}$$

CH-NFR describes a convex hull formulation for NFR

Proof. (of Theorem 4.)

We first show that inequalities (5) do not cut any integer feasible solution $(\bar{x}, \bar{y})$ to NFR. Because $(\bar{x}, \bar{y})$ is an integer feasible solution, there must exist exactly one path in $\hat{B}$ corresponding to the CPC selected by $(\bar{x}, \bar{y})$), and so $\hat{z}(\bar{y}) = 1$ necessarily. By weak duality, inequalities (5) only eliminate solutions (x', y') for which $\hat{z}(y) < 1$. Inequalities (5) thus do not cut any integer feasible solution.

We now show that all the extreme points of the polyhedron defined by CH-NFR, P(CH-NFR), are integer. Suppose by contradiction that (x', y') is a fractional extreme point of P(CH-NFR). Let $\mathbf{v}^*$ be an optimal solution to MFP(y') and recall that $\hat{z}(y') = 1$. Because (x', y') is fractional, there are at least two paths from $\mathbf{r}^{\hat{B}}$ to $\mathbf{t}^{\hat{B}}$ for which the flow selected by $\mathbf{v}^*$ is positive. Let $p^1, \ldots, p^h$ denote the aforementioned paths and let $\hat{f}(p)$ denote the flow selected by $\mathbf{v}^*$ on path p. Note that $\sum_{l \in \{1,\ldots,h\}} \hat{f}(p^l) = 1$ since $\hat{z}(y') = 1$. Recall that each path $p^1, \ldots, p^h$ in $\hat{B}$ corresponds to a CPC, denoted by $Q^1, \ldots, Q^h$. Let $f(p')$ denote the flow selected by y' on path p' (in a component BDD from $\mathcal{B}$). Constraints (2b) imply that

$$\hat{f}(p^l) \leq f(p'), \ \forall l \in \{1, \ldots, h\}, \ \forall p' \in Q^l. \tag{6}$$

Since $\sum_{l \in \{1,\ldots,h\}} \hat{f}(p^l) = 1$, flow balance constraints in CH-NFR imply that

$$\hat{f}(p^l) = f(p'), \ \forall l \in \{1, \ldots, h\}, \ \forall p' \in Q^l; \tag{7}$$

otherwise, there would be a component BDD in $\mathcal{B}$ for which the flow from $\mathbf{r}$ to $\mathbf{t}$ would be strictly greater than 1 or for which a path flow in some Q^l were negative. We conclude that

$$(x', y') = \sum_{l \in \{1,\ldots,h\}} \hat{f}(p^l)(x^l, y^l), \tag{8}$$

where (x^l, y^l) is the unique integer feasible solution to NFR corresponding to Q^l. Since any integer feasible solution to NFR is also a feasible solution to CH-NFR, this contradicts the assumption that (x', y') is an extreme point of P(CH-NFR). □ □

We propose a cutting-plane approach in which inequalities (5) are added iteratively to the linear programming relaxation of NFR. Figure 5 presents an example solution to MFP(y') on an intersection BDD used to obtained a cut based on Theorem 4. Figure 5(a) shows two component BDDs with arcs indexed from 1 to 16. Figure 5(b) displays a fractional flow obtained as a solution to the linear programming relaxation of NFR (non-zero flows displayed alongside the arcs). Figure 5(c) presents the corresponding intersection BDD and the associated optimal solution to MFP(y') given the fractional flows from the left. Note that the only integer feasible solutions to NFR are $x_1 = 1, x_2 = 0, x_3 = 1$ and $x_1 = 0, x_2 = 1, x_3 = 1$. The maximum flow obtained after solving MFR(y') in the intersection BDD is 0 since no flow can be sent in the second layer as $y_{12} = 0$ and $y_{13} = 0$. As a result, we obtain the cut $y_{12} + y_{13} \geq 1$, which eliminates the fractional solution depicted in Figure 5(b) but is satisfied by both feasible solutions.

The intersection BDD can grow exponentially large as a function of K. In order to address this, we propose to consider intersection BDDs on subsets of $\mathcal{B}$. Namely, let $\{\Gamma^1, \ldots, \Gamma^U\}$ satisfy that $\Gamma^l \subseteq \mathcal{B}$ and $|\Gamma^l| \geq 2$ for all $l \in \{1, \ldots, U\}$. We create an intersection BDD for each Γ^l, and, after solving the linear programming relaxation of NFR, if the solution is fractional, we seek cutting planes of the form (5) for each intersection BDD generated. Any cuts valid for these intersection BDDs will be trivially valid for the entire problem. We discuss implementation details specific to the applications tested in the following section.

5 Experimental Results

Experimental results were run on a machine having an Intel Xeon E5–1650 CPU running at 3.60 GHz with 32 GB of RAM on Windows 10. We coded our algorithm in Java using Eclipse SDK version 4.7.1 and all optimization problems were solved using CPLEX 12.7.1 with a time limit of one hour (3600s).

5.1 Results for UBCP

Random UBCP instances were generated to test the effectiveness of the approach presented in this paper. The instances are parametrized by the number of variables n in the instance, the number of quadratic terms q, and the number of cubic terms c. The procedure assigns random objective coefficient values for all the linear terms from a discrete uniform distribution $U(-20, 20)$. Then selects q unique two-element subsets of $\{1, \ldots, n\}$ and c unique three-elements subsets of $\{1, \ldots, n\}$ (independently from the two-element subsets selected), each selected uniformly at random, and assigns to each subset s selected an objective coefficient value w_s drawn independently and uniformly at random from a discrete uniform distribution $U(-20, 20)$. We set $w_s = 0$ for all two- and three-element subsets not selected by the procedure. Five instances were generated for each configuration of $n \in \{25, 30, 35, 40\}$, $q \in \{0, 250, 500\}$, and $c \in \{500, 1000, 1500\}$.

