Decision Diagram Decomposition for Quadratically Constrained Binary Optimization

David Bergman[†1] and Leonardo Lozano[2]

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

October 2, 2018

Abstract

In recent years the use of decision diagrams within the context of discrete optimization has proliferated. This paper continues this expansion by proposing the use of decision diagrams for modeling and solving binary optimization problems with quadratic constraints. The model proposes the use of multiple decision diagrams to decompose a quadratic matrix so that each component diagram has provably limited size. The decision diagrams are then linked through channeling constraints to ensure that the solution represented is consistent across the decision diagrams and that the original quadratic constraints are satisfied. The resulting family of decision diagrams are optimized over by a dedicated cutting-plane algorithm akin to Benders decomposition. The approach is general, in that commercial integer programming solvers can readily apply the technique. A thorough experimental evaluation on both benchmark and synthetic instances exhibits that the proposed decision diagram reformulation provides significant improvements over current methods for quadratic constraints in state-of-the-art solvers.

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

1 Introduction

Integer programming (IP) applications are pervasive in the real world. Industry demand for IP software has led researchers to study general-purpose algorithms for this broad class of optimization problems, and companies to develop commercial solvers with state-of-the-art implementations of those algorithms. This enables practitioners to have ready access to the power of optimization. Leading the way are `CPLEX` (Optimizer 2018) and `Gurobi` (Gurobi Optimization 2018), perhaps the most commonly used IP solvers both in academia and in industry.

Alongside the solvers a series of benchmark libraries provide instances for wide-scale testing and evaluation of competing optimization solvers. Of note is the `MIPLIB` instances (available online at http://miplib.zib.de/), that provide a range of problem sizes and characteristics, so that researchers can test and evaluate general-purpose and problem-specific algorithms on a common set of instances. This also allows researchers to track solver progress and encourages them to expand the computational limits of IP, with some studies estimating machine-independent speedups on the order of thousands per decade (Bixby 2012).

*leolozano@uc.edu
†david.bergman@uconn.edu

The tremendous improvement in algorithms and solvers for *linear* (mixed) IP has caused a recent shift in research focus. The research literature, and commercial solvers, are actively developing general-purpose algorithms for (mixed) IP models with *quadratic* terms. This is reflected in the expansion of modeling capability in both `CPLEX` and `Gurobi` to allow quadratic terms, and the creation of `QPLIB` (Furini et al. 2017), to serve the same purpose as `MIPLIB`—provide a wide-range of instances with varying size and characteristics that researchers can use to evaluate the merits of their proposed algorithms.

The body of work on quadratic programming is vast (see Gould (2000) for a thorough list of references on quadratic programming, with accompanying website listing references ftp://ftp.numerical.rl.ac.uk/pub/qpbook/qpbook.bib). In discrete optimization, most work has focused on optimization problems with quadratic *objective functions* (Billionnet et al. 2013, Buchheim and Wiegele 2013, Kochenberger et al. 2014). More specifically, there is also a large body of work on IP problems with quadratic *constraints*, so called *(mixed)-integer quadratically constrained quadratic programming* (MIQCP) (see Section 5 of Burer and Letchford (2012) for a survey of approaches).

We focus on binary optimization problems with non-convex quadratic constraints, for which there are two common solution approaches. The first is a direct linear transformation of each quadratic term $x_i \cdot x_j$, either using a quadratic (Glover and Woolsey 1974) or linear (Glover 1975) number of additional variables (many such transformation exists). Alternatively, one can approach the problem via a two-step reformulation whereupon non-convex constraints (i.e., ones for which the matrix in the quadratic term is not positive semi-definite) are converted into convex constraints, and then tight relaxations are formulated, most often based on semi-define programming (Billionnet et al. 2009), noting that there is an entire body of literature focusing on solving binary and integer optimization problems where the objective and constraints are convex (see, for example, Buchheim et al. (2012)). This latter approach is often called the *quadratic convex reformulation* (QCR) method.

Optimization solvers have adopted the best general-purpose techniques for handling non-convex quadratic constraints, to provide users with the best approaches available in the literature. Nonetheless, quadratic constraints, particularly those that are non-convex, present a considerable computational challenge.

This paper presents an automated mechanism for reformulating convex and non-convex quadratic constraints for IQCP. The reformulation does not presuppose any structure on the matrices in the constraints. Thorough experimental evaluation indicates that the reformulation considerably outperforms CPLEX solving the same model, thereby providing a general-purpose reformulation technique that can be readily implemented in commercial solvers to yield orders-of-magnitude improvement in solution times. The reformulation is based on *decision diagram decomposition*.

The use of decision diagrams (DDs) in optimization has grown in interest within the optimization community (Becker et al. 2005, Behle and Eisenbrand 2007, Bergman et al. 2011, Gatterbauer and Suciu 2014, Bergman et al. 2014, 2016, Haus and Michini 2016, 2017, Nishino et al. 2015, Perez and Régin 2015, 2016) (see also Bergman et al. (2016)). Most aligned in scope with the current paper is the work on using multiple DDs to represent and solve optimization problems (Bergman and Cire 2016, 2018, Lozano et al. 2018).

In Bergman and Cire (2016) the idea of decomposing a problem so that DDs can exploit local structure is introduced. They propose an automatic *network flow* reformulation for translating the DDs into a lifted space, where the paths in the collection of DDs are connected to the original problem variables through linear channeling constraints. The authors applied to idea to the maximum independent set and unconstrained binary quadratic programming showing speedups over single DD-based optimization approaches and a commercial IP solver on a limited set of instances.

DD-based decomposition was expanded in Bergman and Cire (2018). Three real-world non-linear

optimization problems from portfolio optimization, healthcare management, and retail product assortment were modeled and solved by identifying decomposition and utilizing the network-flow reformulations from Bergman and Cire (2016).

The promise of DD-based decomposition is further improved by Lozano et al. (2018), where the authors refer to the problem of optimizing over connected decision diagrams as the *consistent path problem* (CPP). A formal computational complexity study and polyhedral analysis is conducted. Cutting planes and separation routines are presented that result in improved algorithms for unconstrained binary cubic optimization and a variant of the market split problem (Cornuéjols and Dawande 1999, Hadžić et al. 2009).

The literature on multiple DD-based optimization, to the best of our knowledge, has only focused on reformulating complicated objective function terms. This paper studies the application of multiple DDs to quadratic constraints. In particular, a novel decomposition of quadratic constraints for which each component has a DD of limited size is introduced and proved correct. The decomposition was suggested in Bergman and Cire (2016) for decomposing a quadratic objective function terms, but was presented without proof of correctness. This paper fills this gap in the literature.

We present a novel reformulation of quadratic constraints for binary quadratically constrained programming problems that does not presuppose any structure on the matrices—they can be of any form and need not belong to the set of semi-definite positive matrices or any other restrictive class. Our approach is an automated, and general mechanism for reformulating quadratic constraints that can be readily implemented by commercial optimization solvers. This can be combined with methods used for dealing with quadratic objective function terms, more general convex objective functions or any other mechanism for relaxing complicated elements of a problem, but to evaluate the effectiveness of the approach in isolation of confounding factors, we focus on problems with linear objective functions.

The procedure calls for decomposing quadratic terms in the constraints into component matrices, each of which has a provable limited-size *binary decision diagram* (BDD). The BDDs in common constraints are linked to enforce that the constraint they participate in are satisfied, and the BDDs in different constraints are linked to ensure that the paths selected are consistent. This paper also presents a cutting-plane approach for solving the ensuing BDD decomposition through branch and cut. This is a general approach, that can be implemented to solve any problem containing quadratic constraints on binary variables.

In order to test the effectiveness of the proposed algorithm, a thorough experimental evaluation is conducted. The instances tested include all binary problem instances in `QPLIB` that have linear objective functions and quadratic constraints, as well as synthetic randomly generated instances to broaden the scope of the experimental evaluation. In almost every instance, our proposed algorithm provides considerable improvements in runtime over `CPLEX` solving the same model.

SUMMARY OF CONTRIBUTIONS: The main contributions of the paper are therefore:

- An application of BDD decomposition to quadratically constrained binary optimization problems;
- An adaptation of the decomposition of objective function matrices introduced in Bergman and Cire (2016) to quadratic constraints;
- A formal proof that the decomposition is valid;
- A formal proof that the component matrices in the decomposition each have limited-size BDDs; and

- *A generalizable and state-of-the-art generic framework for solving quadratically constrained binary optimization problems that can readily be adapted by commercial optimization solvers.*

The remainder of the paper is organized as follows. Section 2 formally introduces quadratically constrained binary optimization problems, and Section 3 introduces BDDs for this class of problems. Section 4 describes a construction algorithm for the BDDs. Section 5 describes the decomposition that we exploit for constructing limited-width BDDs. Section 6 provides a base arc-based model for transforming a BDD decomposition into a network flow model. Section 7 presents a cutting-plane algorithm for optimizing over the BDDs. Section 8 reports results from a thorough experimental evaluation. We conclude in Section 9 with some ideas for future work.

2 Quadratically Constrained Binary Optimization

This paper focuses on solution approaches to *quadratically constrained binary optimization problems* (QCBOP). Specifically, we study the problem

$$\min \quad c^T x \tag{1a}$$
$$\text{s.t.} \quad x^T Q^h x \geq b_h, \quad h = 1, \ldots, q \tag{1b}$$
$$x \in \mathbb{B}^n. \tag{1c}$$

Note that more complex objective functions can be incorporated, since the methodology presented focuses on novel linear reformulation of the quadratic constraints through BDDs, but to streamline the exposition we focus solely on linear objective functions. We also focus on binary optimization and so specialize the discussion to binary decision diagrams, though the concepts and algorithms extend to general integer variables.

