Effects of Disjunctive Conic Cuts within a Branch and Conic Cut Algorithm to Solve Asset Allocation Problems

Sertalp B. Çay, Julio C. Góez, and Tamás Terlaky

Department of Industrial and Systems Engineering

Lehigh University, Bethlehem, PA, USA

Department of Business and Management Science

NHH Norwegian School of Economics, Bergen, Norway

ISE Technical Report 16T-005

Effects of Disjunctive Conic Cuts within a Branch and Conic Cut Algorithm to Solve Asset Allocation Problems

Sertalp B. Çay[a], Julio C. Góez[b], and Tamás Terlaky[a]

[a]Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA

[b]Department of Business and Management Science, NHH Norwegian School of Economics, Bergen, Norway

May 30, 2016

Abstract

Recently, Mixed Integer Second Order Cone Optimization (MISOCO) has gained attention. This interest has been driven by the availability of efficient and mature methods to solve second order cone optimization (SOCO) problems and the wide range of applications of MISOCO. Financial optimization is an important application of MISOCO, where the variants of Markowitz' classical mean-variance portfolio optimization problem leads to MISOCO problems. In this work we show that the recently developed methodology of Disjunctive Conic Cuts (DCC) and Disjunctive Cylindrical Cuts (DCyC) provides a powerful tool to solve variations of portfolio problems. Our aim is to contribute to narrowing the gap between theoretical developments and practical implementation. We analyze the effect of these cuts on portfolio optimization problems within a Branch and Conic Cut (BCC) framework. The proposed methodology shows that DCCs and DCyCs are effective in practical settings.

Keywords — mixed integer conic optimization, branch and conic cut, computational optimization, portfolio optimization, asset allocation optimization

1 Introduction

The interest in mixed integer second order cone optimization (MISOCO) has been growing recently both in academia and industry. An important driver of this interest in MISOCO is the availability of several powerful optimization packages which include second order cone optimization (SOCO) solvers [3, 21, 23]. These solvers have profited from the recent theoretical developments to provide efficient and accurate software that are capable of tackling large scale optimization problems. Supported with these new capabilities, practitioners and academics are realizing the potential of MISOCO in real-life applications. As a result, sectors like aviation, healthcare, energy and finance [2, 8, 12, 24] have found MISOCO as an alternative to build more accurate mathematical models. Undoubtedly, this application-driven community has helped to speed up the theoretical and computational developments on the MISOCO technology. However, a gap exists between what the solvers offer today and the methodological advances that we have recently achieved. To close the gap between theoretical developments and practical implementations, we studied the effectiveness of recently developed Disjunctive Conic Cuts and Disjunctive Cylindrical Cuts [9, 10, 11] for MISOCO. More

Email addresses: `sertalpbilal@gmail.com` (Sertalp B. Çay), `julio.goez@nhh.no` (Julio C. Góez), `terlaky@lehigh.edu` (Tamás Terlaky) specifically, We applied these cuts to the asset allocation problems (AAPs) in a branch and conic cut (BCC) framework.

Before discussing the specifics of AAPs, we need to introduce the MISOCO problem and briefly review the fundamental conic cuts ideas and results. A general form MISOCO problem can be presented as follows:

$$\begin{array}{rrcl} \text{minimize:} & c^\top x & & \\ \text{subject to:} & Ax & = & b, \\ & x & \in & \mathcal{K}, \\ & x & \in & \mathbb{Z}^d \times \mathbb{R}^{n-d}, \end{array}$$

where $A \in \mathbb{R}^{m\times n}$ is a full rank matrix, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. Also, $\mathcal{K}$ is a Cartesian product of second order cones of various dimensions, i.e., $\mathcal{K} = \mathbb{L}^{n_1} \times \mathbb{L}^{n_2} \times \cdots \times \mathbb{L}^{n_k}$, where $\mathbb{L}^{n_i} = \{x^i \in \mathbb{R}^{n_i} \mid x^i_1 \geq \|x^i_{2:n_i}\|\}$, for $i = 1, \ldots, k$, with $\sum_{i=1}^k n_i = n$ for $x = \left((x^1)^\top, (x^2)^\top, \ldots, (x^k)^\top\right)^\top$, and $x^i \in \mathbb{R}^{n_i}$.

The motivation for this work comes from the recent methodological developments in the MISOCO field. These developments have shown promising results to provide faster solutions to MISOCO problems, which is of great interest for practitioners. Most of the theoretical effort on the MISOCO field has focused on generating valid inequalities. Recent developments include cuts for 0-1 conic optimization [18], conic mixed-integer rounding cuts [5], intersection cuts for mixed-integer conic quadratic sets [4], separation for conic knapsack constraints [7], split cuts for mixed-integer conic quadratic problems [27], and disjunctive cuts for cross-sections of the second-order cones [32]. Finally, one of the most significant development for MISOCO solution algorithms is the discovery of Disjunctive Conic Cuts (DCC) and Disjunctive Cylindrical Cuts (DCyC) [9, 10, 11, 20]. In general, a conic cut is a cone which cuts off some non-integer solutions but none of the feasible integer solutions. Under mild assumptions, adding DCCs and DCyCs gives the convex hull of a closed convex set and its intersection with a linear disjunction. These are shown to be the tightest cuts that can be generated for such disjunctions.

Beyond the theoretical benefits of DCCs and DCyCs to obtain tighter formulations, providing strategies for efficient implementation has utmost importance for research on solution techniques. For this reason, we are interested in exploring the effect of DCCs and DCyCs on real-life applications. Recent work has laid down strong methodological foundations for this family of cuts for MISOCO problems. However, there is much less research on implementation challenges and the effects of DCCs and DCyCs in practice within a full MISOCO solution framework. Even for general conic cuts, previous studies does not provide enough computational results. In these studies cuts are added only at or very close to the root note [5, 18] or not implemented at all [4, 6]. Turning our interest to DCCs, Góez [20] provides preliminary results where DCCs are added both in the root node and inside the BCC tree. Presented by Bonami et al. [16], constrained layout problems have convex quadratic constraints, which can be written as a second order cone. Góez [20] showed that binary variables in these convex quadratic constraints can be used as a disjunction to generate DCCs, which are tighter than the original constraint. Generating these DCCs and replacing the original constraints with them as a preprocessing step reduced the solution time significantly in almost all cases. The author also presented experiments with randomly generated problems, where DCCs are added within the BCC tree by using simple decisions. Promising results of these experiments motivated us to investigate further for a better management of DCCs within a BCC framework.

Our goal in this paper is to enhance the understanding on how to use the novel DCC technology. In general, one can generate DCCs and DCyCs by using the disjunctions obtained by the interim solutions in a BCC algorithm. We demonstrate the effectiveness of DCCs and DCyCs within a BCC framework. For this purpose, we revisit the design of most major components of a BCC framework: branching, cutting and cut management. Also, we explore different decisions in these elements to maximize the benefit gained from the DCCs and DCyCs.

As we have already mentioned, we limit the scope of this study to special classes of AAPs, which are variations of Markowitz' classical portfolio optimization model [26]. We study an AAP formulation that minimizes the risk of the portfolio, while satisfying a prescribed return level. The practical setting of this problem for different markets requires some additional constraints. For instance, in the business market, big companies often buy stocks in large batches for liquidity [15]. Such constraints require integer variables in the optimization model. Bonami and Lejeune [15] and Saglam [29] reviewed various similar constraints which arise in portfolio optimization that require integer and binary variables. In all these cases the common denominator is that the resulting optimization models are MISOCO problems. In this paper, we study four of these MISOCO problems. These results can be extended to other problems formulated as MISOCO. Benefits of using these cuts in a BCC framework can be observed as lower number of nodes in the BCC tree, lower total solution time, and higher numerical accuracy in some cases.

The rest of this paper is structured as follows. In Section 2, we provide background information about mixed integer AAP models. This is followed by the description of our BCC framework and the cut generation process in Section 3. Cut addition, branching, and searching strategies for our BCC framework are also discussed in Section 3. We present our experiments, computational results, and their interpretation in Section 4. Finally, we summarize our findings in Section 5.

2 MISOCO for AAPs

The classical Markowitz' mean-variance portfolio optimization problem for n risky assets is formulated as follows:

$$\begin{array}{lrcll} \text{minimize:} & x^\top \Sigma x & & & \\ \text{subject to:} & \mu_0 x_0 + \mu^\top x & = & r & \\ & x_0 + \sum_{i=1}^{n} x_i & = & 1 & \\ & x_i & \geq & 0 & i = 1, \dots, n, \end{array} \tag{1}$$

where $\Sigma \in \mathbb{R}^{n\times n}$ is a positive definite variance-covariance matrix, r is the desired return level from the investment, $\mu \in \mathbb{R}^n$ is a vector of the expected returns μ_i for each risky asset, and $x \in \mathbb{R}^n$ represents the proportion of the capital invested in each risky asset. Also, x_0 and μ_0 denote the fraction of capital invested in the money market and its expected return level, respectively. Money market investment is usually a low-return but risk-free investment type [15].

The AAP is a variation of the portfolio optimization problem, where one allocates a given capital into various types of investments, such as equities, bonds, currencies, and cash to achieve a certain return level while minimizing the risk. Notice that until now there are no discrete variables in the problem formulation. In the rest of this section we describe three different AAP formulations which require discrete variables in the problem formulation, as shown by Bonami and Lejeune [15], Cornuejols and Tütüncü [19], and Saglam [29].

2.1 Round-lot constrained AAP

Our first type of mixed integer AAP formulation appears due to a special set of constraints, called round-lot or minimum transaction lot constraints [15, 25]. In financial settings, large investors often buy risky assets in large batches which are called even lots. This strategy corresponds with the idea that in the business market trading even lots is easier than trading small batches.