Consider any subset $s' = \{i, j, k\}$ with $|s'| = 3$, and the BDD in Figure 6, where, with a slight abuse of notation, we drop the set notation from subscripts (e.g., $w_{\{i,j\}} \equiv w_{i,j}$), and any arcs without a label are to be assigned arc-cost 0. In this diagram, the total length of each path corresponds to the objective function terms that arise from sets $s \subseteq 2^{\{i,j,k\}}$. For example, the path $u_1 \to u_3 \to u_6 \to u_8$ along the dashed→solid→solid arcs corresponds to solution $x_i = 0, x_j = 1, x_k = 1$ with length $0 + (w_j) + (w_{j,k} + w_k)$ which equals the objective function evaluation of these three variables.

We note in passing that the BDD can be compressed via reduction (Hooker 2013). For example, if $w_{i,j} = w_{j,k}$, nodes u_5 and u_6 can be merged by redefining the head of arc (u_3, u_6) to u_5, and removing node u_6 and all arcs directed out of it. This will not affect the lengths of paths that correspond to solutions. Suppose further that this common value was 0. Then node u_7 could also be merged with u_5 and u_6.

For a binary cubic optimization instance, a valid CPP can be constructed as follows. We arbitrarily order the cubic terms for which $w_s \neq 0$. For each term s' in this order, we construct a BDD as in Figure 6, including the objective function terms with non-zero coefficients corresponding to subsets of s' if they have not appeared in any BDD created thus far. This ensures that we do not double count objective function terms by including them in multiple BDDs. Note that there may exist quadratic terms that are not represented by any of the BDDs generated for the cubic terms. In this case we add an additional binary variable for each quadratic term not represented in the BDDs and formulate the model using a standard McCormick linearization technique as shown below.

We define subsets of BDDs for the cut-generation procedure as follows. For any $i, j \in \{1, \ldots, n\}$ we generate a subset of BDDs $\Gamma_1^{i,j}$ containing every BDD associated with a subset $s \in \{\{l, r, k\} \mid l = i,\ r = j,\ k \in \{1, \ldots, n\}\}$, $w_s \neq 0$ and a subset of BDDs $\Gamma_2^{i,j}$ containing BDDs corresponding to subsets $s \in \{\{l, k, r\} \mid l = i,\ r = j,\ k \in \{1, \ldots, n\}\}$, $w_s \neq 0$. We then generate cuts iteratively as described below, where NFRLP($\mathcal{C}$) denotes the linear programming relaxation of NFR including generated cuts in set $\mathcal{C}$.

Step 1: Solve NFRLP($\mathcal{C}$) and record the optimal solution (x', y') found, the objective value, z, and the dual variables corresponding to the cuts, π'.

Step 2: Solve MFP(y') for every $\Gamma_1^{i,j}$ and $\Gamma_2^{i,j}$ generated. If no new cuts are found, then terminate the procedure. Otherwise, add the new cuts to $\mathcal{C}$ and remove from $\mathcal{C}$ any cut c whose corresponding dual variable $\pi'_c \geq 0.1$.

Step 3: If the procedure has generated more than 5000 cuts or z has not decreased in the last 50 iterations, then terminate the procedure. Otherwise, return to Step 1.

In order to evaluate the effectiveness of the cutting-plane algorithm, we compare three algorithms. The first is the base network-model reformulation of the consistent path problem defined on the collection of BDDs described above, denoted by `BDD No Cuts`. The second is this reformulation together with cutting planes, denoted by `BDD Plus Cuts`. The third, denoted by `IP`, is the natural reformulation of associating additional binary variables $x_{i,j}$ with every two-index term with a non-zero $w_{i,j}$ and $x_{i,j,k}$ with every three-index term with a non-zero $w_{i,j,k}$:

$$\max \quad \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{i,j} x_{i,j} + \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} w_{i,j,k} x_{i,j,k} \qquad \text{(UBCP-IP)}$$

$$\begin{aligned}
\text{s.t.} \quad & x_{i,j} \geq x_i + x_j - 1; && \forall i, j \in \{1, \ldots, n\}, i < j \\
& x_{i,j} \leq x_i; \quad x_{i,j} \leq x_j; && \forall i, j \in \{1, \ldots, n\}, i < j \\
& x_{i,j,k} \geq x_i + x_j + x_k - 2; && \forall i, j, k \in \{1, \ldots, n\}, i < j < k \\
& x_{i,j,k} \leq x_i; \quad x_{i,j,k} \leq x_j; \quad x_{i,j,k} \leq x_k; && \forall i, j, k \in \{1, \ldots, n\}, i < j < k \\
& x_i, x_{i,j}, x_{i,j,k} \in \{0, 1\} && \forall i, j, k \in \{1, \ldots, n\}, i < j < k
\end{aligned}$$

Figure 7 depicts a cumulative distribution plot of performance comparing all three algorithms. The plot is split into two sections. The left portion indicates the number of instances solved by a given time. The right shows the number of instances that have at most the corresponding percent optimality gap after 3600 seconds. Note that all instances solved within 3600 seconds have an optimality gap of 0 and so the plot, for each algorithm, is non-decreasing.

Interestingly, `IP` outperforms `BDD No Cuts` on this benchmark set, showing that without the use of the cuts identified in the present paper, the use of the network-model reformulation via BDDs is ineffective. However, by introducing cuts, we see that `BDD Plus Cuts` is able to solve more instances in less computational time than `IP`. Also, even among those instances not solved in 3600 seconds, the gaps are significantly smaller when cuts are added, further exhibiting the necessity to include the cutting planes.