3 Binary Decision Diagrams for Quadratic Forms

We use BDDs to represent the evaluation of a quadratic function over binary variables. Specifically, for the context of this paper, a binary decision diagram $B = (m, N, \ell, A, d, \text{var})$ 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* $\ell(u) \in \{1, \ldots, m+1\}$, 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} also only consists of one node; the *true terminal* $\mathbf{t}$. 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., $\omega(B) = \max_j \omega(L_j)$). This is a key parameter that we seek to control.

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 $u \neq \mathbf{t}$ has 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 $X = \{x_1, \ldots, x_n\}$ each of the first m layers $L_1, \ldots, L_m$ is associated with a unique variable in X (note, $m \leq n$) which is denoted by $\text{var}(L_j)$. The index of the x-variable $\text{var}(L_j)$ is denoted

by $\mathsf{i}(L_j)$. 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 $p = (a_1, \ldots, a_m)$ be a path from $\mathbf{r}$ to $\mathbf{t}$ (i.e., $t(a_1) = \mathbf{r},\ h(a_m) = \mathbf{t}$, and for $j = 1, \ldots, m-1,\ h(a_j) = t(a_{j+1})$). 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 $j \in \{1, \ldots, m\}$, 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.

If (i) $\mathcal{S}(B) = \mathbf{Sol}(P)$ and (ii) for every $x \in \mathbf{Sol}(P)$, $x^T Q x = w\left(p^B(x)\right)$, we say that B is an *exact* BDD for Q, where Q is any real-valued $n \times n$ matrix. For general matrices, the size of the BDD will be exponential in n.

When it is necessary to distinguish between BDDs because several are in use, we will use a superscript to indicate which BDD (for example, the root node of BDD B will be denoted by $\mathbf{r}^B$).

Example 1 *(adapted from Bergman and Cire (2016)) Figure 1 depicts a BDD for matrix function matrix Q'. Arcs a with $d(a) = 0$ or 1 are depicted as dashed or solid, respectively. The weight $w(a)$ is shown next to each arc. Consider, for example, the path corresponding to the root to terminal*

$$Q' = \begin{bmatrix} +1 & -1 & +1 & 0 & 0 \\ -1 & -1 & -1 & -1 & 0 \\ +1 & -1 & -1 & +1 & -1 \\ 0 & -1 & +1 & +1 & +1 \\ 0 & 0 & -1 & +1 & -1 \end{bmatrix}$$

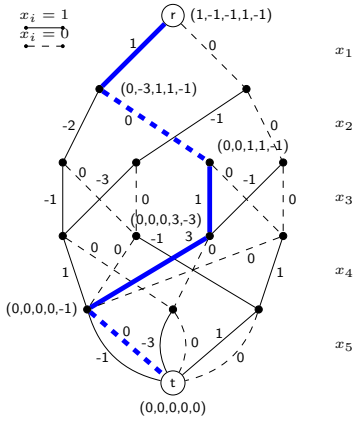

Figure 1: Constraint matrix for an instance with an exact BDD. States are indicated on nodes in which the path corresponding to solution $(1, 0, 1, 1, 0)$ traverses.

path of solid-dashed-solid-solid-dashed. This corresponds to solution $x' = (1, 0, 1, 1, 0)$. It is easy to verify that $x'Q'x' = 5$, as does the sum of the weights of the arcs on the paths.

4 BDD Construction

The depicted BDD also contains vector state information on the nodes of the highlighted path, which is used to construct the BDD. This will be discussed next.

Given Q, one can build an exact BDD iteratively through recursion Bergman and Cire (2016). For general problem classes, dynamic-programming models are associated, and common states are merged during construction, layer-by-layer. Since the focus of this paper is on QCBOP, the discussion is tailored specifically to this problem class.

A construction algorithm for creating BDDs for quadratic objective functions was introduced in Bergman and Cire (2016) without proof of correctness, which is reproduced and formally proven in Theorem 1. The proof describes the algorithm and proves it's correctness.

Theorem 1 *There exists an algorithm for constructing an exact BDD for objective matrix Q that runs in time $O(n \cdot \omega(B) \cdot \log(\omega(B)))$.*

Proof. We begin by defining a compilation procedure which associates a *state* $\sigma(u) \in \mathbb{Z}^n$ with each node, and constructs the BDD layer-by-layer. For simplicity, we assume without loss of generality that the variables are increasingly ordered by their indexes, i.e., $\text{var}(L_1) = x_1, \ldots, \text{var}(L_n) = x_n$. Starting from L_1, $\mathbf{r}$ is created, added to L_1, and $\sigma(\mathbf{r})$ is set to $(q_{1,1}, \ldots, q_{n,n})$. Having constructed the nodes on layers $L_1, \ldots, L_j$ and all arcs directed into nodes in those layers, L_{j+1} is constructed via extension from the nodes in L_j. Specifically, for each node $u \in L_j$, create two nodes u^0, u^1 and two arcs $a^0 = \left(u, u^0\right), a_1 = (u, u^1)$ with $d(a^0) = 0$, $d(a^1) = 1$, and $w(a_0) = 0$, $w(a^1) = \sigma(u)_j$. The states $\sigma(u^0)$, $\sigma(u^1)$ are set by index according to

$$\sigma(u^0)_k = \begin{cases} 0, & k \leq j \\ \sigma(u)_k, & k > j, \end{cases} \tag{2}$$

and

$$\sigma(u^1)_k = \begin{cases} 0, & k \leq j \\ \sigma(u)_k + q_{j,k} + q_{k,j}, & k > j. \end{cases} \tag{3}$$

After all nodes in L_j are processed, and their associated extensions are created, the states partition the nodes into equivalence classes and nodes with common states are *merge*. In particular, if any two nodes u', u'' created have the same state (i.e., $\sigma(u') = \sigma(u'')$), all arcs directed to u'' are redirected to u', and u'' is deleted. The remaining nodes become L_{j+1} and the process continues until L_{n+1}, which, by construction, will have exactly one node due to formulas in (2) and (3) making the singleton state $(0, \ldots, 0)$, and this one node becomes $\mathbf{t}$. Note that the merging operation can be done upon creation of each node, or after all nodes are created, and the work per layers can be implemented in time $O(|L_{j+1}| \cdot \log |L_{j+1}|)$ Bergman et al. (2016), and so the entire construction procedure takes time proportional to $O(n \cdot \omega(B) \cdot \log(\omega(B)))$.

The proof of correctness relies on the following claim:
Claim: $\forall u \in L_j, \forall \mathbf{r} - u$ paths $p = (a_1, \ldots, a_{j-1})$ traversing nodes $\mathbf{r} = u_1, \ldots, u_j = u$, and $\forall i \in \{1, \ldots, j\}$,

$$\sigma(u_i)_j = q_{j,j} + \sum_{i'=1}^{i-1} d(a_{i'}) \cdot \left(q_{i',j} + q_{j,i'}\right). \tag{4}$$

Fix an arbitrary node in an arbitrary layer, $u^* \in L_{j^*}$. The proof establishes the claim by induction on $i \in \{1, \ldots, j^*\}$. For the base case we show validity of the claim for $i = 1$ and $i = 2$. For $i = 1$,

$$\sigma(u_1)_1 = \sigma(\mathbf{r})_1 = q_{1,1}.$$

For $i = 2$, fix any node $u^* \in L_2$. There are three cases. If u^* has one single zero-arc directed into it from $\mathbf{r}$ (i.e., $d(a_1) = 0$), then

$$\sigma(u^*)_2 = \sigma(\mathbf{r})_2 = q_{2,2},$$

as desired. If u^* has one single one-arc directed into it from $\mathbf{r}$ (i.e., $d(a_1) = 1$), then

$$\sigma(u^*)_2 = \sigma(\mathbf{r})_2 + q_{1,2} + q_{2,1} = q_{2,2} + q_{1,2} + q_{2,1} = q_{2,2} + d(a_1) \cdot (q_{1,2} + q_{2,1}),$$

again, as desired. The last of the three cases for $i = 2$ is when u^* has two incoming arcs, both from $\mathbf{r}$. In this case $q_{1,2} + q_{2,1} = 0$, because otherwise the states arising from expanding $\mathbf{r}$ would be different, in which case the result holds.

Now, fix $i^* \leq j^* - 1$, and assume

$$\sigma(u_{i^*})_j = q_{j,j} + \sum_{i'=1}^{i^*-1} d(a_{i'}) \cdot \left(q_{i',j} + q_{j,i'}\right). \tag{5}$$

Establishing the result for $i = i^* + 1$ completes the proof of the claim. By construction,

$$\sigma(u_{i^*+1})_j = \sigma(u_{i^*})_j + d(a_{i^*}) \cdot (q_{i^*,j} + q_{j,i^*}) \tag{6}$$

so that

$$\begin{aligned}
\sigma(u_{i^*+1})_j &= q_{j,j} + \sum_{i'=1}^{i^*-1} d(a_{i'}) \cdot \left(q_{i',j} + q_{j,i'}\right) + d(a_{i^*}) \cdot (q_{i^*,j} + q_{j,i^*}) \\
&= q_{j,j} + \sum_{i'=1}^{i^*} d(a_{i'}) \cdot \left(q_{i',j} + q_{j,i'}\right),
\end{aligned}$$

as desired.