Before presenting the modeling we need to define the relation between the even lots and the vector x. Denote κ_i as the fixed batch size and ν_i as the market value of each asset i. Let d denote the total capital, and define z_i as the number of even lots to buy of asset i. Then, the round-lot relationship between x_i and z_i is defined as

$$x_i = \frac{\nu_i \kappa_i}{d} z_i, \quad i = 1, \dots, n, \tag{2}$$

which defines the proportion of capital invested in asset i in terms of z_i. When discussing this problem Bonami and Lejeune [15] simplified the presentation considering a common batch size for all assets.

After adding the round-lot restriction (2) to the models, the expected return may not be satisfied as an equality. However, one can still aim at a return level r that is a lower bound for the selected portfolio. As a result, we can write the problem as

$$\begin{array}{llcll} \text{minimize:} & x^\top \Sigma x & & & \\ \text{subject to:} & \mu_0 x_0 + \mu^\top x & \geq & r & \\ & x_0 + \sum_{i=1}^{n} x_i & = & 1 & \\ & x_i & = & \frac{\nu_i \kappa_i}{d} z_i & i = 1, \ldots, n \\ & 0 \leq x_i & \leq & 1 & i = 1, \ldots, n \\ & z & \in & \mathbb{Z}_+^n. & \end{array} \tag{3}$$

Here the constraints $x_i \leq 1, i = 1, \ldots, n$ are redundant, but they are required when allowing shorting operation. Shorting is a common practice, where investors can sell financial instruments that are not owned by them, and purchase them at a later time. This practice is being applied if the investor expects decrease in value of the investment. In (3) one can allow shorting by dropping the non-negativity on x, the investment fractions, and on z, the number of batches purchased.

We may simplify the discussion further by using (2) to express problem (3) all in terms of z. Let $a \in \mathbb{R}^n$ be defined as

$$a_i = \frac{\nu_i \kappa_i}{d}, \quad i = 1, \ldots, n,$$

which is the constant batch size for investment in asset i due to the round-lot constraint. Let $\hat{\Sigma} = \operatorname{diag}(a)^\top \cdot \Sigma \cdot \operatorname{diag}(a)$, and allow us to introduce a new variable t to move the quadratic objective function to the constraints set. Moreover, note that

$$x_0 = 1 - \sum_{i=1}^{n} x_i,$$

thus x_0 can be substituted out from the formulation. Also, let $\hat{\mu} = -\operatorname{diag}(\mu) a + \mu_0 a$, and $\hat{r} = \mu_0 - r$, then we obtain the following equivalent formulation of (3):

$$\begin{array}{llcll} \text{minimize:} & t & & & \\ \text{subject to:} & \hat{\mu}^\top z & \leq & \hat{r} & \\ & a^\top z & \leq & 1 & \\ & 0 \leq a_i z_i & \leq & 1 & i = 1, \ldots, n \\ & z^\top \hat{\Sigma} z & \leq & t & \\ & z & \in & \mathbb{Z}_+^n. & \end{array} \tag{4}$$

Note that formulation (4) may be written as an equivalent MISOCO problem by replacing the convex quadratic constraint $z^\top \hat{\Sigma} z \leq t$ by its conic equivalent. Since matrices Σ and $\hat{\Sigma}$ are symmetric and positive definite, we can replace $\hat{\Sigma}$ by $\hat{\Sigma}^{1/2} \hat{\Sigma}^{1/2}$, where $\hat{\Sigma}^{1/2}$ is the symmetric square root of matrix $\hat{\Sigma}$. If we use the equivalences

$$z^\top \hat{\Sigma} z = \left\| \hat{\Sigma}^{1/2} z \right\|^2, \quad t = \left(\frac{t+1}{2} \right)^2 - \left(\frac{t-1}{2} \right)^2$$

then constraint $z^\top \hat{\Sigma} z \leq t$ can be replaced by its second order conic form of

$$\sqrt{\left\| \hat{\Sigma}^{1/2} z \right\|^2 + \left(\frac{t-1}{2} \right)^2} \leq \frac{t+1}{2}.$$

We use this transformation in the rest of this section whenever we are going to rewrite a problem as an equivalent MISOCO.

2.2 Cardinality and diversification constrained AAP

Investors may want to limit the number of assets to be invested in due to certain reasons. In this case, the investor's aim is to find the best asset allocation that allows to obtain a certain return level while keeping the focus on a fixed number of assets. One's motivation to restrain from investing in too many assets at the same time is to avoid situation where a small amount is invested in an asset. This might bring new costs, such as tracking, that are often neglected in the portfolio optimization settings. Bienstock [13] shows that cardinality constrained quadratic optimization problems are difficult to solve.

We model cardinality constrained AAPs by introducing new binary variables y to the original model (1). We ignore the money market investment for simplicity. After moving the objective function into the constrains as we did earlier, and requesting a minimum return level as a lower bound, we obtain

$$\begin{array}{lrcll}
\text{minimize:} & t & & & \\
\text{subject to:} & \mu^\top x & \geq & r & \\
& \sum_{i=1}^{n} x_i & = & 1 & \\
& x_i & \leq & z_i & i = 1, \ldots, n \\
& \sum_{i=1}^{n} z_i & \leq & k & \\
& x^\top \Sigma x & \leq & t & \\
& x_i & \geq & 0 & i = 1, \ldots, n \\
& z_i & \in & \{0, 1\} & i = 1, \ldots, n,
\end{array} \tag{5}$$

where k is the number of assets to invest in.

A refinement on (5) may be to limit the number of assets bought that are related to each other, which is a common practice in AAPs. This is usually done by classifying assets into sectors or countries and imposing a minimum and maximum number of stocks to invest in for each of these classes. In this way, an investor can prevent buying too many stocks from the same sector, which are known to be related beforehand. This is known as the diversification constraint and it is a general form of the cardinality constraint, which considers various subsets. For a subset $\bar{\mathcal{N}}_j \subseteq \mathcal{N} = \{1, \ldots, n\}$ and cardinality k_j, we represent diversification constraints as follows

$$\sum_{i \in \bar{\mathcal{N}}_j} z_i \leq k_j \qquad \forall j, \tag{6}$$

where $\cup_{\forall j} \bar{\mathcal{N}}_j = \mathcal{N}$. Thus, a cardinality constraint is simply a specialized diversification constraint when $\bar{\mathcal{N}}_j = \mathcal{N}$.

We now present two alternatives to reformulate (5) for taking advantage of the DCCs and DCyCs technology. The first alternative is to replace the linear cardinality constraint by the convex quadratic constraint

$$\sum_{i=1}^{n} z_i^2 \leq k. \tag{7}$$

Observe that, since z is a binary vector, this reformulation does not change the feasible region of the problem, which gives us a variation of the cardinality constrained AAP. Hence, we have the following equivalent reformulation for the AAP with a quadratic cardinality constraint:

$$\begin{array}{lrcll} \text{minimize:} & t & & & \\ \text{subject to:} & \mu^\top x & \geq & r & \\ & \sum_{i=1}^{n} x_i & = & 1 & \\ & x_i & \leq & z_i & i = 1, \ldots, n \\ & \sum_{i=1}^{n} z_i^2 & \leq & k & \\ & x^\top \Sigma x & \leq & t & \\ & x_i & \geq & 0 & i = 1, \ldots, n \\ & z_i & \in & \{0, 1\} & i = 1, \ldots, n. \end{array} \tag{8}$$

Some of the assets in (7) can be left linear in the constraint to obtain the following general form for quadratic cardinality constraint

$$\sum_{i \in \mathcal{U}} z_i^2 + \sum_{i \in \mathcal{N} \setminus \mathcal{U}} z_i \leq k, \tag{9}$$

where $\mathcal{U} \subseteq \mathcal{N}$. The same, equivalent quadratic reformulation can be applied to each of the diversification constraints (6).

The second alternative is obtained when any, or all of the bound constraints $x_i \leq z_i$ are replaced by $x_i^2 \leq z_i$. Hence, we obtain the following reformulation:

$$\begin{array}{lrcll} \text{minimize:} & t & & & \\ \text{subject to:} & \mu^\top x & \geq & r & \\ & \sum_{i=1}^{n} x_i & = & 1 & \\ & x_i^2 & \leq & z_i & i = 1, \ldots, n \\ & \sum_{i \in \mathcal{U}} z_i^2 + \sum_{i \in \mathcal{N} \setminus \mathcal{U}} z_i & \leq & k & \\ & x^\top \Sigma x & \leq & t & \\ & x_i & \geq & 0 & i = 1, \ldots, n \\ & z_i & \in & \{0, 1\} & i = 1, \ldots, n. \end{array} \tag{10}$$

This variation is equivalent to (5) due to the constraint that variables z_i are binary.

In the next section we show how one can generate DCCs for the alternative quadratically constrained reformulations (8) and (10).

3 Methodology

We present our methodology to solve the AAPs introduced in the previous section. We begin with discussing the components of our BCC framework. Then, we describe the cut generation procedure that is used within the BCC framework. Next, we present the cut generation strategies we use for our experiments followed by the branching and searching strategies we test.

3.1 Branch and conic cut framework

There are several options to solve AAPs from both academic [30, 31] and commercial side [3, 21, 23]. Unfortunately, none of the off-the-shelf commercial solvers provide user API to implement nonlinear cuts in their branch and cut algorithms. For these reasons, we developed our own BCC framework.