A more refined view of the differences in runtime can be seen through the scatter plots in Figure 8. For each plot there is a point for every instance, with the coordinates corresponding to the runtime for `BDD Plus Cuts`, as well as `BDD No Cuts` and `IP`, respectively, on the left and right plots. For each point, the color corresponds to c, the size corresponds to n, and the shape corresponds to q. Any point above the diagonal indicates an instance for which `BDD Plus Cuts` had shorter solution times than the algorithm of comparison. These plots show that there are smaller, easier instances for which the other algorithms can outperform `BDD Plus Cuts`, but as the size of

the instance and the complexity (measured by the number of objective function terms) grows, the performance of `BDD Plus Cuts` enhanced in comparison to the other algorithms.

Detailed solution statistics appear in the Appendix.

5.2 Results for MMP

To test the cutting-plane approach for the MMP, instances were generated by a similar approach to that for the market split problem in Cornuéjols and Dawande (1999). In particular, we generate five random instances for each configuration of $r \in \{20, 22, 24, 26, 28\}$, $p \in \{4, 6, 8\}$, and $d \in \{10, 12, 14\}$. Coefficients a_{ij} are drawn independently and uniformly at random from a discrete uniform $U(0, 100)$. Right-hand side coefficients are set to $b_i = \left(\sum_{j=1}^{r} a_{ij}\right)/d$, for all $i = 1, \ldots, p$.

We generate BDDs corresponding to constraints (1a) based on the following dynamic program (DP). For a given constraint i let $\Lambda^k = \{\lambda^k\}$ be the state of the system at stage k, i.e., having assigned values to $k-1$ x-variables. State variable λ^k records the maximum between the goal defined by b_i and the sum of $a_{ij}\hat{x}_{jq}$ for variables already decided at stage k. Let t denote the terminal state and $\Lambda^1 = 0$. We define transition function as follows:

$$\Lambda^{k+1} = \phi(\Lambda^k, \hat{x}_{jq}), \tag{9}$$

where $\hat{x}_{jq}$ is the value assigned to variable x_{jq}, and

$$\phi(\Lambda^k, \hat{x}_{jq}) = \begin{cases} t & \text{if } x_{jq} \text{ is the last variable in constraint } i \\ \max\{b_i, \lambda^k + a_{ij}\hat{x}_{jq}\} & \text{otherwise} \end{cases} \tag{10}$$

Transition costs capture the deviation from the desired goal and are defined as follows:

$$c(\Lambda^k, \Lambda^{k+1}, \hat{x}_{jq}) = \begin{cases} a_{ij}\hat{x}_{jq} & \text{if } \lambda^k \geq b_i \\ \max\{0, (\lambda^k + a_{ij}\hat{x}_{jq}) - b_i\} & \text{if } \Lambda^{k+1} \neq t \\ |b_i - (\lambda^k + a_{ij}\hat{x}_{jq})| & \text{otherwise} \end{cases} \tag{11}$$

We generate a BDD from the state-transition graph of the DP described above. Figure 9 shows an example of a BDD generated for constraint $3x_{11} + 5x_{21} + 4x_{31} + s_{11} = 4$.

Note that since the demand for a product i in retailer j (denoted by a_{ij}) is independent of the division, the BDDs generated for constraints corresponding to the same product share the same set of nodes, arcs, and costs. The difference between these BDDs appears in the x-variables associated with the layers, which vary according to the division.

Instead of reformulating MMP as a CPP, we embed a relaxed CPP into MIP formulation (1) to enforce a lower bound on the optimal objective value. We define this relaxation by replacing all the BDDs associated with the same product by only one BDD and requiring d units of flow from $\mathbf{r}$ to $\mathbf{t}$. As a result, our proposed BDD decomposition has $K = p$ BDDs. Let $\mathbf{s}^+$ and $\mathbf{s}^-$ denote the upper and lower deviation from the established goals, respectively. Define $\mathsf{i}\left(L_j^{B_k}\right)$ as the index associated with the retailer in the x-variable corresponding to layer j of B_k. The MIP formulation

used for the `BDD No Cuts` approach is:

$$\max \quad \sum_{i=1}^{p}\sum_{q=1}^{d}(s_{iq}^{+}+s_{iq}^{-}) \tag{12a}$$

$$\text{s.t.} \quad \sum_{j=1}^{r} a_{ij}x_{jq} - s_{iq}^{+} + s_{iq}^{-} = b_i \qquad \forall i \in \{1,\dots,p\}, \forall q \in \{1,\dots,d\} \tag{12b}$$

$$\sum_{q=1}^{d} x_{jq} = 1 \qquad \forall j \in \{1,\dots,r\} \tag{12c}$$

$$\sum_{a\in\delta^{+}(\mathbf{r}^{B_k})} y_a = \sum_{a\in\delta^{-}(\mathbf{t}^{B_k})} y_a = d \qquad \forall k \in \{1,\dots,p\} \tag{12d}$$

$$\sum_{a\in\delta^{+}(u)} y_a = \sum_{a\in\delta^{-}(u)} y_a \qquad \forall k \in \{1,\dots,p\}, \forall u \in N^{B_k} \setminus \left\{\mathbf{r}^{B_k}, \mathbf{t}^{B_k}\right\} \tag{12e}$$

$$\sum_{q=1}^{d} x_{\mathrm{i}\left(L_j^{B_k}\right),q} = \sum_{u\in L_j^{B_k}} \sum_{a\in\delta^{+}(u):d(a)=1} y_a \qquad \forall k \in \{1,\dots,p\}, \forall j \in \left\{1,\dots,m^{B_k}\right\} \tag{12f}$$

$$\sum_{i=1}^{p}\sum_{q=1}^{d}(s_{iq}^{+}+s_{iq}^{-}) \geq \sum_{a\in A^{B_1}\cup\cdots\cup A^{B_K}} w_a y_a \tag{12g}$$

$$y_a \in \{0,1\} \qquad \forall a \in A^{B_1}\cup\cdots\cup A^{B_K} \tag{12h}$$

$$x_{jq} \in \{0,1\} \qquad \forall j \in \{1,\dots,r\}, \forall q \in \{1,\dots,d\} \tag{12i}$$

$$s_{iq}^{+},\ s_{iq}^{-} \geq 0 \qquad \forall i \in \{1,\dots,p\}, \forall q \in \{1,\dots,d\}. \tag{12j}$$