To complete the proof, we show that for any $\mathbf{r}$-$\mathbf{t}$ path $p = (a_1, \ldots, a_n)$, $x(p)^T Q x(p) = \sum_{i=1}^n w(a_i)$.

$$\begin{aligned}
x(p)^T Q x(p) &= \sum_{i=1}^{n} \sum_{i'=1}^{n} x(p)_i x(p)_{i'} q_{i,i'} \\
&= \sum_{i=1}^{n} \sum_{i'=1}^{i} x(p)_i x(p)_{i'} (q_{i,i'} + q_{i,i'}) \\
&= \sum_{i=1}^{n} x(p)_i \cdot q_{i,i} + x(p)_i \sum_{i'=1}^{i-1} x(p)_{i'} (q_{i,i'} + q_{i,i'}) \\
&= \sum_{i=1}^{n} d(a_i) \cdot q_{i,i} + d(a_i) \sum_{i'=1}^{i-1} d(a_{i'}) \cdot (q_{i,i'} + q_{i,i'}) \\
&= \sum_{i=1}^{n} d(a_i) \cdot \left(q_{i,i} + \sum_{i'=1}^{i-1} d(a_{i'}) \left(q_{i,i'} + q_{i,i'}\right) \right) \\
&= \sum_{i=1}^{n} w(a_i),
\end{aligned}$$

as desired and finishing the proof. □

5 Objective Matrix Decomposition for Limited Width BDDs

The construction algorithm in Section 4 can yield a BDD with size 2^n that represents through disjoint paths the evaluation of $x^T Q x$, for each $x \in \mathbb{B}^n$. However, by imposing certain structure on the matrix, the ensuing BDD will be provably limited in size.

To represent a general matrix Q, a decomposition $Q_1, \ldots, Q_K$ is identified so that:

1. $\sum_{k=1}^{K} Q_k = Q$; and
2. The BDD for Q_k is provably limited in size (within memory limits, or user-specified width W).

This is done for each constraint containing a quadratic term, resulting in a compact representation of the matrix evaluation at each $x \in \mathbb{B}^n$. The matrices are connected through channeling constraints both within each decomposition and across the set of decompositions. This is described Section 6.

The decomposition explored in this paper is motivated by Bergman and Cire (2016), relying on the following result. Given a graph $G = (V, E)$, a *path decomposition* of G is a sequence of subsets $V_i \subseteq V$ for which (1) $\forall e \in E, \exists V_i$ s.t. $e \subseteq V_i$ and (2) $\forall i \leq j \leq k, V_i \cap V_k \subseteq V_j$. The *width* of a path decomposition is one less that the maximum cardinality set, $\max_i |X_i| - 1$. The *path-width*, $pw(G)$, of a graph G is the minimum width over all path decompositions. For a symmetric $n \times n$ matrix Q, let $G(Q) = (V, E')$ be the graph where $V = \{1, \ldots, n\}$ with $E' = \{\{i, j\} \in E : q_{i,j} \neq 0\}$. The following result was stated but unproven in Bergman and Cire (2016):

Theorem 2 *Let Q be an $n \times n$ symmetric matrix. There exists a BDD for Q satisfying that $w(B) \leq 2^{pw(G(Q))-1}$.*

Proof. Let $V_1, \ldots, V_\gamma$ be sets that establish the path-width of $G(Q) = (V, E')$. Consider any ordering of the vertices in $G(Q)$ where the vertices in V_1 appear first, and, for $k = 2, \ldots, \gamma$, having ordered all of the vertices in $V_1, \ldots, V_{k-1}$, we arbitrarily order the vertices in $V_k \backslash \cup_{\ell=1}^{k-1} V_\ell$. Let $\tau(1), \ldots, \tau(n)$ be the ordering of the indices of the vertices. The claim is that constructing a BDD with the top-down compilation strategy in Theorem 1 using variable ordering $x_{\tau(1)}, \ldots, x_{\tau(n)}$ results in a BDD with width bounded by $2^{pw(G(Q))-1}$. By the property of the path decomposition,

$$\max\left\{i_2 - i_1 | \left\{\tau\left(i_1\right), \tau\left(i_2\right)\right\} \in E'\right)\right\} = \max_{\ell=1}^{\gamma} |V_\ell| = pw(G(Q)). \tag{7}$$

Fix an arbitrary layer j^*. If $j^* < pw(G(Q))$ the result in the statement of the theorem is trivially satisfied since any layer on a BDD is at most double the size of the previous layer and $|L_1| = 1$. Suppose now that $j^* \geq pw(G(Q))$. Pick an arbitrary node $u \in L_{j^*}$. From (4), for any $\mathbf{r}$-u path $p = (a_1, \ldots, a_{j^*-1})$ we have that for $j \geq j^*$

$$\sigma(u)_j = q_{j,j} + \sum_{i'=1}^{j^*-1} d(a_{i'}) \cdot \left(q_{\tau(i'),j} + q_{j,\tau(i')}\right), \tag{8}$$

and $\sigma(u)_j = 0$ otherwise. By (7), for $i' \leq j^* - pw(G(Q)), q_{\tau(i'),j} = q_{j,\tau(i')} = 0$. Therefore, for $j \geq j^*$

$$\sigma(u)_j = q_{j,j} + \sum_{i'=j^*-pw(G(Q))}^{j^*-1} d(a_{i'}) \cdot \left(q_{\tau(i'),j} + q_{j,\tau(i')}\right). \tag{9}$$

Let $p^1 = (a_1^1, \ldots, a_{j^*-1}^1), p^2 = (a_1^2, \ldots, a_{j^*-1}^2)$ be two arc-specified paths directed out of $\mathbf{r}$ for which $d(a_{i'}^1) = d(a_{i'}^2)$ for all $i' \in \{j^* - pw(G(Q)), \ldots, j^* - 1\}$. Let $u^1, u^2 \in L_{j^*}$ be the nodes the paths end at, respectively. $\sigma(u^1) = \sigma(u^2)$ (since the states coincide on each component) and so $u^1 = u^2$. Therefore, there can be at most $2^{pw(G(Q))-1}$ nodes on this layer, since the state of the nodes depend only on the arc domains of the paths in the immediately previous $pw(G(Q)) - 1$ layers (and not the actual values in Q), concluding the proof. □

Theorem 2 provides a mechanism for decomposing any matrix Q into a set of matrices $Q_1, \ldots, Q_k$ so that each component matrix Q_k will have a provable limited-size BDD. This decomposition is

parametrized, in that the matrices in the decomposition can be chosen so that the resulting BDDs can have arbitrarily limited maximum width, at the expenses of the number of matrices / BDDs in the decomposition.

Let W be the maximum allowable width for any BDD (it is assumed that $W = 2^\rho$, for some $\rho \in \mathbb{Z}^+$ for ease of exposition). We decompose Q into $g(W) := \left\lceil \frac{\log(W)}{n} \right\rceil$ components matrices Q_k, for $k = 1, \ldots, g(W)$ such that $\omega(B(Q_k)) \leq W$. The matrices are defined to be symmetric according to the following rule for $i \leq j$ (letting $q^k_{i,j}$ be the entry in row i and column j of matrix Q_k, and defining $q^k_{j,i} := q^k_{i,j}$):

$$q^k_{i,j} := \begin{cases} q_{i,j}, & \log(W) \cdot (k-1) \leq i < \log(W) \cdot (k) \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Theorem 3 *For any $x \in \mathbb{B}^n$*

$$\sum_{k=1}^{g(W)} x^T Q_k x = x^T Q x$$

and, for $k = 1, \ldots, g(W)$, Q_k has a BDD of width at most W.

Proof. The first result is established by definition, since $q_{i,j}$ appears in exactly one of the component BDDs and $\sum_{k=1}^{g(W)} Q_k = Q$. The second result is established by application of Theorem 2. Fix $k \in \{1, \ldots, g(W)\}$ and consider $G(Q_k)$. For $i = 1, \ldots, n$ let $V_i = \{i\} \cup \{\log(W) \cdot (k-1), \ldots, \log(W) \cdot (k)\}$. Each edge in $G(Q_k)$ is contained in every set V_i, and the pairwise intersection of every pair of the sets is $\{\log(W) \cdot (k-1), \ldots, \log(W) \cdot (k)\}$. The maximum size of any of the defined V_i is $|\{i\} \cup \{\log(W) \cdot (k-1), \ldots, \log(W) \cdot (k)\}| \leq \log(W) + 1$, as desired and concluding the proof. □

Example 2 *Consider again the matrix and BDD in Example 1. The path decomposition $V_1 = \{1, 2, 3\}, V_2 = \{2, 3, 4\}, V_3 = \{3, 4, 5\}$ establishes that $pw(G) \leq 3 - 1 = 2$. Therefore, there exists a BDD with width at most $2^2 = 4$. This is realized by using the variable ordering x_1, x_2, x_3, x_4, x_5, depicted in the BDD drawn.*

Furthermore, consider, for example, L_4. The one state listed in the BDD in this layer is $(0, 0, 0, 3, -3)$. This state only depends on x_2, x_3. For the path depicted in blue that ends at the node u that is associated with this state, the arc-domains are $(1, 0, 1)$. The other path ending at u has arc-domain $(0, 0, 1)$. Therefore, the states arising from these paths coincide, since the arc-domains on layer 2 and 3 are the same.

6 Network Flow Reformulation

We propose a linear reformulation for QCBOP based on a network-flow lifting (Bergman and Cire 2016, 2018). Our formulation contains both the original variables and the flow variables associated with the BDDs, which are used to formulate the (complicating) quadratic constraints as linear constraints in the lifted space.