Our implementation is based on the BCC framework shown in Algorithm 1. The most important aspects of this framework are the conic cut generation, and the cut selection, branching, and searching strategies corresponding to steps 5, 11, 13, 28, respectively. In the current implementation, we use the IPM solver of MOSEK [3] to solve the SOCO subproblems in the tree formed by the branching process. It is well known that IPMs solve SOCO relaxations in polynomial time. Our main focus in this work is the implementation of DCC and DCyCs in the BCC framework, but note that Algorithm 1 allows the use of other types of conic cuts in the literature. As a subset of conic cuts, linear cuts from the Mixed Integer Optimization (MILO) literature can be also used here.

Algorithm 1 Branch and conic cut algorithm for MISOCO

```
Require: A MISOCO problem $M$
Ensure: Optimal solution $x^*$, infeasible flag if $M$ is infeasible, or unboundedness flag if $M$ is unbounded.
 1: $\mathcal{L} = \{M\}$. $f_U = \infty, x^* = \emptyset$.                    ▷ Initialization
 2: if $\mathcal{L} = \emptyset$ then                                       ▷ Termination
 3:     return $x^*$
 4: end if
 5: Choose and remove a problem $N^i$ from $\mathcal{L}$.                   ▷ Selection
 6: Solve continuous relaxation of $N^i$ ($\bar{N}^i$) with an IPM          ▷ Evaluation
 7: if $\bar{N}^i$ is infeasible then
 8:     Go to Step 2.
 9: else
10:     if still has cut and iteration budget then
11:         Search for valid DCCs and DCyCs                                 ▷ Cut generation
12:         if valid cuts exist then
13:             Select which cuts to be added                               ▷ Cut selection
14:             Add cuts to the subproblem $N^i$.
15:             Go to Step 6.
16:         else
17:             Let $f^i = \min\{\bar{N}^i\}$, $x^i = \arg\min\{\bar{N}^i\}$
18:         end if
19:     end if
20: end if
21: if $f^i \geq f_U$ then                                                  ▷ Pruning
22:     Go to Step 2.
23: else if $x^i$ is integer infeasible then
24:     Go to Step 28.
25: else
26:     Set $f_U = f^i, x^* = x$. Delete all problems with $f_L^j \geq f_U$ from set $\mathcal{L}$. Go to Step 2.
27: end if
28: Choose an integer infeasible variable $x_j$ and define $N^{i1}, N^{i2}$ as follows   ▷ Branching

        $$N^{i1} = N^i \cap \{x_j \leq \lfloor x_j \rfloor\}, \quad N^{i2} = N^i \cap \{x_j \geq \lceil x_j \rceil\}$$

    and add them to $\mathcal{L}$. Go to step 2.
```

3.2 Disjunctive conic and cylindrical cut generation

In this subsection, we show how DCCs and DCyCs may be generated for the problems introduced in Section 2. Our goal is to provide the procedure to generate the cuts for a subproblem in Step 11 of Algorithm 1. We follow the cut generation procedures described by Belotti et al. [9, 10, 11], Góez [20].

3.2.1 DCyCs for round-lot constrained AAP

We focus here on how to generate DCyCs for problem (4) using the quadratic constraint

$$z^\top \hat{\Sigma} z \leq t. \tag{11}$$

Let z^*, t^* be the optimal solution to the continuous relaxation of a subproblem. For any asset $i = 1, \dots, n$ where $z_i^* \notin \mathbb{Z}^+$ we can write the following disjunction

$$z_i \leq \lfloor z_i^* \rfloor \quad \vee \quad z_i \geq \lceil z_i^* \rceil. \tag{12}$$

Let us introduce the following notation

$$w = \begin{bmatrix} t \\ z \end{bmatrix}, \; P = \begin{bmatrix} 0 & 0 \\ 0 & \hat{\Sigma} \end{bmatrix} \text{ and } p = \begin{bmatrix} -1/2 \\ 0_{n\times 1} \end{bmatrix}.$$

Hence, we can rewrite (11) in the quadric form as

$$w^\top P w + 2p^\top w \leq 0 \tag{13}$$

Inequality (13) defines a paraboloid since matrix P has exactly one non-positive eigenvalue, which is zero. Due to the shape of the constraint, we can generate disjunctive cylindrical cuts for this constraint as described in [9, 10, 11, 20].

To simplify the DCyC derivation we transform the quadric into a standard representation. Using eigenvalue decomposition the matrix P may be decomposed into $P = V^\top D V$, where D is a diagonal matrix build with the eigenvalues of P, and V is an orthogonal matrix. Note that D is singular because the first element in its diagonal is zero. Let

$$\bar{D} = \begin{bmatrix} 1 & 0 \\ 0 & D_{2:n+1} \end{bmatrix} \quad \text{and} \quad J = \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix}$$

where $\bar{D}$ is simply the matrix obtained after the first diagonal element of D is replaced by 1. By doing that, we obtain a non-singular matrix $\bar{D}$. Let $\overline{V} = \bar{D}^{1/2} V$, then inequality (13) can be rewritten as

$$w^\top \overline{V}^\top J \overline{V} w + 2\left(p^\top \overline{V}^{-1}\right)\left(\overline{V} w\right) \leq 0.$$

Let $u = \overline{V} w$ and $\bar{p} = \left(p^\top \overline{V}^{-1}\right)^\top$, which allows us to rewrite (11) as

$$u^\top J u + 2\bar{p}^\top u \leq 0. \tag{14}$$

Now, consider the parallel hyperplanes

$$\mathcal{A}_i^= = \left\{ w \in \mathbb{R}^{n+1} \mid e_i^\top w = \lfloor z_i^* \rfloor \right\} \text{ and } \mathcal{B}_i^= = \left\{ w \in \mathbb{R}^{n+1} \mid e_i^\top w = \lceil z_i^* \rceil \right\},$$

where e_i is the $(i+1)^{th}$ unit vector, taking into account that the first position belongs to variable t in w. Note that $\mathcal{A}_i^=$ and $\mathcal{B}_i^=$ are the two hyperplanes defining the boundaries of disjunction (12). To keep $\mathcal{A}_i^=$ and $\mathcal{B}_i^=$ consistent with the standardization of (11) we may use the equality $u = \overline{V} w$. Hence, we have that $w = \overline{V}^{-1} u$, and $\mathcal{A}_i^=$ and $\mathcal{B}_i^=$ may be written in terms of u as follows

$$\mathcal{A}_i^= = \left\{ u \in \mathbb{R}^{n+1} \mid e_i^\top \overline{V}^{-1} u = \lfloor z_i^* \rfloor \right\} \text{ and } \mathcal{B}_i^= = \left\{ u \in \mathbb{R}^{n+1} \mid e_i^\top \overline{V}^{-1} u = \lceil z_i^* \rceil \right\},$$

where $e_i \overline{V}^{-1}$ is the $i+1^{th}$ row of $\overline{V}^{-1}$. Let $h_i = \left(e_i^\top \overline{V}^{-1}\right)^\top$, and

$$\alpha_i = \frac{\lfloor z_i^* \rfloor}{\|h_i\|} \text{ and } \beta_i = \frac{\lceil z_i^* \rceil}{\|h_i\|}.$$

Our final step is to identify our DCyC in the uni-parametric family of quadratics $Q(\tau)$ that have the same intersection with $\mathcal{A}_i^= \cup \mathcal{B}_i^=$ and quadric (14) [10], which is characterized by

$$P_i(\tau) = J + \tau \frac{h_i}{\|h_i\|}\left(\frac{h_i}{\|h_i\|}\right)^\top, \qquad p_i(\tau) = \bar{p} - \tau \frac{\alpha+\beta}{2}\frac{h_i}{\|h_i\|}, \qquad \rho_i(\tau) = \tau\alpha\beta.$$

Note that $Pw^c = -p$ is not solvable and the first element of e_i is always zero. Therefore, setting $\tau = -1$ we obtain a cylinder in the family of quadrics. Specifically, the parabolic cylinder

$$u^\top P_i(-1)u + 2p_i(-1)^\top u + \rho_i(-1) \le 0, \tag{15}$$

which cuts off the solution of the continuous relaxation. The cylinder (15) is a DCyC, and it may be written in terms of original variables as follows

$$w^\top \overline{V}^\top P_i(-1)\overline{V}w + 2p_i(-1)^\top \overline{V}w + \rho_i(-1) \le 0.$$

Our round-lot constrained asset allocation subproblem after a single DCyC is added becomes

$$\begin{array}{lrcl}
\text{minimize:} & t & & \\
\text{subject to:} & b^\top z & \le & \hat{r} \\
 & a^\top z & \le & 1 \\
 & 0 \le a_i z_i & \le & 1 \qquad i = 1,\dots,n \\
 & z^\top \hat{\Sigma} z & \le & t \\
 & \begin{bmatrix} t \\ z \end{bmatrix}^\top \overline{V}^\top P_i(-1)\overline{V}\begin{bmatrix} t \\ z \end{bmatrix} + 2p_i(-1)^\top \overline{V}\begin{bmatrix} t \\ z \end{bmatrix} + \rho_i(-1) & \le & 0 \\
 & z & \in & \mathbb{Z}_+^n.
\end{array}$$

Notice that, the costly part of this cut generation is the eigenvalue decomposition of $\hat{\Sigma}$ at the beginning of the operation. However, the same eigenvalue decomposition can be used for all cuts generated on this constraint. Therefore, in our implementation, we apply the eigenvalue decomposition only once and use it everywhere in the BCC tree.