Constraints (12b)–(12c) are the original constraints from MIP formulation (1). Constraints (12d) require that d units of flow traverse each one of the BDDs in the CPP. Constraints (12e) ensure flow balance. Constraints (12g) link the x-variables with the flow on the arcs on the BDDs. Note that constraints (12g) aggregate x-variables associated with the same retailer index. As a result, paths in a CPC could represent infeasible solutions to the original MMP and the total flow cost is a lower bound on the original objective function, which is enforced by constraint (12g).

Again we compare the three algorithms, `BDD Plus Cuts`, `BDD No Cuts`, and `IP`. Detailed solution statistics appear in the Appendix.

Our results indicate that using BDDs proves critical to identifying a practical solution to the instances tested. In particular, of the 225 instances tested, `IP` only solves 45, while `BDD No Cuts` is able to solve 168. Adding the cuts allows two more instances to be solved, for a total of 170. Additionally, the average runtimes over those instances solved by `BDD No Cuts` is 256.81, while for `BDD Plus Cuts` it is 248.39. These results indicate that adding cuts to network models can increase efficiency and reduce solution times, albeit to a modest extent for this application.

To summarize our findings, a cumulative distribution plot of performance appears in Figure 10. This plot readily shows how effective the BDD decomposition is: Without including the network models the IP is unable to effectively solve the problem.

6 Conclusion and Future Work

In this paper we contribute a formal analysis of the consistent path problem. We provide a detailed analysis of the complexity of the problem and a complete description of the convex hull of feasible

solutions, when the variables in each network are order associated. Our results indicate that the cutting-plane algorithm developed is effective in reducing the solution times and the remaining optimality gap for both the unconstrained binary cubic optimization problem and the market multisplit problem.

This work lays the groundwork for investigating efficient algorithms for optimizing over a collection of BDDs. This paper focuses on the case when the BDDs are order associated. Our theoretical results indicate that the consistent path problem where the BDDs are not order associated is significantly harder, and identifying cutting planes in the more general setting is a natural and important extension. Future work includes investigating cutting planes for the more general version of the CPP, when the BDDs are not order associated, exploring other combinatorial optimization problems for which we can exploit BDD decompositions, and studying alternative techniques for solving the CPP.

References

B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer. BDDs in a branch and cut framework. In S. Nikoletseas, editor, *Experimental and Efficient Algorithms, Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms (WEA 05)*, volume 3503 of *Lecture Notes in Computer Science*, pages 452–463, Heidelberg, Germany, 2005. Springer.

D. Bergman, W. J. van Hoeve, and J. N. Hooker. Manipulating MDD relaxations for combinatorial optimization. In T. Achterberg and J. C. Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 8th International Conference, CPAIOR*, volume 6697 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2011.

D. Bergman, A. A. Ciré, W.-J. van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016a.

David Bergman and Andre A. Ciré. Decomposition based on decision diagrams. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming: 13th International Conference, CPAIOR*, volume 9676 of *Lecture Notes in Computer Science*, pages 45–54. Springer International Publishing, 2016.

David Bergman and Andre A. Ciré. Discrete nonlinear optimization by state-space decompositions. *Management Science*, page to appear, 2018.

David Bergman, André A. Ciré, Willem-Jan van Hoeve, and John N. Hooker. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, Cham, Switzerland, 2016b.

R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

R. E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

G. Cornuéjols and Milind Dawande. A class of hard small 0-1 programs. *INFORMS Journal on Computing*, 11(2):205–210, 1999.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman & Co., Princeton, NJ, 1979.

T. Hadžić and J. N. Hooker. Postoptimality analysis for integer programming using binary decision diagrams. Technical report, Carnegie Mellon University, Pittsburgh, PA, 2006.

T. Hadžić and J. N. Hooker. Cost-bounded binary decision diagrams for 0-1 programming. In E. Loute and L. Wolsey, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 4th International Conference, CPAIOR*, volume 4510 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007.

T. Hadžić, J. N. Hooker, B. O'Sullivan, and P. Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In P. J. Stuckey, editor, *Principles and Practice of Constraint Programming (CP 2008)*, volume 5202 of *Lecture Notes in Computer Science*, pages 448–462. Springer, 2008.

Tarik Hadžić, Eoin O'Mahony, Barry O'Sullivan, and Meinolf Sellmann. Enhanced inference for the market split problem. In *21st IEEE International Conference on Tools with Artificial Intelligence*, pages 716–723, Los Alamitos, CA, 2009. IEEE Computer Society.

S. Hoda, W.-J. van Hoeve, and J. N. Hooker. A systematic approach to MDD-based constraint programming. In *Proceedings of the 16th International Conference on Principles and Practices of Constraint Programming*, volume 6308 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2010.

John N. Hooker. Decision diagrams and dynamic programming. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 10th International Conference, CPAIOR*, volume 7874 of *Lecture Notes in Computer Science*, pages 94–110. Springer, Berlin, Heidelberg, 2013.

Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula. BDD based decomposition of logic functions with application to EPGA synthesis. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 642–647, Dallas, TX, 1993. ACM Press.