Let $\mathcal{B}_h = \{B_{h1}, \ldots, B_{hK}\}$ be a collection of BDDs that generates an exact BDD decomposition for constraint $h \in \{1, \ldots, q\}$. In particular, the decomposition investigated is that obtained by decomposing Q_k based on Theorem 3 so that each has a provable limited width, and hence size. Let y_a be a binary variable that denotes the flow on arc a and let $\delta^+(u)/\delta^-(u)$ be the arcs directed out of/into node u, respectively. Recall that, for each BDD B_{hk} and for each layer j in B_{hk}, the index of the x-variable $\text{var}(L_j^{B_{hk}})$ is denoted by $\text{i}(L_j^{B_{hk}})$. We reformulate QCBOP as follows:

$$\text{NFR:} \quad \min \quad c^T x \tag{11a}$$

$$\text{s.t.} \quad \sum_{a\in\delta^+(\mathbf{r}^{B_{hk}})} y_a = \sum_{a\in\delta^-(\mathbf{t}^{B_{hk}})} y_a = 1 \qquad \forall h\in\{1,\dots,q\},\ k\in\{1,\dots,K\} \tag{11b}$$

$$\sum_{a\in\delta^+(u)} y_a = \sum_{a\in\delta^-(u)} y_a \qquad \forall h\in\{1,\dots,q\},\ k\in\{1,\dots,K\},\ u\in N^{B_{hk}}\setminus\{\mathbf{r}^{B_{hk}},\mathbf{t}^{B_{hk}}\} \tag{11c}$$

$$x_{\mathsf{i}\left(L_j^{B_{hk}}\right)} = \sum_{u\in L_j^{B_{hk}}} \sum_{a\in\delta^+(u):d(a)=1} y_a \quad \forall h\in\{1,\dots,q\},\ k\in\{1,\dots,K\},\ j\in\{1,\dots,m^{B_{hk}}\} \tag{11d}$$

$$\sum_{a\in A^{B_{h1}}\cup\cdots\cup A^{B_{hK}}} w_a y_a \geq b_h \qquad \forall h\in\{1,\dots,q\} \tag{11e}$$

$$y_a\in\{0,1\} \qquad \forall h\in\{1,\dots,q\},\ a\in A^{B_{h1}}\cup\cdots\cup A^{B_{hK}} \tag{11f}$$

$$x_i\in\{0,1\} \qquad \forall i\in\{1,\dots,n\}. \tag{11g}$$

The objective function (11a) equals the original objective function defined on the x-variables. Constraints (11b)–(11c) ensure flow balance and require one unit of flow from the root node to the terminal node in every BDD. Constraints (11d) force the x-variables to equate to the sum of the one-arcs on the layers that correspond to that variable across all BDDs, which ensures that for any choice of $x\in\mathbb{B}^n$ a corresponding consistent set of paths is selected for every BDD collection $\mathcal{B}_h$. Constraints (11e) then impose the original quadratic constraints because by construction $\sum_{a\in A^{B_{h1}}\cup\cdots\cup A^{B_{hK}}} w_a y_a = x^T Q^h x$ for every $h=1,\dots,q$. Constraints (11f)–(11g) ensure that the variables are binary. Note that constraints (11f) can be relaxed because for any choice of $x\in\mathbb{B}^n$ there is a unique consistent set of paths for each BDD collection $\mathcal{B}_h$. As a result, constraints (11d) along with constraints (11g) imply that the y-variables are binary.

Lemma 1 *NFR is equivalent to QCBOP.*

Proof. Every feasible solution, x, to QCBOP corresponds to exactly one feasible solution, (x,y), to NFR having the same objective, and vice versa. □

7 Cutting-Plane Algorithms

The size of NFR depends on the number of original constraints, the number of BDDs contained in each BDD collection, and the number of nodes and arcs in each BDD. If the total number of nodes and arcs across all the BDDs used in the decomposition is large, then NFR will be a large MIP, and may potentially be too difficult to solve. For this reason, we investigate a cutting-plane algorithm akin to Benders' decomposition, which iteratively solves considerably smaller subproblems although it may require a large number of cuts to achieve optimality, and a combined network reformulation and cutting-plane algorithm.

7.1 Defining the Master Problem

Let $\mathcal{B}_h=\{B_{h1},\dots,B_{hK}\}$ be a collection of BDDs that generates an exact BDD decomposition for constraint $h\in\{1,\dots,q\}$. With a slight abuse of notation we denote the index of the x-variable associated to an arc a by $\mathsf{i}(a)$. We follow a similar approach as Lozano and Smith (2018) by defining arc capacities that depend on the x-variables. For a given $\hat{x}\in\mathbb{B}^n$ let the capacity of arc a be $\hat{x}_{\mathsf{i}(a)}$

if $d(a) = 1$ and $1 - \hat{x}_{\mathrm{i}(a)}$ otherwise. Consider the following shortest-path problem, defined for any BDD B_{hk}, and a vector $\hat{x} \in \mathbb{B}^n$, where v^{hk} is a vector of arc-flow variables:

$$\mathrm{LSP}(B_{hk}, \hat{x}): \quad \max \quad \sum_{a \in A^{B_{hk}}} w_a v_a^{hk} \tag{12a}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(u)} v_a^{hk} - \sum_{a \in \delta^-(u)} v_a^{hk} = \begin{cases} 1, & \text{for } u = \mathbf{r}^{B_{hk}} \\ 0, & \text{for } u \in N^{B_{hk}} \setminus \{\mathbf{r}^{B_{hk}}, \mathbf{t}^{B_{hk}}\} \\ -1, & \text{for } u = \mathbf{t}^{B_{hk}} \end{cases} \tag{12b}$$

$$v_a^{hk} \leq \hat{x}_{\mathrm{i}(a)} \qquad \forall a \in A^{B_{hk}} : d(a) = 1 \tag{12c}$$

$$v_a^{hk} \leq 1 - \hat{x}_{\mathrm{i}(a)} \qquad \forall a \in A^{B_{hk}} : d(a) = 0 \tag{12d}$$

$$v_a^{hk} \geq 0 \qquad \forall a \in A^{B_{hk}}. \tag{12e}$$

Now consider the dual of $\mathrm{LSP}(B_{hk}, \hat{x})$. Let $\boldsymbol{\pi}^{hk}$ denote the dual variables associated with flow-balance constraints (12b), $\boldsymbol{\alpha}^{hk}$ denote the dual variables corresponding to arc-capacity constraints (12c), and $\boldsymbol{\beta}^{hk}$ denote the dual variables associated with arc-capacity constraints (12d). The dual of $\mathrm{LSP}(B_{hk}, \hat{x})$ is given by:

$$\mathrm{DSP}(B_{hk}, \hat{x}): \quad \min \quad \pi^{hk}_{\mathbf{r}^{B_{hk}}} + \sum_{a \in A^{B_{hk}} : d(a) = 1} \hat{x}_{\mathrm{i}(a)} \alpha_a^{hk} + \sum_{a \in A^{B_{hk}} : d(a) = 0} (1 - \hat{x}_{\mathrm{i}(a)}) \beta_a^{hk} \tag{13a}$$

$$\text{s.t.} \quad \pi_{t(a)} - \pi_{h(a)} + \alpha_a^{hk} \geq w_a \qquad \forall a \in A^{B_{hk}} : d(a) = 1 \tag{13b}$$

$$\pi_{t(a)} - \pi_{h(a)} + \beta_a^{hk} \geq w_a \qquad \forall a \in A^{B_{hk}} : d(a) = 0 \tag{13c}$$

$$\boldsymbol{\pi}^{hk} \text{ unrestricted} \tag{13d}$$

$$\boldsymbol{\alpha}^{hk}, \ \boldsymbol{\beta}^{hk} \geq \mathbf{0}, \tag{13e}$$

where we set $\pi^{hk}_{\mathbf{t}^{B_{hk}}} = 0$ because the corresponding flow-balance constraint is linearly dependent on the remaining constraints.

We now formulate the master problem for our proposed cutting-plane algorithm. Let $\Psi^h = \{(\hat{\boldsymbol{\pi}}^{h1}, \hat{\boldsymbol{\alpha}}^{h1}, \hat{\boldsymbol{\beta}}^{h1}), \ldots, (\hat{\boldsymbol{\pi}}^{hK}, \hat{\boldsymbol{\alpha}}^{hK}, \hat{\boldsymbol{\beta}}^{hK})\}$ denote a collection of dual extreme point solutions corresponding to the K dual subproblems defined for BDDs in $\mathcal{B}_h$. Define

$$\Lambda^h = \{\hat{\Psi}^h \mid \exists \hat{x} \text{ such that } (\hat{\boldsymbol{\pi}}^{hk}, \hat{\boldsymbol{\alpha}}^{hk}, \hat{\boldsymbol{\beta}}^{hk}) \in \hat{\Psi}^h \text{ is optimal to } \mathrm{DSP}(B_{hk}, \hat{x}) \ \forall k \in \{1, \ldots, K\}\} \tag{14}$$

as the set of all extreme point optimal dual solution collections for a given constraint $h \in \{1, \ldots, q\}$. We formulate our master problem as follows:

$$\text{MP:} \quad \min \quad c^T x \tag{15a}$$

$$\text{s.t.} \quad \sum_{k \in \{1, \ldots, K\}} \left(\hat{\pi}^{hk}_{\mathbf{r}^{B_{hk}}} + \sum_{a \in A^{B_{hk}} : d(a) = 1} \hat{\alpha}_a^{hk} x_{\mathrm{i}(a)} + \sum_{a \in A^{B_{hk}} : d(a) = 0} \hat{\beta}_a^{hk} (1 - x_{\mathrm{i}(a)}) \right) \geq b_h \qquad \forall h \in \{1, \ldots, q\}, \ \hat{\Psi}^h \in \Lambda^h \tag{15b}$$

$$x_i \in \{0, 1\} \qquad \forall i \in \{1, \ldots, n\}. \tag{15c}$$

Proposition 1 *MP is equivalent to QCBOP.*

Proof We first consider a feasible solution, $\hat{x}$, to QCBOP. Solution $\hat{x}$ defines a unique consistent path set for each BDD collection $\mathcal{B}_h$, denoted by $(\hat{v}^{h1}, \ldots, \hat{v}^{hK})$. Because of constraints (12c)–(12d), $\hat{v}^{hk}$ is the only feasible and therefore optimal solution to LSP$(B_{hk}, \hat{x})$ for $k = 1, \ldots, K$. By weak duality, we have that for any $\hat{\Psi}^h \in \Lambda^h$ and any $(\hat{\boldsymbol{\pi}}^{hk}, \hat{\boldsymbol{\alpha}}^{hk}, \hat{\boldsymbol{\beta}}^{hk}) \in \hat{\Psi}^h$

$$\hat{\pi}^{hk}_{\mathbf{r}^{B_{hk}}} + \sum_{a \in A^{B_{hk}}: d(a)=1} \hat{\alpha}^{hk}_a x_{\mathrm{i}(a)} + \sum_{a \in A^{B_{hk}}: d(a)=0} \hat{\beta}^{hk}_a (1 - x_{\mathrm{i}(a)}) \geq \sum_{a \in A^{B_{hk}}} w_a \hat{v}^{hk}_a. \tag{16}$$