To illustrate DCyCs on round-lot constrained AAPs, consider the constraint

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}^\top \begin{bmatrix} 8.529 & 5.850 \\ 5.850 & 8.805 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \le t,$$

which is in the form of (11)[1]. Solving the continous relaxation of the corresponding round-lot constrained AAP, we get the optimal solution $z_1 = 0.286, z_2 = 0$. Using the parallel disjunction

$$z_1 \le 0, \qquad \vee \qquad z_1 \ge 1$$

and following the procedure described in this section, we obtain the following the DCyC

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}^\top \begin{bmatrix} 3.887 & 5.850 \\ 5.850 & 8.805 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + 4.642z_1 \le t.$$

Adding this DCyC to the problem formulation cuts off the continuous relaxation solution. A 3D plot of the original constraint and the corresponding DCyC of this example is given in Figure 1. Figure 2 shows the projection of the quadratic and DCC onto $t - z_1$ plane.

[1]Approximate numbers up to three digits precision are given here.

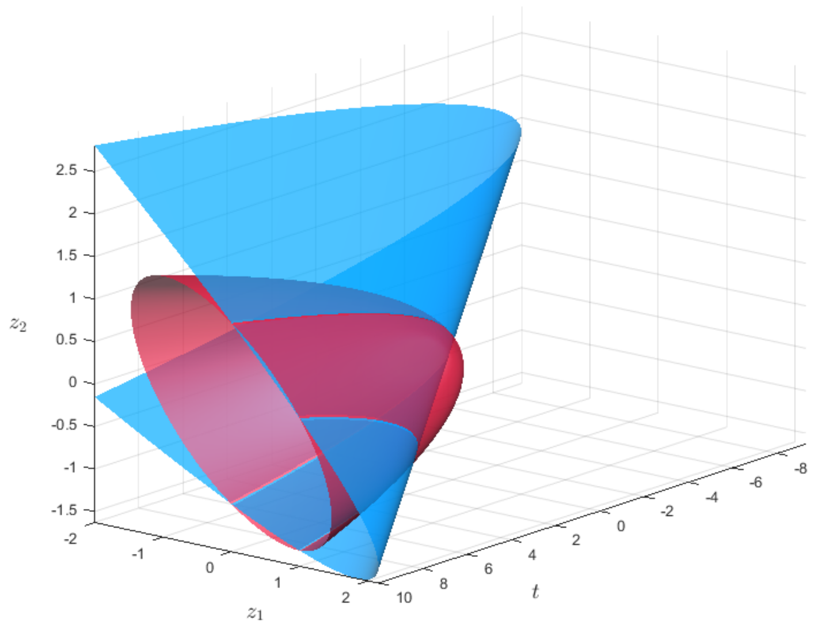

Figure 1: An illustrative DCyC on the quadric of an instance of round-lot constrained AAP.

3.2.2 DCC for quadratic cardinality constrained AAP

For the AAP with quadratic cardinality constraint (8), we can define the disjunctions on the binary variables z. Let z^* be a solution in a subproblem, where $z_i^* \notin \{0, 1\}$, for some $i \in \{1, \dots, n\}$. We use here a parallel disjunction defined as

$$z_i \leq 0 \;\vee\; z_i \geq 1.$$

Let us write the constraint $\sum_{i=1}^{n} z_i^2 \leq k$ as

$$z^T z - k \leq 0.$$

Notice that this time our quadric is a ball around the origin with radius $\sqrt{k}$. Hence, we can generate a DCC using the uni-parametric family of quadrics in [10], which in this case is given by

$$P(\tau) = I + \tau e_i (e_i)^\top, \qquad p(\tau) = -\frac{\tau}{2} e_i, \qquad \rho(\tau) = -k,$$

where e_i is a unit vector with 1 at the i^{th} position, and 0 elsewhere.

To find the DCC for this constraint, we need to solve the following equation:

$$p(\tau)^\top P(\tau) p(\tau) - \rho(\tau) = 0, \tag{16}$$

which is a quadratic equation in terms of τ. There are two cones in this family, and the larger root of (16) gives us the DCyC that tightens the formulation [9, 10, 11, 20]. Using that result we obtain a valid cut for this case when

$$\tau = 2k \left(\sqrt{1 - \frac{1}{k}} - 1 \right).$$

Finally, we can write our DCC as follows:

$$z^\top z + \tau z_j^2 - \tau z_j \leq k.$$

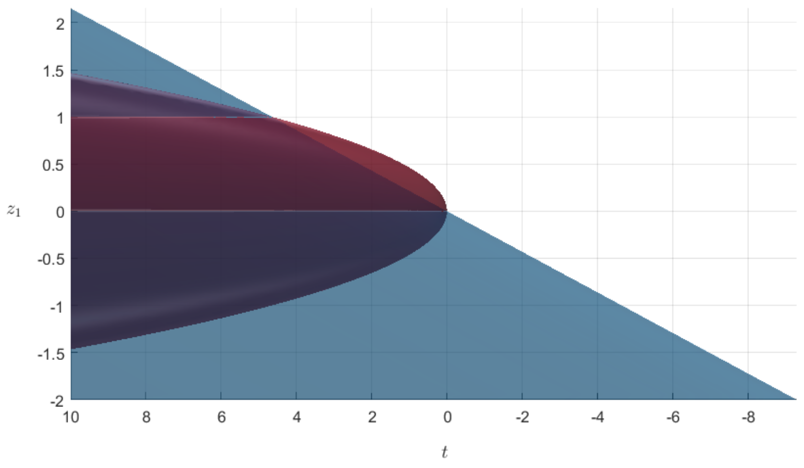

Figure 2: Projection of original quadratic and DCyC onto $t - z_1$ plane. DCyC cuts off some of non-integer points.

An illustration of the intersection of the quadratic cardinality constraint $z_1^2 + z_2^2 + z_3^2 \leq 2$, and the generated DCC which is based on the disjunction $z_1 \leq 0$ or $z_1 \geq 1$ is given in Figure 3. Note that, since the z variables are binary, there are total of n such DCCs that we can add to the cardinality constrained AAP.

Keeping some of the terms linear in the quadratic cardinality constraint (9) allows us to derive DCCs for cardinality constrained AAP of various dimensions. An important observation is that if we have only one asset in the portfolio, i.e, $n = 1$, or if we keep all but one of the assets linear, then the DCC we obtain is simply the original linear cardinality constraint (5).

Diversification constraints are similar to the case of cardinality constraints. This case is equivalent of having multiple cardinality constraints on smaller subsets. We follow the DCC generation procedure described by Belotti et al. [9, 10, 11], Góez [20] for any asset i which belongs to country ℓ, on diversification constraint

$$\sum_{j \in \bar{\mathcal{N}}_\ell} z_j^2 \leq k_\ell$$

by using the disjunction $z_i \leq 0$ or $z_i \geq 1$. The DCC corresponds to this case is

$$\sum_{j \in \bar{\mathcal{N}}_\ell} z_j^2 - 2z_i^2 + 2z_i \leq k_\ell.$$

3.2.3 DCCs for quadratic bound constrained AAP

The use of quadratic bound constraints allows for a simplified process to derive a DCC. Similar to the previous sections, we start with a disjunction on a binary variable z_i. We generate our DCC on the quadratic cardinality constraint, that is

$$\begin{bmatrix} z_j \\ x_j \end{bmatrix}^\top \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_j \\ x_j \end{bmatrix} + 2 \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} \begin{bmatrix} z_j \\ x_j \end{bmatrix} \leq 0.$$

Since the coefficient matrix of the quadratic term in the constraint has an eigenvalue 0, and equality $Px = p$ is not solvable, the shape of the quadric is a paraboloid. Since we have a finite intersection with the parallel

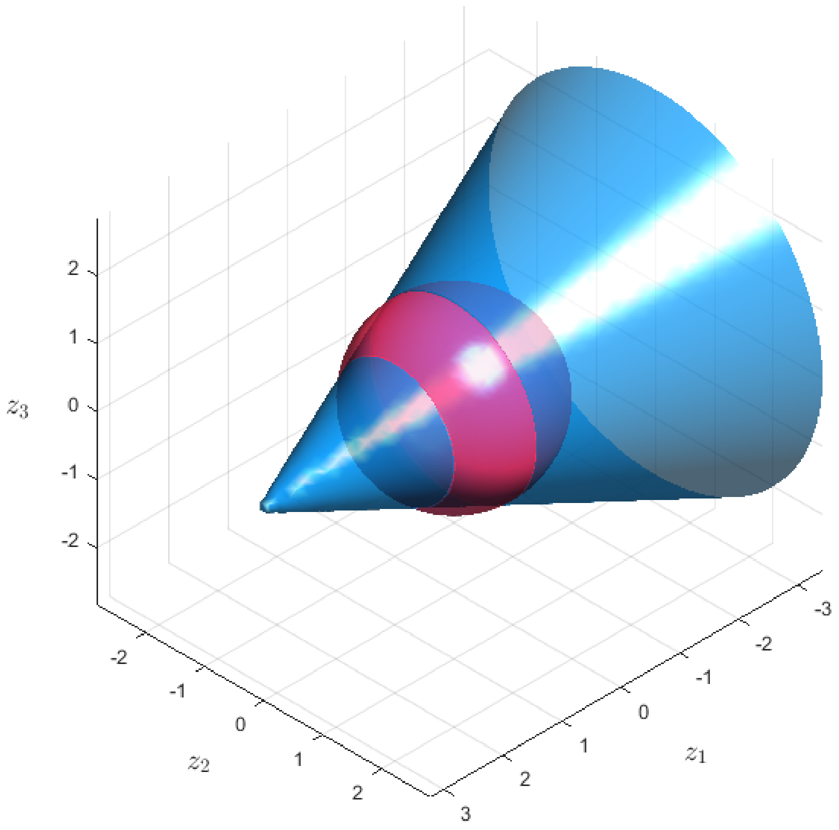

Figure 3: DCC on the quadratic cardinality constraint $z_1^2 + z_2^2 + z_3^2 \leq 2$.

hyperplanes $z_j = 0$ and $z_j = 1$, the DCC in this case is

$$x_j^2 - z_j^2 \leq 0$$

which is illusrated in Figure 4. This constraint is equivalent to the two linear constraints $x_j \leq z_j$ and $x_j \leq -z_j$.