Guillaume Perez and Jean-Charles Régin. Relations between MDDs and tuples and dynamic modifications of MDDs based constraints. *arXiv preprint arXiv:1505.02552*, 2015.

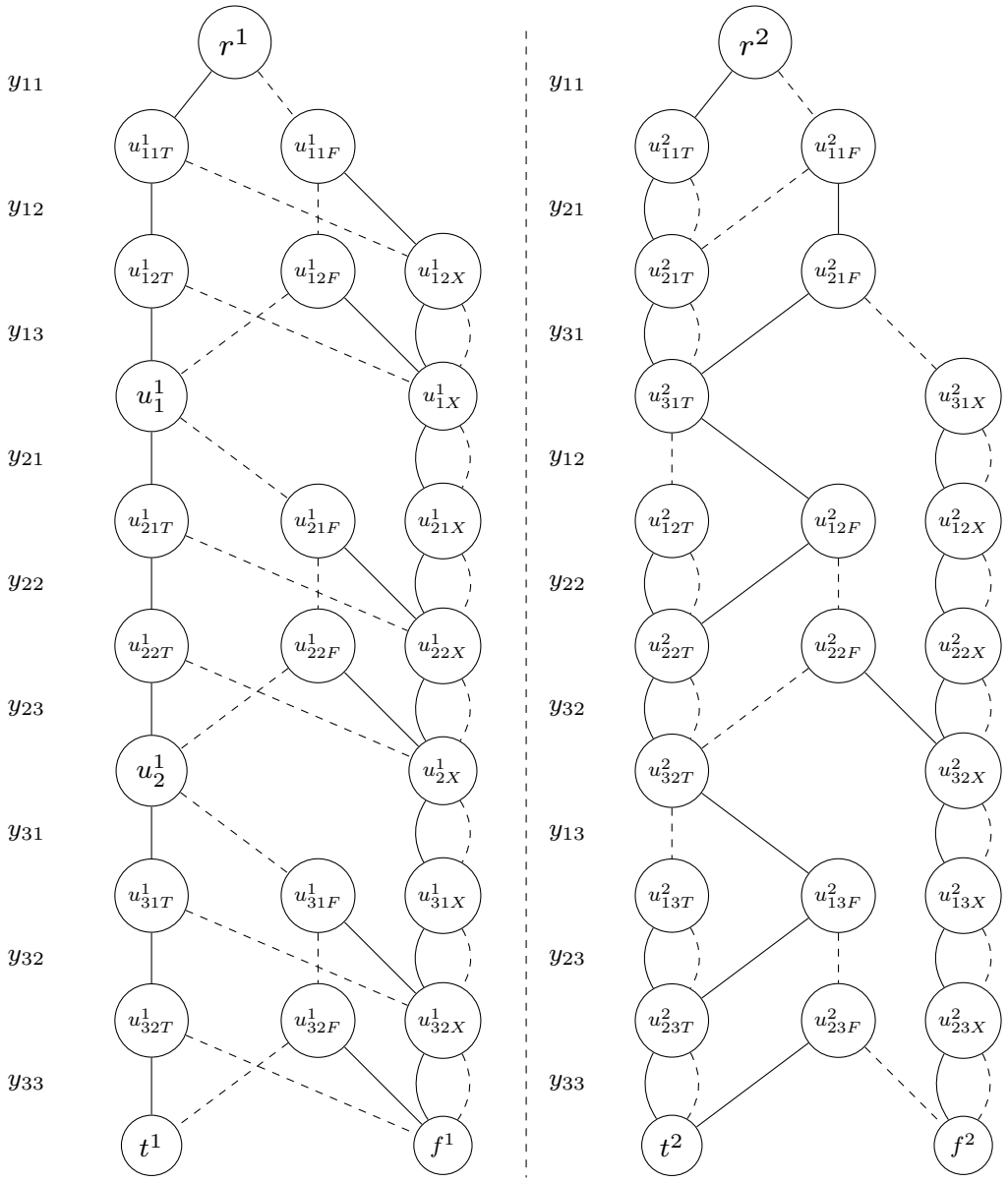

Figure 3: BDDs used to transform a 3SAT instance into a CPCD instance.