Summing over $k \in \{1, \ldots, K\}$ yields

$$\sum_{k \in \{1,\ldots,K\}} \left(\hat{\pi}^{hk}_{\mathbf{r}^{B_{hk}}} + \sum_{a \in A^{B_{hk}}: d(a)=1} \hat{\alpha}^{hk}_a x_{\mathrm{i}(a)} + \sum_{a \in A^{B_{hk}}: d(a)=0} \hat{\beta}^{hk}_a (1 - x_{\mathrm{i}(a)}) \right) \geq \sum_{k \in \{1,\ldots,K\}} \sum_{a \in A^{B_{hk}}} w_a \hat{v}^{hk}_a. \tag{17}$$

By construction we have that

$$\sum_{k \in \{1,\ldots,K\}} \sum_{a \in A^{B_{hk}}} w_a \hat{v}^{hk}_a = \hat{x}^T Q^h \hat{x}. \tag{18}$$

Since $\hat{x}$ is feasible to QCBOP we get that $\hat{x}^T Q^h \hat{x} \geq b_h$ and conclude that

$$\sum_{k \in \{1,\ldots,K\}} \left(\hat{\pi}^{hk}_{\mathbf{r}^{B_{hk}}} + \sum_{a \in A^{B_{hk}}: d(a)=1} \hat{\alpha}^{hk}_a x_{\mathrm{i}(a)} + \sum_{a \in A^{B_{hk}}: d(a)=0} \hat{\beta}^{hk}_a (1 - x_{\mathrm{i}(a)}) \right) \geq b_h \qquad \forall h \in \{1, \ldots, q\},\ \hat{\Psi}^h \in \Lambda^h, \tag{19}$$

and thus $\hat{x}$ is feasible to MP.

Now consider that $\hat{x}$ is a feasible solution to MP and define $(\hat{v}^{h1}, \ldots, \hat{v}^{hK})$ as before. By strong duality there exists a collection of dual solutions $\hat{\Psi}^h \in \Lambda^h$ for which

$$\sum_{k \in \{1,\ldots,K\}} \left(\hat{\pi}^{hk}_{\mathbf{r}^{B_{hk}}} + \sum_{a \in A^{B_{hk}}: d(a)=1} \hat{\alpha}^{hk}_a x_{\mathrm{i}(a)} + \sum_{a \in A^{B_{hk}}: d(a)=0} \hat{\beta}^{hk}_a (1 - x_{\mathrm{i}(a)}) \right) = \sum_{k \in \{1,\ldots,K\}} \sum_{a \in A^{B_{hk}}} w_a \hat{v}^{hk}_a, \tag{20}$$

for every $h \in \{1, \ldots, q\}$. Constraints (15b) then ensure that

$$\sum_{k \in \{1,\ldots,K\}} \sum_{a \in A^{B_{hk}}} w_a \hat{v}^{hk}_a \geq b_h \quad \forall h \in \{1, \ldots, q\}, \tag{21}$$

which by construction is equivalent to

$$\hat{x}^T Q^h \hat{x} \geq b_h \quad \forall h \in \{1, \ldots, q\}. \tag{22}$$

We conclude that $\hat{x}$ is feasible to QCBOP.

Because the objective function for both problems is the same, any feasible QCBOP solution corresponds to a MP solution having the same objective, and vice versa. This completes the proof. □

7.2 A Pure Cutting-Plane Approach

We now describe our proposed cutting-plane approach in which inequalities (15b) are added iteratively. Let RMP($\mathcal{C}$) denote the *relaxed master problem*, which is defined exactly as MP but includes only cuts in set $\mathcal{C}$. We generate new cuts iteratively as described below.

Step 0: Generate a BDD collection $\mathcal{B}_h$ for every constraint $h \in \{1,\ldots,q\}$. Set $UB_0 = \infty$, $LB_0 = -\infty$, and $i = 0$. Initialize incumbent solution $\bar{x} = \mathbf{0}$. Initialize the set of cuts $\mathcal{C} = \emptyset$.

Step 1: Set $i = i+1$. Solve RMP($\mathcal{C}$). If the problem is infeasible, then terminate and conclude that the original problem is infeasible. Otherwise, record the optimal solution $\hat{x}$ found and the objective value, z^*.

Step 2: Let $LB_i = z^*$. If $\hat{x}$ is feasible to the original problem, then set $UB_i = z^*$ and update incumbent $\bar{x} = \hat{x}$. Otherwise, set $UB_i = UB_{i-1}$.

Step 3: If $LB_i = UB_i$, then terminate with an optimal solution $\bar{x}$. Otherwise, compute a collection of optimal dual solutions, $\hat{\Psi}^h$, for every $h \in \{1,\ldots,q\}$. If $\hat{x}^T Q^h \hat{x} < b_h$, then add the cut derived from $\hat{\Psi}^h$,

$$\sum_{k\in\{1,\ldots,K\}} \left(\hat{\pi}^{hk}_{\mathbf{r}^{B_{hk}}} + \sum_{a\in A^{B_{hk}}:d(a)=1} \hat{\alpha}^{hk}_a x_{\mathrm{i}(a)} + \sum_{a\in A^{B_{hk}}:d(a)=0} \hat{\beta}^{hk}_a (1 - x_{\mathrm{i}(a)}) \right) \geq b_h,$$

to $\mathcal{C}$. Return to Step 1.

The finiteness of our cutting-plane algorithm follows from the fact that there is a finite number of extreme points for each dual subproblem.

We now briefly discuss how to obtain a collection of optimal dual solutions, $\hat{\Psi}^h$, for a given vector $\hat{x}$ and $h \in \{1,\ldots,q\}$. Note that LSP($B_{hk}, \hat{x}$) is a shortest-path problem over an acyclic graph having a unique feasible (optimal) solution. We can obtain dual solutions to DSP($B_{hk}, \hat{x}$) in $\mathcal{O}(|A^{B_{hk}}|)$ steps (Ahuja et al. 1993) as descried below.

We first compute values for the π^{hk}-variables by setting π^{hk}_u equal to the shortest path length from node u to node $\mathbf{t}^{B_{hk}}$ in BDD B_{hk}, taking into account the arc flow capacities defined by constraints (12c)–(12d). This step is possible because there exists a path from every node to node t since every node has exactly two outgoing arcs, one with capacity equal to 1 and the other with capacity equal to 0. Next, we select α^{hk}-variables to satisfy $\alpha^{hk}_a \geq w_a + \pi_{h(a)} - \pi_{t(a)}$ for all $a \in A^{B_{hk}} : d(a) = 1$. If $w_a + \pi_{h(a)} - \pi_{t(a)} \leq 0$ and $d(a) = 1$, then we set $\alpha^{hk}_a = 0$. Otherwise, if $d(a) = 1$, we set $\alpha^{hk}_a = w_a + \pi_{h(a)} - \pi_{t(a)}$. Finally, we set $\beta^{hk}_a = 0$ if $w_a + \pi_{h(a)} - \pi_{t(a)} \leq 0$ and $d(a) = 0$. Otherwise, if $d(a) = 0$, we set $\beta^{hk}_a = w_a + \pi_{h(a)} - \pi_{t(a)}$.

7.3 A Combined Algorithm

We proposed a combined algorithm that reformulates some of the quadratic constraints as done in NFR via constraints (11b)–(11f) and enforces the remaining constraints using cuts (15b). The idea behind this algorithm is to balance the size of the RMP and the number of cuts needed to prove optimality. In a pure cutting-plane approach the RMP only contains cuts in set $\mathcal{C}$, which results in a manageable-sized RMP but could potentially yield weak lower bounds. On the other hand, adding some of the constraints (11b)–(11f) to the RMP could yield stronger bounds at the expense of having to solve a larger MIP at each iteration of the algorithm.

We define a *threshold*, $\tau \geq 0$, and a *deviation*, $\delta \geq 0$. Our proposed combined approach is described below.

Step 0: Start solving the problem with the pure cutting-plane algorithm. For every constraint $h \in \{1,\ldots,q\}$, record the number of corresponding cuts added, accumulated as θ_h.

Step 1: If $\theta_h = \tau$ for some $h \in \{1, \ldots, q\}$, then add constraints (11b)–(11f) associated with any $h \in \{1, \ldots, q\}$ such that $\theta_h \geq \tau - \delta$ to the RMP.
Step 2: Set $\theta_h = 0$ for all $h \in \{1, \ldots, q\}$ and return to Step 0.

Note that if $\tau = \infty$, then the combined algorithm reduces to the pure cutting-plane algorithm, and if $\tau = 0$, then the combined algorithm reduces to NFR.

8 Experimental Results

In this section we provide results from a thorough experimental evaluation of the proposed algorithms for QCBOP. All experiments were run on a machine having an Intel Xeon E5–1650 CPU (six cores) running at 3.60 GHz with 32 GB of RAM on Windows 10. We coded our algorithms in Java using Eclipse SDK version 4.7.1. and solved all optimization problems using CPLEX 12.7.1 with a time limit of four hours (14, 400s).