It should be emphasized that, the DCC makes the quadratic bound constraints redundant in this problem formulation. Similar to the previous case, there are total of n such DCCs that can be generated in this way. In the actual implementation, the redundant constraints are removed.

3.3 Cut management strategies

Adding DCCs and DCyCs requires many decisions inside the BCC algorithm. First, we need to decide when and how many cuts are inserted to the subproblems, second, we need to decide about the order of cut generation.

We consider five different cut application strategies in our BCC framework.

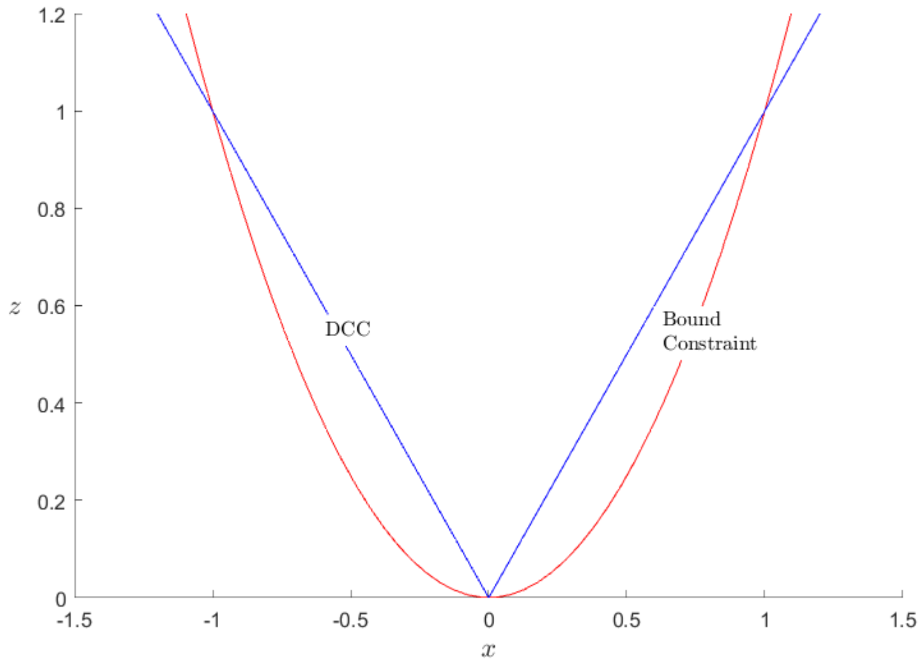

Figure 4: DCC for quadratic bound constrained AAP.

- The first option is to disable all cut generation procedures, hence solving the problem simply with a pure Branch and Bound method. This solution strategy is considered as the base case and all improvements obtained with DCCs and DCyCs are measured compared to this strategy. This method is abbreviated as BB in the numerical results.
- The second cut application strategy is to add a fixed number of cuts to the subproblems until a cut limit is reached on a subproblem. In this method, the order of the cuts that is applied to the subproblem is determined with an ordering rule. Details of this rules are presented in Section 3.4. This method is abbreviated as BCC-F in the numerical results.
- The third method is to keep adding DCCs and DCyCs as long as the objective is making sufficient improvement and then solving the relaxations iteratively. Improvement is defined by a parameter ϵ, by requesting that at least ϵ improvement is obtained. In our experiments we choose $\epsilon = 10^{-3}$. If the procedure fails to obtain an improvement in the objective value, then the framework proceeds with branching. This method is abbreviated as BCC-I in the numerical results.
- Our fourth method is to add all possible DCCs and DCyCs at the root node. We use this rule for cardinality and bound constrained AAPs, where all possible DCCs can be easily obtained at the root node. This method is abbreviated as BCC-R in the numerical results.
- Our final cut application strategy is to produce all the available cuts at a subproblem and then order them in terms of their depth. We use the violation in the generated cut as a measure of its depth. This method is abbreviated as BCC-D in the numerical results.

In the implementation, all the cuts generated at a subproblem are inherited by their children. We limit the total number of DCCs on a subproblem by 10. As mentioned earlier, an important decision in the implementation is to choose which DCCs and DCyCs to be added to the subproblem. We ranked variables to branch and generate a cut using four different rules. These rules are presented in the following subsection.

3.4 Branching and searching

Branching is one of the most important decisions inside the BCC algorithm. Here we list four different rules for selecting the index $\hat{\imath}$ of the variable that will be used for branching. We use these branching decision rules in our implementation.

Our first branching method is *most fractional* (MF) branching, which is borrowed from the MILO literature. In this method one branches using the integer variable whose fractional part is closest to half [1]. This branching rule may be formulated as

$$\hat{\imath} = \underset{i: z_i \notin \mathbb{Z}^+}{\arg\max} \left\{ \min \left\{ \lceil z_i \rceil - z_i, z_i - \lfloor z_i \rfloor \right\} \right\}.$$

Our second branching rule is called *highest cost* (HC) branching, where the variable with the highest return rate (μ) is chosen. We propose this branching rule due to our observation that assets with higher return rates have more impact on the solution when they are fixed. The rule is defined as follows

$$\hat{\imath} = \underset{i: z_i \notin \mathbb{Z}^+}{\arg\max}\, \mu_i.$$

The next two branching rules were proposed by Bonami and Lejeune [15] for portfolio optimization problems with integer variables. The third branching rule uses for branching the asset with the greatest variance Σ_{ii}. This rule is called *idiosyncratic risk* branching (IR), and is defined as follows,

$$\hat{\imath} = \underset{i: z_i \notin \mathbb{Z}^+}{\arg\max}\, \Sigma_{ii}.$$

The fourth rule is called *portfolio risk* (PR) branching. The main idea in this branching rule is to calculate a score for each variable based on the current condition of the problem. Let us simplify the portfolio optimization problem as

$$\begin{array}{rrcl} \text{minimize:} & z^\top \hat{\Sigma} z & & \\ \text{subject to:} & Az & \leq & b \\ & z & \geq & 0, \end{array} \tag{17}$$

where the corresponding Lagrangian of (17) is

$$\mathcal{L}_\lambda(z) = z^\top \hat{\Sigma} z + \lambda^\top (Az - b).$$

Displacement in z changes the Lagrangian as much as

$$\begin{aligned} \mathcal{L}_\lambda(z+\epsilon) - \mathcal{L}_\lambda(z) &= (z+\epsilon)^\top \hat{\Sigma}(z+\epsilon) + \lambda^\top (A(z+\epsilon) - b) - z^\top \hat{\Sigma} z - \lambda^\top (Az - b) \\ &= \epsilon^\top \hat{\Sigma} \epsilon + \left(2z^\top \hat{\Sigma} + \lambda^\top A \right) \epsilon. \end{aligned}$$

At an optimal solution z^* Karush-Kuhn-Tucker conditions satisfy

$$\nabla \mathcal{L}_\lambda(z^*) = 2(z^*)^\top \hat{\Sigma} + \lambda^\top A = 0,$$

hence

$$\mathcal{L}_\lambda(z^* + \epsilon) - \mathcal{L}_\lambda(z^*) = \epsilon^\top \hat{\Sigma} \epsilon.$$

Suppose we have a fractional value for a variable z_i. Branching on z_i creates two branches and for branches with $z_i \leq \lfloor z_i^* \rfloor$ and $z_i \geq \lceil z_i^* \rceil$, the changes in the Lagrangian are

$$\begin{aligned} \delta_j^- &= (z_i^* - \lfloor z_i^* \rfloor) e_i^\top \hat{\Sigma} (z_i^* - \lfloor z_i^* \rfloor) e_i = (z_i^* - \lfloor z_i \rfloor)^2 \Sigma_{ii} \\ \delta_i^+ &= (\lceil z_i^* \rceil - z_i^*) e_i^\top \hat{\Sigma} (\lceil z_i^* \rceil - z_i^*) e_i = (\lceil z_i \rceil - z_i^*)^2 \Sigma_{ii}, \end{aligned}$$

respectively, where e_i is the i^{th} unit vector. The final score associated with variable z_i is calculated as the combination of these two values, such as

$$\delta_i = \alpha \min \left\{ \delta_i^-, \delta_i^+ \right\} + \beta \max \left\{ \delta_i^-, \delta_i^+ \right\},$$

where α and β are weights of the minimum and the maximum of δ_i^- and δ_i^+, respectively. Finally, we choose the variable with the highest δ, i.e.,

$$\hat{i} = \arg \max_{i : z_i \notin \mathbb{Z}^+} \delta_i.$$

We use these same rules for both branching and also choosing a variable to generate DCCs and DCyCs.

The last component we need is the searching rule. In the BCC algorithm searching refers to choosing which subproblem to be processed next. Searching has also strong implications on the performance of a BCC implementation. We use the following searching rules, which are based on well known searching strategies: depth-first with a priority to upper bound constraints, depth-first with a priority to lower bound constraints, and best-first.

4 Numerical results

We analyze in this section the effects of DCCs on the BCC tree size and the solution time. For this purpose, we present experiments with various settings to identify the benefits of the DCCs.