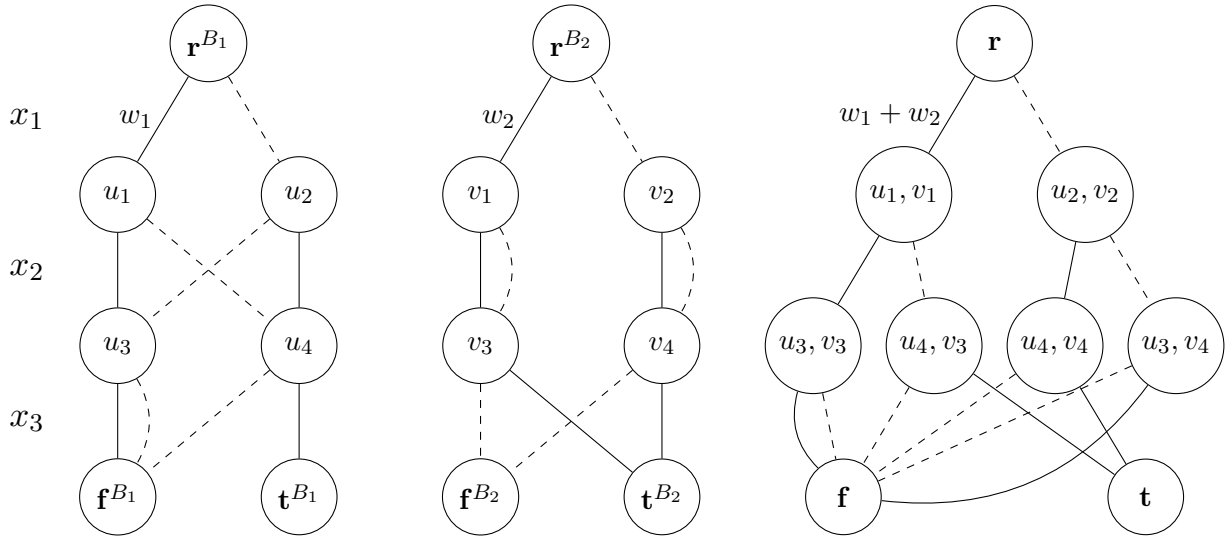

Figure 4: Building an intersection BDD

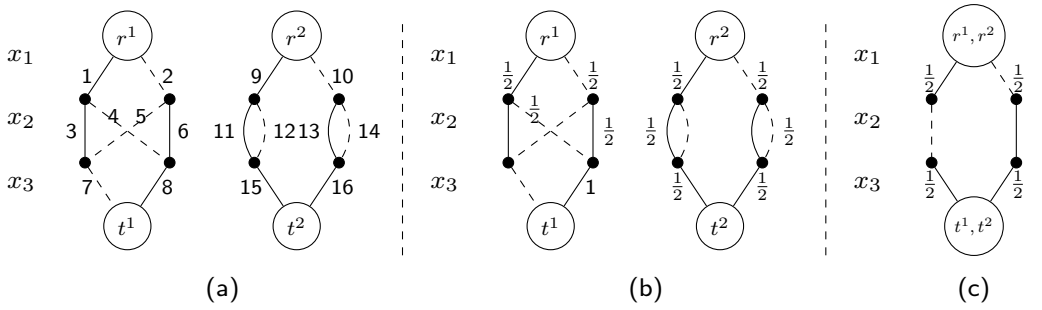

Figure 5: (a) Two component BDDs with arcs numbered from 1 to 16 (b) Fractional solution to NFR linear programming relaxation (c) Optimal solution to MFR over an intersection BDD .

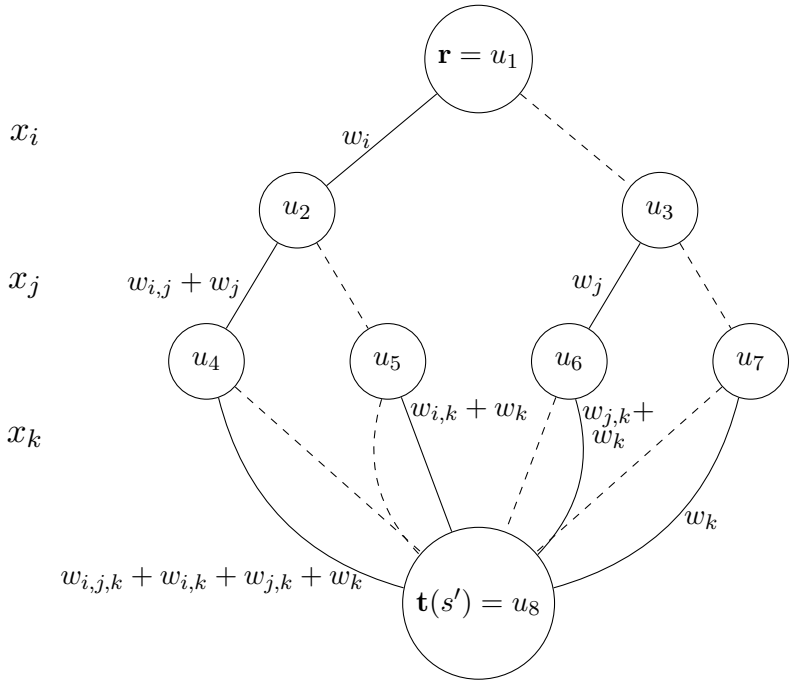

Figure 6: BDD for all cubic and quadratic terms defined on $\{i, j, k\}$.