We compare four algorithms over the entire set of instances. The first is the BDD-based network flow reformulation presented in (11), denoted by "`BDD`". The second is the pure cutting-plane algorithm described in Section 7.2, denoted by "`BDDCuts`". The third algorithm, denoted by "`BDD+`", is the combined reformulation and cuts approach described in Section 7.3. The fourth is the original problem formulation presented in (1) solved using CPLEX quadratic solver with the default settings, denoted by "`CPLEX`".

We coded `BDDCuts` and `BDD+` algorithms using a branch-and-cut implementation (via callbacks) in which cuts are added throughout the branch-and-bound search, solving the subproblems each time that an integer solution is found. We adapted the first cut strengthening strategy from Lozano and Smith (2018) and defined a simple variable ordering rule for the BDD construction that sorts the variables in decreasing order according to the number of nonzero coefficients in each constraint. After tuning the algorithms parameters, we set the width for the BDD decomposition to 4, threshold τ to 100, and deviation δ to 10.

8.1 Instances

Tests were run on two sets of instances. The first is a synthetic and randomly generated benchmark set which we refer to as **S**, used to provide an analysis of how the algorithm works on a wide-range of problems with different instance characteristic. The second is a benchmark set which we refer to as **B**, used to evaluate the performance of the algorithm on previously introduced instances from the literature. Specifically:
S: The *synthetic* instances are generated as follows. An instance is parameterized by the number of variables n, the number of quadratic constraints q, and the density of the constraints matrices d. We generate five random instances for each configuration of $n \in \{25, 50, 75, 100\}$, $q \in \{10, 30, 50\}$, and $d \in \{0.01, 0.05, 0.1\}$. Objective function coefficients c_i are drawn independently and uniformly at random from a discrete uniform $U(1, 50)$. For each constraint matrix Q^h, coefficients $q^h_{i,j}$ are independently set to 0 with a probability of $1-d$ for $i = 1, \ldots, n$, $j = i, \ldots, n$ to achieve a constraint matrix density of d. Nonzero coefficients $q^h_{i,j}$ are drawn independently and uniformly at random from a discrete uniform $U(1, 25)$ and then we set $q^h_{j,i} = q^h_{i,j}$ to ensure symmetry. Finally, we set the right-hand side coefficients to $b_h = \sum_{i=1}^{n} \sum_{j=i}^{n} \lambda_{ij} q^h_{ij}$, where λ_{ij} values are drawn independently from a uniform $U(0, 1)$ distribution. The resulting 180 instances and the code used to generate them are provided as an online supplement to this paper.
B: The *benchmark* set is taken from a recently compiled `QPLIB`, available at http://qplib.zib.de/instances.html. We use all available pure binary instances with a linear objective function (a total of 9 instances).

8.2 Performance Comparison

We provide a comparison of solution times and objective function bounds for the four algorithms mentioned above. Figure 2 is divided into two sections, the left portion reporting execution time in seconds and the right portion reporting ending gap at 14,400 seconds. The y-axis reports the number of instances either solved by the corresponding time, or with an optimality gap less than or equal to the x-coordinate, respectively. The four lines correspond to `BDD`, `BDDCuts`, `BDD+`, and `CPLEX`.

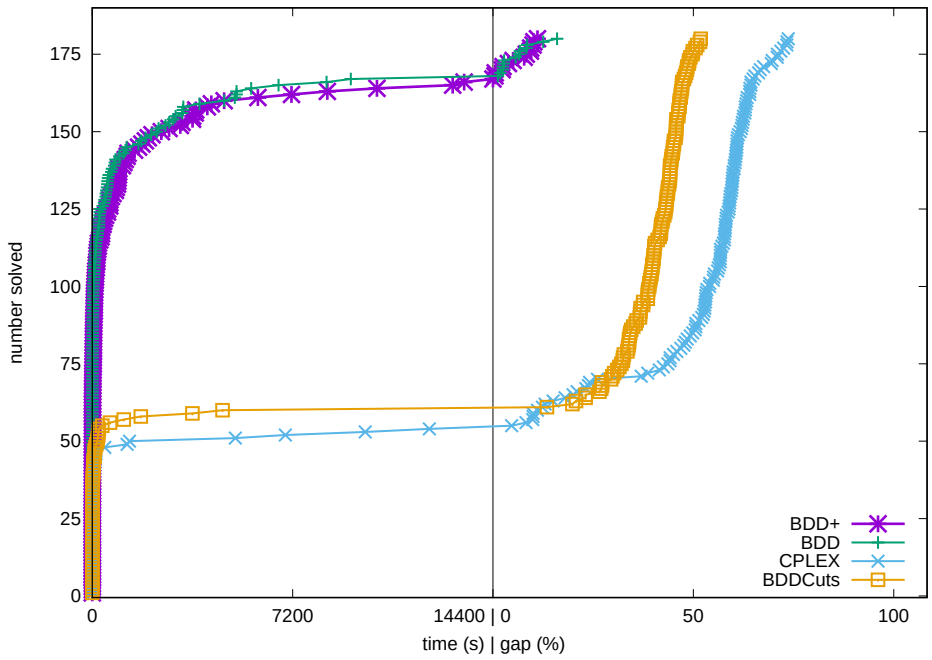

Figure 2: Cumulative distribution plot of performance comparing `BDD`, `BDDCuts`, `BDD+`, and `CLPEX` on instances in set **S**.

This plot readily shows the significant dominance of `BDD+` and `BDD` over `CPLEX` and `BDDCuts` on this set of instances. In particular, at any given time windows below $14,400$ seconds, `BDD+` and `BDD` solves many more instances than the other two algorithms. In aggregate, both `BDD` and `BDD+` solve 167 instances, while `BDDCuts` and `CPLEx` both solve only 60 and 54 instances, respectively, within the time limit. Additionally, the ending gaps for instances that are unsolved are also much smaller for `BDD+` and `BDD`, and are slightly smaller for `BDDCuts` than `CPLEX`. For instances not solved by any of the approaches, the ending gaps for `BDD+` are no more than 0.11 while for `CPLEX`, ending gaps can be as large as 0.74, exhibiting that even for very hard instances, `BDD+` provides significant improvements over `CPLEX`.

Figure 3 presents a more detailed instance-by-instance comparison between `BDD+` and `CPLEX`. This scatter plot contains a point for every instance. The size of the point provides a visual depiction of the number of variables (larger points correspond to larger instances) the color corresponds to the density, and the shape indicates the number of quadratic constraints. The coordinates correspond to the runtime for the algorithms, with points above the diagonal indicating instances for which `BDD+` had shorter solution times than `CPLEX`.

This plot shows that `BDD+` outperforms `CPLEX` on every single instance in set **S**. Several instances

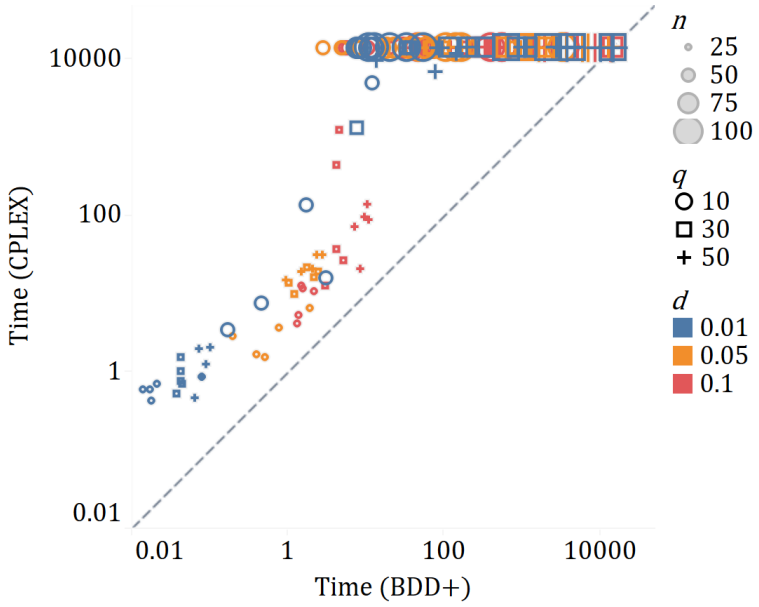

Figure 3: Scatter plots comparing runtime differences between BDD+ and CPLEX on instances in **S**.

that CPLEX could not solve within the time limit were solved by BDD+ in less than 5 seconds. On those instances solved by CPLEX and BDD+ in the time limit, runtime speedups (calculated as the ratio between CPLEX runtime and BDD+ runtime) range from 2.5 to 722, with an average speedup of 50.59. The difference in performance grows drastically larger with the number of variables in the instance.

The difference in performance between BDDCuts and the other BDD techniques is clear, but the difference between BDD and BDD+ is harder to identify. A closer looks reveals that as d grows, the performance of BDD+ increases, i.e., the effectiveness of the cuts grows. BDD solves 3 more instances than BDD+ when $d = 0.01$ while BDD+ solves 3 more instances than BDD when $d \geq 0.05$.

Figure 4 provides a scatter plot comparing BDD and BDD+. The plot indicates a slight increase in the effectiveness of the cuts as the density grows. For those instances with 100 variables, in the 11 instances commonly solved by both techniques within 14400 seconds with $d = .01$, BDD solves each faster, with an average speedup of 5.22. Contrastingly, for the 9 instances solved by both techniques within 14400 seconds with 100 variables with $d = 0.1$, BDD+ solves 6 faster. This indicates that for large dense instances, the cuts become effective and should be used.

In order to test this hypothesis, we generated an additional set of instances with $n = 50, q = 30$, and $d \in \{0.2, 0, 3, 0.4, 0.5, 0.6\}$ (5 instances per configuration). Figure 5 depicts the results in a line plot, where for every density of instances generated we plot the average runtime for both algorithms and linearly interpolate between. This plot confirms the hypothesis that as the density of the constraint matrices increase, the cutting planes become relatively more effective and BDD+ outperforms BDD. On this set of instances, BDD+ is faster than BDD in 18 out of 25, and has an average relative speedup of 4.4 (and up to 16.5).