For the experiments we work with two different data sets. First, we use the data set of Black and Litterman [14], which takes into consideration 3 types of investment – equity, bond and currency – for seven countries, which are Germany, France, Japan, United Kingdom, United States, Canada and Australia. The time interval of the data goes from January 1975 to August 1991. The US dollar is used as the base unit, hence there are 20 assets in total. All face values are set to one unit. This dataset is abbreviated as AA in the numerical experiments. Second, we implemented the method described in Hirschberger et al. [22] to generate random portfolio data sets. These sets have been shown to be very close to realistic instances in the original study. We set the expected value of the covariance of assets to $2 \cdot 10^{-3}$, their standard deviation $4 \cdot 10^{-6}$ and the expected value of the variance of assets to 0.1 as used in the original paper. These random portfolio data sets and an open-source script to generate random portfolio data sets are available online [17]. A list of the problem instances are given in Table 1. 10 random datasets we used for experiments are abbreviated from RD0 to RD9 in the numerical experiments.

Instance	# of assets	Capital	Target return
AA	20	400000	5%
RD0	20	400000	8%
RD1	30	100000	7%
RD2	40	300000	3%
RD3	40	300000	4%
RD4	10	800000	3%
RD5	10	700000	5%
RD6	10	600000	2%
RD7	10	1000000	5%
RD8	10	400000	7%
RD9	10	700000	8%

Table 1: Problem instances

The discussion of our results is organized as follows. First, we discuss the effects of DCCs on the BCC tree size and the lower bound obtained in the nodes. Then, we compare the various branching, cutting and searching rules, to choose the default settings for our BCC solver. Later we give a comparison of cut application strategies. We close this section with a comparison of a pure BB, our BCC, and a commercial solver in terms of tree size.

4.1 The effect of DCCs on the objective value and the BCC tree

Our first set of experiments demonstrates the effects of DCCs on the nodes of the BCC tree. As discussed above, DCCs are used to tighten a given MISOCO formulation. Therefore, it is expected that adding DCCs to the nodes of the BCC tree will improve the lower bound in a given node. However, adding cuts to the formulation is an expensive operation. For that reason, it is crucial to look for a systematic way to select which cuts to add to the subproblems. The goal of these experiments is to provide insights on which indicators may be used to identify which DCCs should be added in the BCC tree.

The first indicator we explored is the depth of the DCCs. In the round-lot AAPs, we use the constraint (11) to derive our DCCs, which is obtained by introducing an auxiliary variable and moving the objective function to the constraints. In this case, our results show that the depth of the added DCC has a positive correlation with the improvement on the optimal objective value, as is shown in Figure 5. In particular, that figures shows an almost linear relation between the improvement on the objective value of a subproblem with the depth of the DCC. It is important to notice that this is true provided that the constraint (11) is active at the optimal solution.

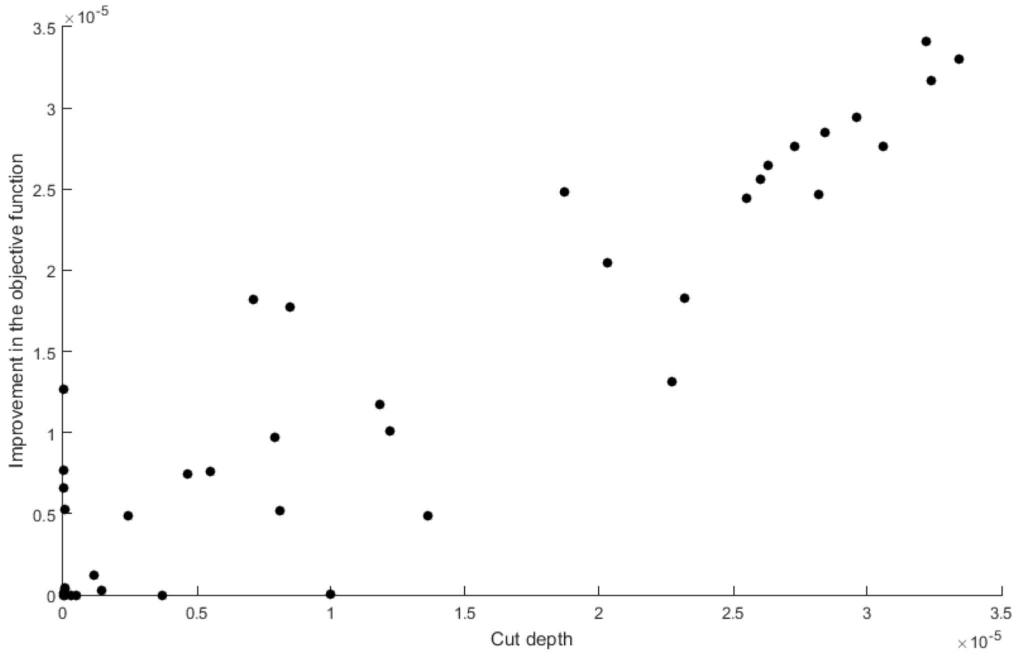

Figure 5: Improvement on the objective value versus node level of generated cuts for round-lot constrained AAPs.

The second indicator of the DCC performance we explored is the level at which the cut is generated in the BCC tree. Recall from our methodology description in Section 3 that we are using IPMs for solving the node relaxation. As a consequence, the addition of a DCC or DCyC may increase the solution time for a subproblem as explained in the following paragraph. This effect can be observed in our experiments. In particular, adding DCCs near root node increases the solution time for all subproblems, hence the BCC solution time. However, DCCs added in the lower levels have less effect on the improvement in the lower bound. Figure 6 shows how improvements in the objective value relates to the node level at which a DCC is added. In that figure we observe that DCCs added in the upper levels are more effective for improving the objective value. Further, adding cuts near root node decreases the BCC tree size significantly, reducing the solution time in general. For this reason, we prefer to add DCCs in the upper levels of the BCC tree compared to lower levels. Consequently, our cut generation procedure starts at the root node for BCC-F method. To lessen the possible increase of the solution time due to new conic constraints, we limited the number of DCCs added to the subproblems.

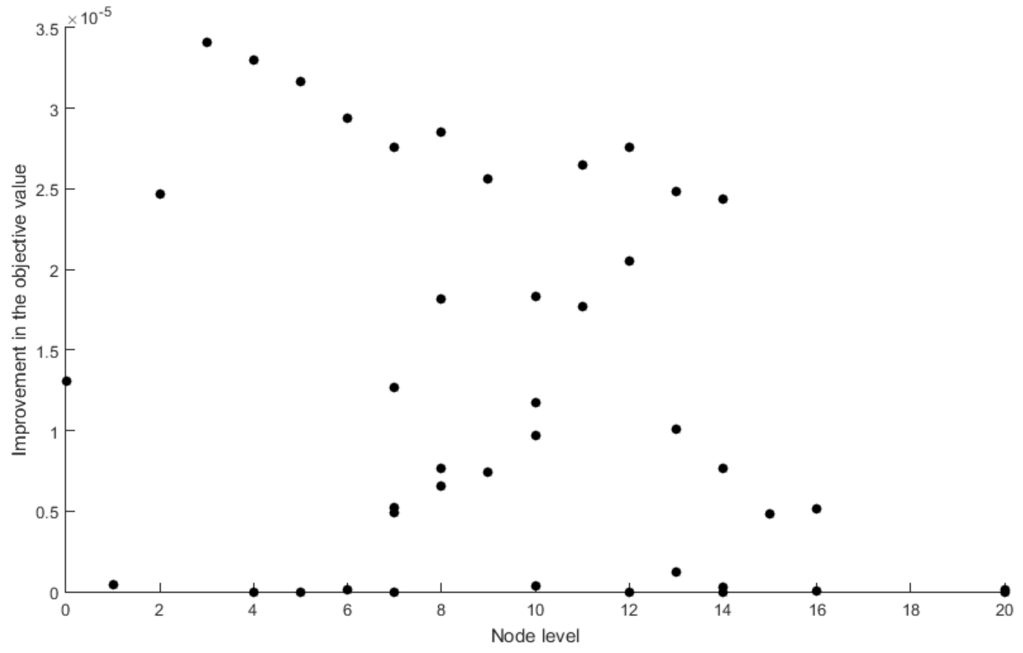

Figure 6: Improvement on the objective value versus node level of generated cuts for roundlot constrained AAPs.

The third indicator we need to consider is the dimension of the DCCs. As discussed earlier, some of the assets in the cardinality constrained AAPs can be left linear for the quadratic cardinality constraint. Notice that, DCCs for cardinality constrained AAPs are full in size, hence the number of quadratic terms in the cardinality constraint gives the dimension of the DCC, as well. We varied the number of assets that are left linear in the quadratic cardinality constraint, which provided us DCyCs with various dimensions. Figure 7 shows that the total solution time increases with the number of quadratic terms in the cardinality constraint. Although we have a single type of DCCs for this setting, one should aim to choose lower dimensional DCCs to minimize its effects on solution time. As mentioned by Pólik and Terlaky [28], iteration complexity of IPMs are independent of the dimension of the cones. However, cost per iteration is affected by the dimension significantly. Although adding a small cone versus adding a big cone has the same complexity in theory, adding a big cone leads longer arithmetic operations, which eventually increases solution time in practice. Since our approach focuses on practice, we were careful in terms of adding a high number of big DCCs in the BCC tree. On the other hand, adding small DCCs are generally a good practice that does not affect IPM solution time significantly.