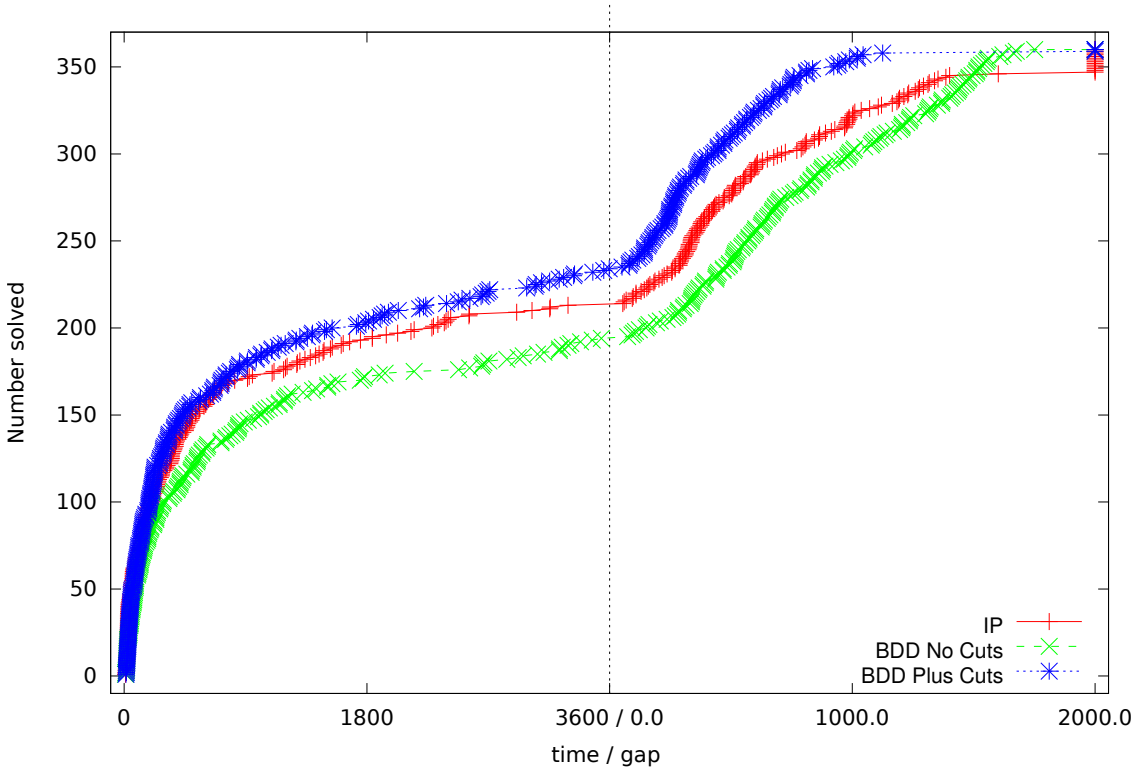

Figure 7: Cumulative distribution plot of performance, comparing `BDD Plus Cuts`, `BDD No Cuts`, and `IP`

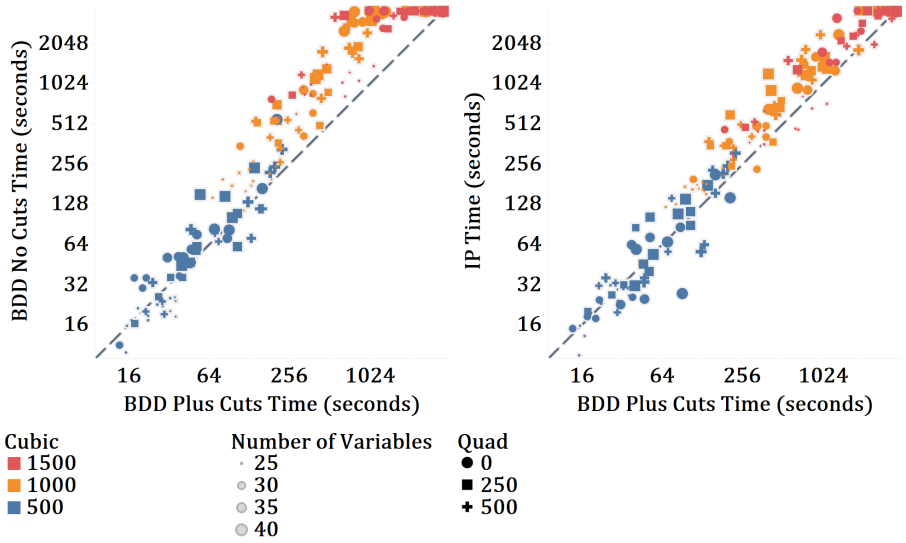

Figure 8: Scatter plots comparing runtime differences between `BDD Plus Cuts`, `BDD No Cuts`, and `IP`

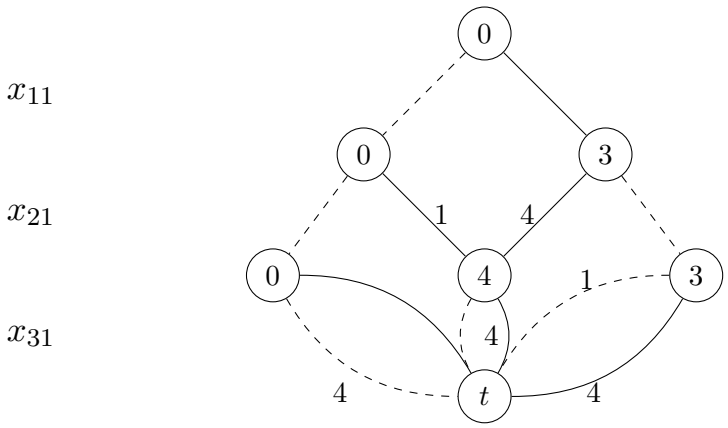

Figure 9: BDD corresponding to an example constraint from MMP.