Additional detailed solution statistics for **S** are provided in the Appendix.

The relative superiority of BDD and BDD+ over CPLEX is corroborated on an application to the instances in **B**. Table 1 reports the results for these experiments. The first four columns present the instance name, the number of variables, the number of quadratic constraints, and the number of linear constraints, respectively. The remaining columns show the CPU runtime in seconds (Time), the optimality gap (as reported by the optimizer) for the instances that are not solved within the

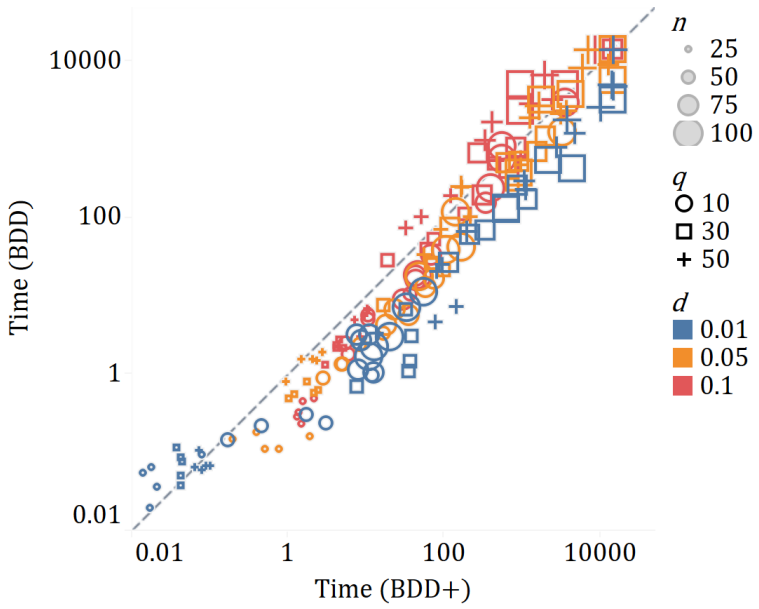

Figure 4: Scatter plots comparing runtime differences between `BDD+` and `BDD` on instances in **S**.

time limit (Gap), the number of cuts added (#Cuts), and the total number of BDDs used in the decomposition (#BDDs).

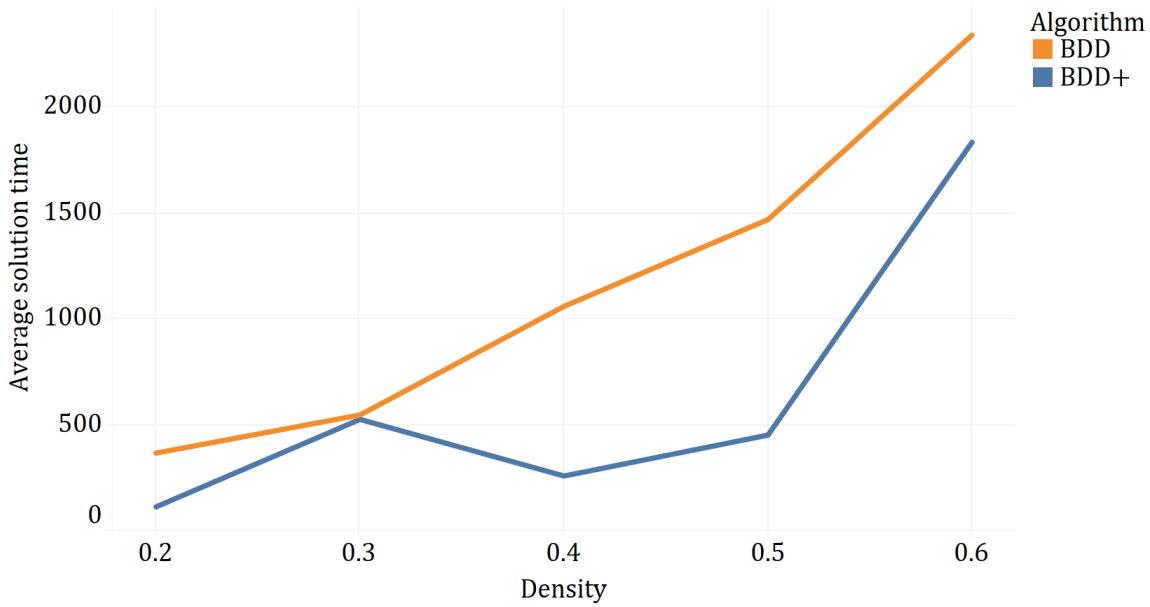

Figure 5: Line plot comparing runtime differences between `BDD+` and `BDD` on random instances with varying density.

Table 1: Comparing BDD, BDDCuts, BDD+, and CPLEX on instances in **B**

Name	n	q	l	BDD		BDD+			BDDCuts			CPLEX		#BDDs
				Time	Gap (%)	Time	Gap (%)	#Cuts	Time	Gap (%)	#Cuts	Time	Gap (%)	
2047	136	17	2040	89	0	408	0	1340	2504	0	1682	226	0	136
2055	153	18	2448	174	0	330	0	1498	435	0	8355	1668	0	144
2060	171	19	2907	1495	0	2021	0	1561	6078	0	20400	>14400	9	171
2067	190	20	3420	251	0	353	0	1564	319	0	3881	558	0	180
2073	210	21	3990	9096	0	>14400	3	1696	>14400	12	2391	>14400	9	210
2077	231	22	4620	>14400	2	>14400	4	1847	>14400	11	1211	>14400	11	220
2085	253	23	5313	>14400	6	>14400	11	2066	>14400	15	3698	>14400	17	253
2087	276	24	6072	556	0	509	0	1267	578	0	133	51	0	264
2096	300	25	6900	>14400	6	>14400	11	2127	>14400	16	1721	>14400	16	300

For this set of instances BDD is the best performer, solving 6 out of 9 instances within the time limit. Moreover, in 9 out of 9 instances, BDD solves the instance faster than CPLEX, or ends with a considerably tighter optimality gap. The speedups achieved by BDD when compared to CPLEX are up to 9.6, with an average speedup of 3.2.

The second best performer is BDD+, which solves 5 out 9 instances within the time limit and marginally faster on average than BDDcuts, which also solves 5 instances. CPLEX only solves 4 out of 9 instances within the time limit. Regarding the number of cuts, BDD+ adds on average considerably fewer cuts than BDDCuts (e.g., see instance 2060) since the MP formulation is stronger for BDD+. However, for these instances BDD performs better than BDD+, which supports the idea of the existing trade-off between the size of the RMP and the number of cuts needed to prove optimality (stated in Section 7.3) and suggests that we should investigate further strengthening strategies for the cuts.

9 Discussion and Future Work

We propose a generic framework for reasoning over quadratically constrained binary optimization problems using a hybrid decision diagram / integer programming model. The benefit of the approach is exhibited by application to benchmark and synthetic instances, providing substantial improvement over state-of-the-art commercial integer programming solvers.

There are several important extensions that we anticipate resulting from this work. Our results were run using a simple heuristic for ordering the variables in the BDDs. Variable ordering can have a significant impact on the quality of BDDs in optimization (Bergman et al. 2012) and investigating this thoroughly can further strengthen the quality of the obtained results.

Another extension is to problems with quadratic objective function as well. As stated in the introduction, the application of the reformulation presented in this paper can be applied in the presence or absence of quadratic objective terms, whether or not the reformulation is equally effective when the objective function also has quadratic terms would be interesting to investigate.

Perhaps the most critical extension is to investigate other constraints (linear or nonlinear) that can benefit from decomposed reformulations. The approach suggested is generic and can be implemented automatically for quadratic constraints, but it also lends itself to more general terms that might benefit from a similar decomposition scheme. Given any decomposition of problem constraints, the algorithm for optimizing over the collection of decision diagrams that is presented in this paper can be readily applied, but with the caveat that there is no general BDD construction procedure that guarantees limited-sized BDDs. An open line of research relates to the development of BDD decomposition schemes for constraints having different functional forms, as well as to the implementation details for specific applications, which may require a substantial amount of attention.

References

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, NJ, 1993.

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 Efficient and Experimental Algorithms (WEA 05)*, volume 3503 of *Lecture Notes in Computer Science*, pages 452–463. Springer, 2005.

M. Behle and F. Eisenbrand. 0/1 vertex and facet enumeration with BDDs. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 158–165, 2007.

David Bergman and Andre A. Cire. Decomposition based on decision diagrams. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 45–54, Cham, 2016. Springer International Publishing. ISBN 978-3-319-33954-2.

David Bergman and Andre A. Cire. Discrete nonlinear optimization by state-space decompositions. *Management Science*, 0(0):null, 2018. doi: 10.1287/mnsc.2017.2849. URL https://doi.org/10.1287/mnsc.2017.2849.

David Bergman, Willem-Jan van Hoeve, and John N. Hooker. Manipulating mdd relaxations for combinatorial optimization. In Tobias Achterberg and J. Christopher Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 20–35, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-21311-3.

David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John N. Hooker. Variable ordering for the application of bdds to the maximum independent set problem. In *Proceedings of the 9th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, CPAIOR'12, pages 34–49, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-29827-1. doi: 10.1007/978-3-642-29828-8_3. URL http://dx.doi.org/10.1007/978-3-642-29828-8_3.

David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014. doi: 10.1287/ijoc.2013.0561. URL https://doi.org/10.1287/ijoc.2013.0561.

David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016. doi: 10.1287/ijoc.2015.0648. URL https://doi.org/10.1287/ijoc.2015.0648.