We may summarize our conclusions from the results presented in this subsection as follows. First, the depth of a DCC is an important indicator for predicting its effect on the lower bound for problems where the objective is used for generating DCCs. Second, DCCs are more effective when they are added closer to the root node of a BCC tree. Hence, to maximize the benefit from the DCCs, one should add them closer to the root node of a BCC tree when possible. Third, the dimension of a DCC is an important indicator of its

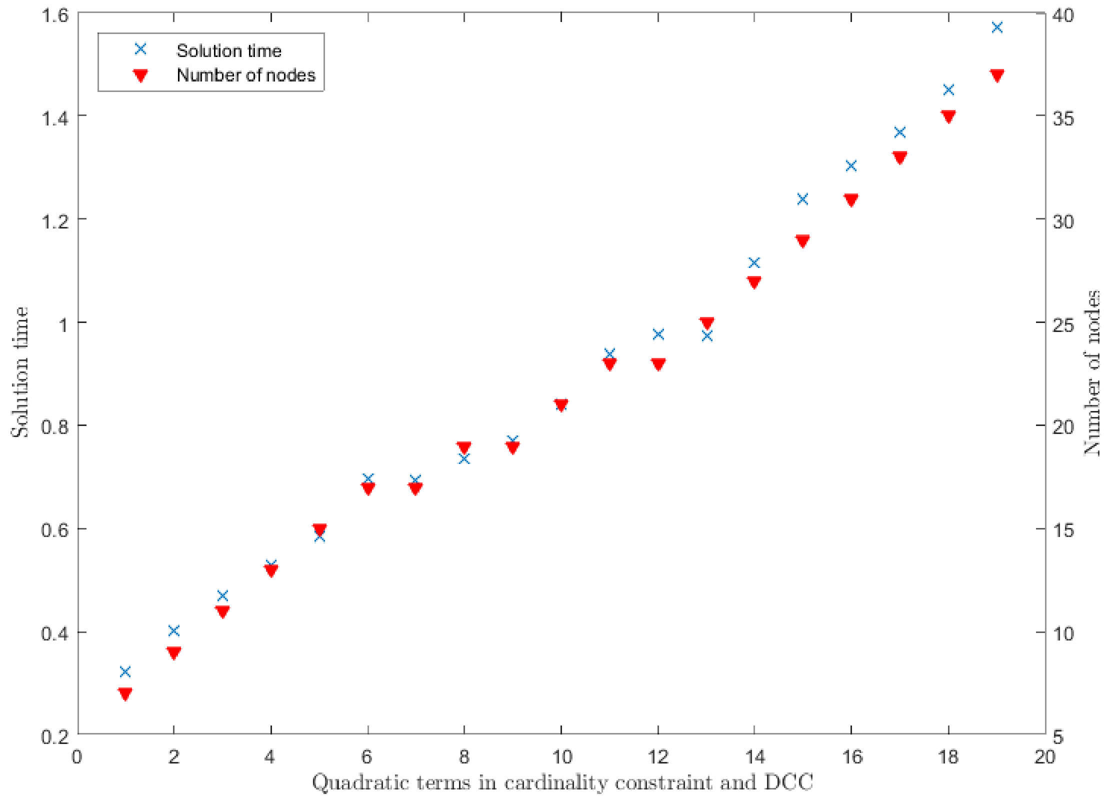

Figure 7: Change in solution time and number of nodes for various dimension of DCCs.

effect on solution time. One should be more selective and reluctant when adding big DCCs inside the BCC tree.

4.2 The effect of branching, cutting and searching rules on the BCC tree

For any given subproblem, one can either branch or generate a cut based on an integer infeasible variable. In the solution of a subproblem, we often get multiple integer infeasible variables. Hence, we need to decide which variable to use for branching or cutting. We use here the variable selection rules presented in Section 3.4. Here, we are interested on exploring their effects on the solution process.

Before comparing the branching and cutting strategies, we start with identfying the most effective searching rule for the AAPs. Table 2 shows the average tree size among all branching and cutting rule alternatives for our four different cut generation strategies. Breadth-first dominates other searcing rules in terms of average BCC tree size both for BB and BCCs with various settings. For that reason, we use breadth-first searching rule as a default option in our BCC implementation.

We are particularly interested in how branch and cut ordering rules affects the BCC when solving AAPs. In Table 3 we compare average BCC tree sizes for all combinations of branching and cut ordering for round-lot AAPs. For PR branching, we chose $\alpha = 1/3$ and $\beta = 2/3$ in our implementation. This selection has an empirical basis, and was based on our practical observations. From Table 3 we observe that ordering branching variables with the MF rule and cuts with the HC rule gives us the minimum tree size in most of

	BB	BCC-F	BCC-I	BCC-D
Best-first	10141.89	9808.78	9999.82	8539.32
Breadth-first	229.64	354.17	354.23	305.82
Depth-first (Left)	4858.77	8402.07	8422.13	7352.95
Depth-first (Right)	6704.66	12545.24	12617.00	9830.61

Table 2: Average BCC tree size with various searching strategies over cut generation strategies for round-lot AAPs.

the cases.

		Branching			
Cutting	Strategy	MF	HC	PR	IR
MF	BCC-F	1175.89	10578.30	6964.66	10708.18
	BCC-I	1176.02	11102.91	7599.55	10710.84
HC	BCC-F	797.91	8210.52	8355.30	8471.70
	BCC-I	798.02	8210.52	8407.82	8476.80
PR	BCC-F	1138.68	14365.11	9111.16	8790.75
	BCC-I	1140.82	14595.95	9111.75	8790.84
IR	BCC-F	881.93	15968.75	8323.77	10598.45
	BCC-I	879.50	16081.18	7675.00	10815.20

Table 3: Comparison of average BCC tree size for branch and cut ordering rules.

Based on these results, we use MF as the default branching rule, HC as the default cut ordering rule, and breadth-first as the default searching rule. We use these settings for the rest of the experiments, unless otherwise noted.

4.3 Comparison of cut application strategies

To identify which cut application strategy performs better, we tested our BCC framework on all data sets for all AAP types. Our aim is to identify the best DCC application strategy for different problem and DCC structures. For this purpose, we compare they performance to a pure BB. We run our experiments by using the default options, MF branching, HC cutting and breadth-first searching, and tested the five cut application strategies presented in Section 3.3.

The results for round-lot constrained AAPs are summarized in Table 6, where we show the number of nodes in the BB and BCC trees and the solution time of various strategies. The results show that in general the BCC methods outperform the BB in terms of the number of nodes. Notice that BCC-I and BCC-F give almost similar results. This is due to fact that BCC-I stops if there is no sufficient improvement in the objective value. For round-lot constrained AAPs, BCC-I and BCC-F give the best performance in general. The number of DCCs generated in these methods are higher than in BCC-D. However, these extra DCCs make a great difference in terms of tree size. In our experiment with the AA instance, the total number of DCCs generated with BCC-I and BCC-F is only one more than with BCC-D, while the number of nodes are almost halved. For a better comparison, performance profiles of solution time and number of nodes are presented in Figure 8 and 9, respectively. Our expectation about DCCs reducing the BCC tree size when compared with a pure BB is achieved by all cut application strategies in many of the cases. On the other hand, an increase in the solution time is often a result of bigger subproblems in the BCC tree nodes. It should be noted that our BCC solver does not benefit from preprocessing and warm-starting between iterative solutions, hence we expect the difference between solution times to be smaller when a full BCC framework is implemented.

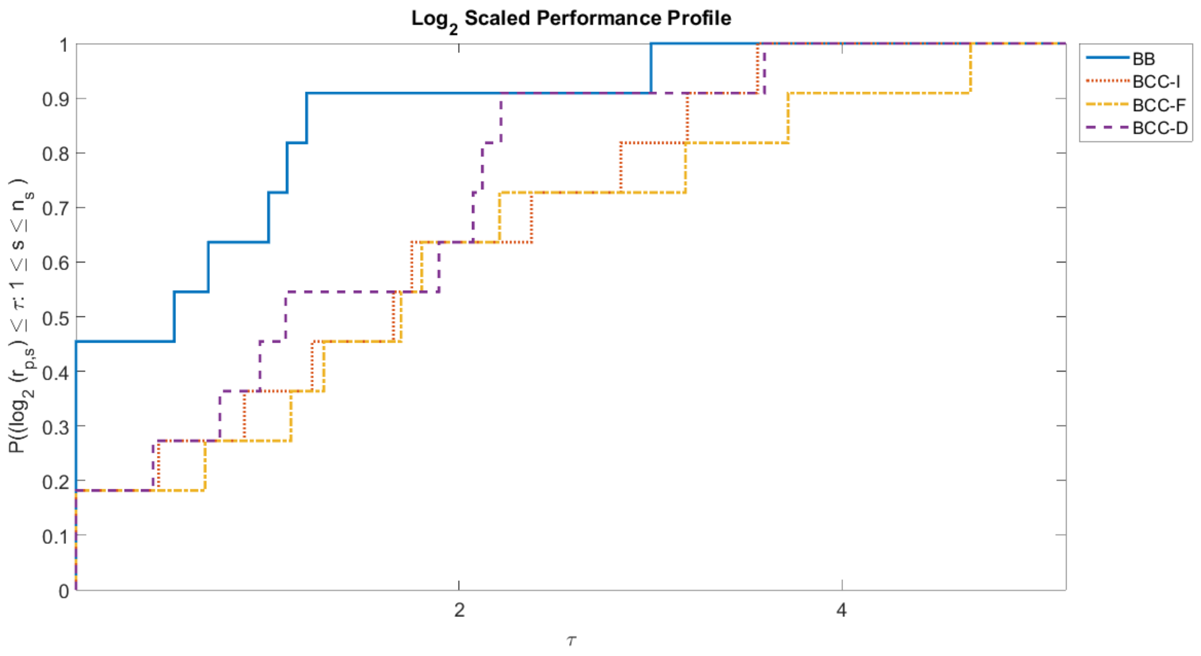

Figure 8: Performance profile of solution time of cut management strategies on round-lot constrained AAP.