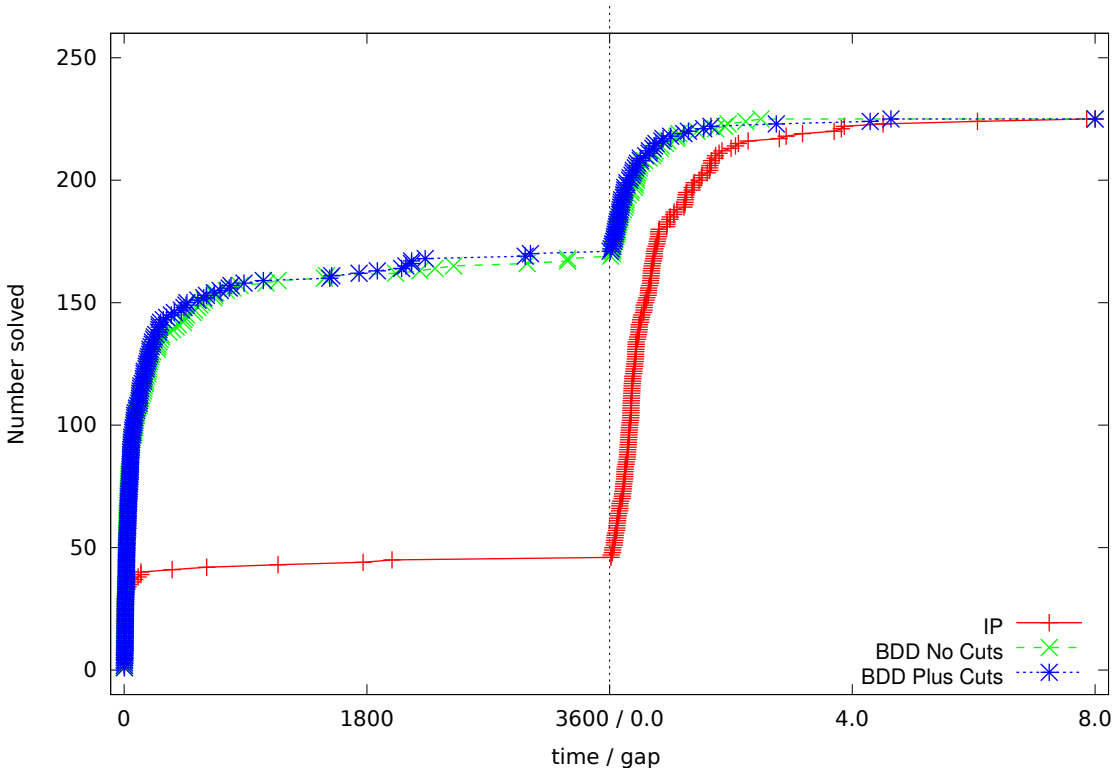

Figure 10: Cumulative distribution plot of performance, comparing `BDD Plus Cuts`, `BDD No Cuts`, and `IP`

7 Appendix

7.1 Computational Results

Table 2 presents detailed results comparing algorithms `IP`, `BDD No Cuts`, and `BDD Plus Cuts` on UBCP instances. The first two columns show the number of variables and the number of cubic terms. The remaining columns show the average CPU time in seconds (Time), the number of instances solved within the one-hour time limit (#Sol), the average upper bound obtained after solving the linear programming relaxation in the root node (Root UB), and the average optimality gap among the instances that are not solved within the time limit (Gap), for the three algorithms. Each row reports results over 15 instances. We record a time of 3600 seconds if the instance is not solved within the time limit or if the algorithm reached an out-of-memory exception.

Table 2: Comparing `IP`, `BDD No Cuts`, and `BDD Plus Cuts` on UBCP instances

n	c	IP				BDD No Cuts				BDD Plus Cuts			
		Time	#Sol	Root UB	Gap	Time	#Sol	Root UB	Gap	Time	#Sol	Root UB	Gap
25	500	24	15	1982.6	-	20	15	1775.7	-	26	15	1043.9	-
	1000	184	15	3630.1	-	210	15	3414.9	-	136	15	1675.8	-
	1500	496	15	5394.9	-	920	15	5185.0	-	480	15	2467.4	-
30	500	40	15	2037.6	-	35	15	1805.9	-	33	15	1201.1	-
	1000	380	15	3692.3	-	508	15	3434.9	-	271	15	1808.6	-
	1500	2052	14	5469.9	0.2	2866	8	5205.0	0.3	1599	15	2553.1	-
35	500	79	15	2212.1	-	69	15	1958.9	-	75	15	1450.7	-
	1000	1316	14	3922.8	0.1	2118	13	3640.8	0.1	779	15	2103.6	-
	1500	3156	4	5693.6	0.3	3556	2	5392.6	0.4	2615	10	2795.1	0.2
40	500	126	15	2169.7	-	176	15	1953.6	-	118	15	1530.6	-
	1000	1750	12	3875.2	0.1	2719	8	3587.7	0.2	1235	15	2232.5	-
	1500	>3600	0	5592.3	0.3	>3600	0	5305.9	0.4	3583	1	2916.0	0.1

Table 3 presents computational results for our experiments on MMP instances. The first two columns show the number of retailers and the number of products. Columns "Time", "#Sol", and "Gap" are defined as before, while column "Root LB" presents the average lower bound obtained after solving the linear programming relaxation in the root node. Each row reports results over 15 instances and we record a time of 3600 seconds if the instance is not solved to optimality within the time limit.

Table 3: Comparing `IP`, `BDD No Cuts`, and `BDD Plus Cuts` on MMP instances

r	p	IP				BDD No Cuts				BDD Plus Cuts			
		Time	#Sol	Root LB	Gap	Time	#Sol	Root LB	Gap	Time	#Sol	Root LB	Gap
20	4	971	11	23.9	0.2	5	15	397.2	-	8	15	398.2	-
	6	2888	3	36.1	0.1	13	15	577.9	-	15	15	579.4	-
	8	2648	4	45.7	0.2	23	15	770.2	-	34	15	772.4	-
22	4	1206	10	19.3	0.2	9	15	295.6	-	14	15	296.3	-
	6	3374	1	30.9	0.4	81	15	473.0	-	144	15	475.0	-
	8	>3600	0	45.3	0.4	600	14	597.3	0.2	445	15	599.3	-
24	4	2310	6	20.5	0.5	332	14	228.4	0.2	69	15	230.0	-
	6	>3600	0	31.1	0.6	636	14	338.7	0.4	625	13	342.4	0.1
	8	3444	1	42.5	0.6	1561	9	455.0	0.2	1558	9	459.0	0.2
26	4	2569	5	23.5	1.5	1085	11	188.7	0.1	806	14	190.0	0.02
	6	3132	2	32.7	0.8	1795	8	271.5	0.3	1708	9	275.0	0.2
	8	3391	1	42.5	0.6	2427	7	357.4	1.0	2466	6	360.9	0.4
28	4	3376	1	22.1	1.9	1736	9	137.9	0.5	1735	9	140.0	0.4
	6	>3600	0	32.5	1.7	2867	4	189.9	0.5	2901	4	194.5	0.8
	8	>3600	0	43.9	0.7	3413	3	268.6	0.8	3518	1	273.2	0.9