Alain Billionnet, Sourour Elloumi, and Marie-Christine Plateau. Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: The qcr method. *Discrete Appl. Math.*, 157(6):1185–1197, March 2009. ISSN 0166-218X. doi: 10.1016/j.dam.2007.12.007. URL http://dx.doi.org/10.1016/j.dam.2007.12.007.

Alain Billionnet, Sourour Elloumi, and Amélie Lambert. An efficient compact quadratic convex reformulation for general integer quadratic programs. *Computational Optimization and Applications*, 54(1):141–162, Jan 2013. ISSN 1573-2894. doi: 10.1007/s10589-012-9474-y. URL https://doi.org/10.1007/s10589-012-9474-y.

Robert E. Bixby. A brief history of linear and mixed-integer programming computation, 2012.

Christoph Buchheim and Angelika Wiegele. Semidefinite relaxations for non-convex quadratic mixed-integer programming. *Mathematical Programming*, 141(1-2):435–452, 2013.

Christoph Buchheim, Alberto Caprara, and Andrea Lodi. An effective branch-and-bound algorithm for convex quadratic integer programming. *Mathematical Programming*, 135(1):369–395, Oct 2012. ISSN 1436-4646. doi: 10.1007/s10107-011-0475-x. URL https://doi.org/10.1007/s10107-011-0475-x.

Samuel Burer and Adam N. Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97 – 106, 2012. ISSN 1876-7354. doi: https://doi.org/10.1016/j.sorms.2012.08.001. URL http://www.sciencedirect.com/science/article/pii/S1876735412000037.

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

Fabio Furini, Emiliano Traversi, Pietro Belotti, Antonio Frangioni, Ambros Gleixner, Nick Gould, Leo Liberti, Andrea Lodi, Ruth Misener, Hans Mittelmann, Nick Sahinidis, Stefan Vigerske, and Angelika Wiegele. QPLIB: A library of quadratic programming instances. Technical report, February 2017. URL http://www.optimization-online.org/DB_HTML/2017/02/5846.html. Available at Optimization Online.

Wolfgang Gatterbauer and Dan Suciu. Oblivious bounds on the probability of boolean functions. *ACM Transactions on Database Systems (TODS)*, 39(1):5, 2014.

Fred Glover. Improved linear integer programming formulations of nonlinear integer problems. *Management Science*, 22(4):455–460, 1975. doi: 10.1287/mnsc.22.4.455. URL https://doi.org/10.1287/mnsc.22.4.455.

Fred Glover and Eugene Woolsey. Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22(1):180–182, 1974. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/169227.

Nicholas I. M. Gould. A quadratic programming bibliography. Technical report, 2000. URL http://www.optimization-online.org/DB_FILE/2001/02/285.pdf.

Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2018. URL http://www.gurobi.com.

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.

Utz-Uwe Haus and Carla Michini. Compact representations of all members of an independence system. *Annals of Mathematics and Artificial Intelligence*, pages 1–18, 2016.

Utz-Uwe Haus and Carla Michini. Compact representations of all members of an independence system. *Annals of Mathematics and Artificial Intelligence*, 79(1-3):145–162, 2017.

Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81, Jul 2014. ISSN 1573-2886. doi: 10.1007/s10878-014-9734-0. URL https://doi.org/10.1007/s10878-014-9734-0.

Leonardo Lozano and J. Cole Smith. A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs. *Mathematical Programming*, 2018. doi: 10.1007/s10107-018-1315-z. URL https://doi.org/10.1007/s10107-018-1315-z.

Leonardo Lozano, David Bergman, and J. Cole Smith. On the Consistent Path Problem. *Optimization Online e-prints*, April 2018.

Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Bdd-constrained search: A unified approach to constrained shortest path problems. In *AAAI*, pages 1219–1225, 2015.

IBM ILOG CPLEX Optimizer. Ibm ilog cplex optimizer, 2018. URL https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer.

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

Guillaume Perez and Jean-Charles Régin. Constructions and in-place operations for mdds based constraints. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 279–293. Springer, 2016.

10 Appendix

10.1 Detailed Solution Statistics on S

Tables 2–5 provide detailed solution statistics for the computational experiments on instances in **S**. The first two columns present the number of variables and the number of quadratic constraints, respectively. The remaining columns show the average CPU runtime in seconds (Avg) computed only among the instances solved within the time limit (the number of instances solved within the time limit is displayed in parentheses), the optimality gap for the instances that are not solved within the time limit (Gap), the number of cuts added (#Cuts), and the total number of BDDs used in the decomposition (#BDDs).

Table 2: Comparing BDD, BDDCuts, BDD+, and CLPEX on instances in **S** having $d = 0.01$

n	q	BDD		BDD+			BDDCuts			CPLEX		#BDDs
		Avg	Gap (%)	Avg	Gap (%)	#Cuts	Avg	Gap (%)	#Cuts	Avg	Gap (%)	
25	10	0.1 (5)	0	0.1 (5)	0	26	0.1 (5)	0	26	0.6 (5)	0	24
25	30	0.1 (5)	0	0.1 (5)	0	56	0.1 (5)	0	55	0.9 (5)	0	70
25	50	0.1 (5)	0	0.1 (5)	0	83	0.1 (5)	0	81	1 (5)	0	115
50	10	0.4 (5)	0	3 (5)	0	378	15 (5)	0	274	1061 (5)	0	67
50	30	3 (5)	0	30 (5)	0	948	437 (5)	0	823	1335 (1)	13	209
50	50	12 (5)	0	82 (5)	0	1932	2038 (5)	0	1438	9619 (3)	9	359
75	10	2 (5)	0	10 (5)	0	964	- (0)	22	3627	- (0)	57	132
75	30	120 (5)	0	542 (5)	0	2369	- (0)	33	6065	- (0)	47	402
75	50	521 (5)	0	1981 (5)	0	3653	- (0)	37	6938	- (0)	48	677
100	10	5 (5)	0	26 (5)	0	981	- (0)	33	6470	- (0)	72	208
100	30	906 (5)	0	1921 (4)	0	2758	- (0)	43	9533	- (0)	62	633
100	50	3562 (4)	1	6964 (2)	2	4433	- (0)	43	10319	- (0)	55	1057

Table 3: Comparing BDD, BDDCuts, BDD+, and CPLEX on instances in **S** having $d = 0.05$

n	q	BDD		BDD+			BDDCuts			CPLEX		#BDDs
		Avg	Gap (%)	Avg	Gap (%)	#Cuts	Avg	Gap (%)	#Cuts	Avg	Gap (%)	
25	10	0.1 (5)	0	0.8 (5)	0	279	1 (5)	0	254	3 (5)	0	62
25	30	0.6 (5)	0	2 (5)	0	755	4 (5)	0	711	16 (5)	0	188
25	50	2 (5)	0	2 (5)	0	937	4 (5)	0	917	24 (5)	0	311
50	10	2 (5)	0	8 (5)	0	908	- (0)	28	8361	- (0)	58	166
50	30	70 (5)	0	200 (5)	0	2531	- (0)	35	12634	- (0)	59	493
50	50	187 (5)	0	296 (5)	0	3992	- (0)	38	13852	- (0)	52	827
75	10	9 (5)	0	42 (5)	0	966	- (0)	39	8210	- (0)	62	287
75	30	589 (5)	0	1038 (5)	0	2727	- (0)	45	12712	- (0)	64	848
75	50	3178 (5)	0	4249 (5)	0	4381	- (0)	46	15223	- (0)	61	1413
100	10	291 (5)	0	737 (5)	0	932	- (0)	43	6432	- (0)	23	401
100	30	3300 (4)	1	5309 (4)	2	2764	- (0)	46	11527	- (0)	62	1208
100	50	5591 (2)	3	4923 (3)	3	4441	- (0)	46	14251	- (0)	63	2015

Table 4: Comparing BDD, BDDCuts, BDD+, and CPLEX on instances in **S**

n	q	BDD		BDD+			BDDCuts			CPLEX		#BDDs
		Avg	Gap (%)	Avg	Gap (%)	#Cuts	Avg	Gap (%)	#Cuts	Avg	Gap (%)	
25	10	0.4 (5)	0	2 (5)	0	795	27 (5)	0	1540	8 (5)	0.00	87
25	30	2 (5)	0	4 (5)	0	2136	83 (5)	0	3652	355 (5)	0.00	259
25	50	5 (5)	0	9 (5)	0	3412	197 (5)	0	6354	83 (5)	0.00	431
50	10	5 (5)	0	14 (5)	0	962	- (0)	33	11357	- (0)	48.00	203
50	30	144 (5)	0	155 (5)	0	2920	- (0)	40	16514	- (0)	54.00	611
50	50	302 (5)	0	328 (5)	0	4731	- (0)	42	19416	- (0)	53.00	1022
75	10	46 (5)	0	105 (5)	0	995	- (0)	44	9346	- (0)	47.00	330
75	30	524 (4)	0	522 (4)	1	3019	- (0)	44	14528	- (0)	59.00	988
75	50	2191 (4)	2	1127 (4)	2	4726	- (0)	47	18110	- (0)	58.00	1645
100	10	932 (5)	0	1027 (5)	0	999	- (0)	43	7103	- (0)	17.00	454
100	30	4248 (3)	2	1823 (3)	4	2955	- (0)	49	12262	- (0)	61.00	1363
100	50	6692 (1)	5	7905 (3)	2	4856	- (0)	50	15363	- (0)	60.00	2273

Table 5: Comparing BDD and BDD+ on instances with higher density

d	BDD	BDD+	
	Avg	Avg	#Cuts
0.2	368 (5)	114 (5)	2844
0.3	548 (5)	527 (5)	2824
0.4	1061 (5)	259 (5)	2849
0.5	1471 (5)	453 (5)	2872
0.6	2342 (5)	1836 (5)	2830