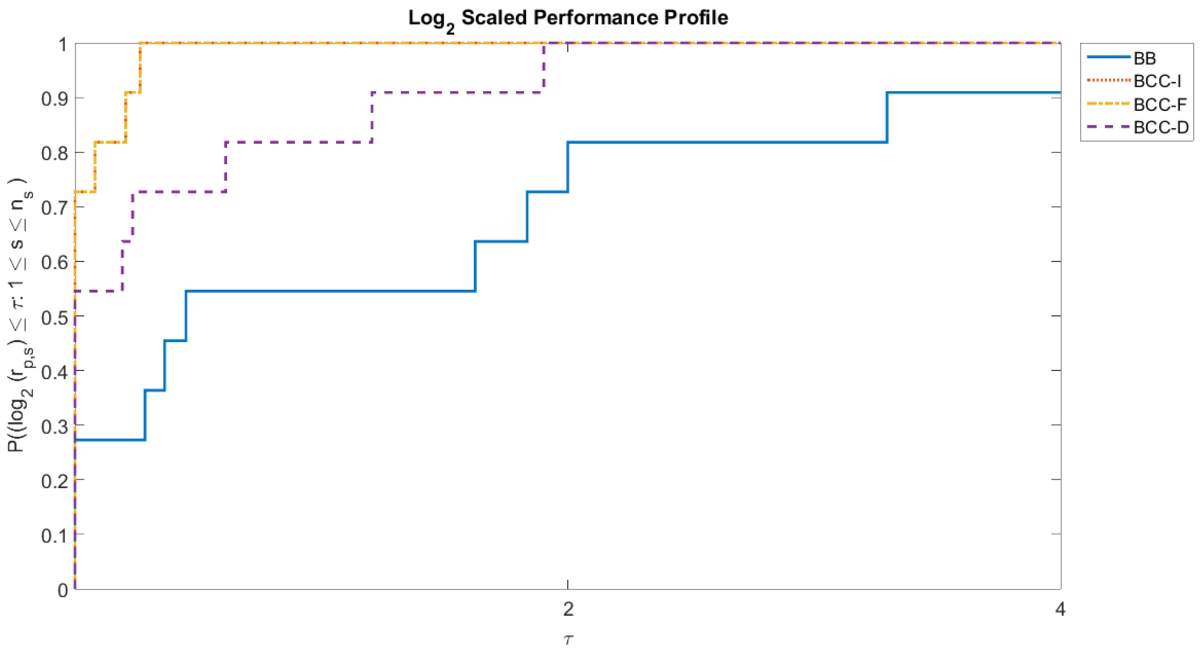

Figure 9: Performance profile of number of nodes of cut management strategies on round-lot constrained AAP.

We choose our cardinality parameter k as 2 in the following experiments. We provide the number of nodes and solution time in Table 7 for each data instance with all cut management strategies. Performance profiles for cardinality constrained AAPs are given in Figure 10 and 11 over five different cardinality parameters, $k = 1, \dots, 5$. Evident from the results, BCC-R, adding possible DCCs at the root node, works better in terms of both solution time and number of nodes in the BCC tree. Since we have limited number of variables, and all of them are binary, adding these cuts at the root node decreases the tree size significantly. Notice that although other methods, BCC-I and BCC-F, generate more DCCs, their number of nodes are higher compared to BCC-R. The success of BCC-R matches with the preliminary results of Góez [20] on constrained layout problems. Even for instances with 40 variables, RD2 and RD3, BCC-R provides significant reduction in terms of number of nodes in the BCC tree and solution time. Note that DCCs generated for cardinality-constrained AAPs are full in sizes, hence the performance of BCC-R is significant. We conclude that DCCs generated for binary variables are better to be added in the root of the BCC tree. Due to nature of the binary variables, we can even drop the original constraint after adding the corresponding DCCs.

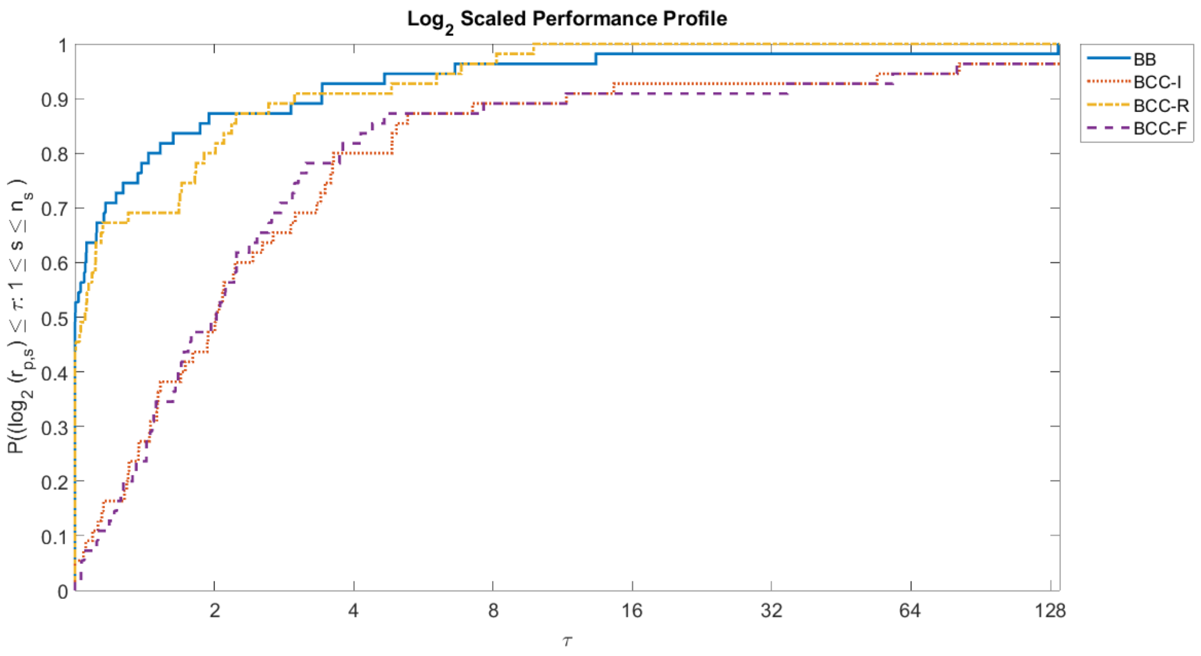

Figure 10: Performance profile of solution time of cut management strategies on cardinality constrained AAP.

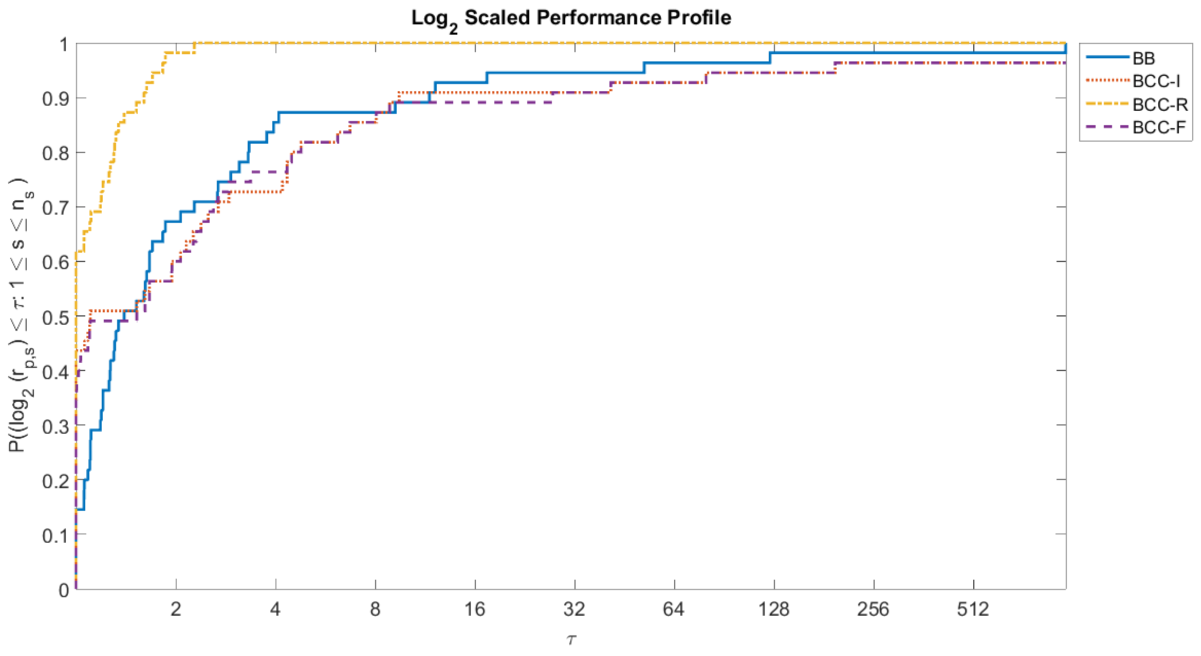

Figure 11: Performance profile of number of nodes of cut management strategies on cardinality constrained AAP.

The results for quadratic bound constrained AAPs are summarized Table 8. These were tested with the default BCC parameters. DCCs generated for quadratic bound constrained AAPs are low in dimension and cheap to calculate. Our results show that BCC-R dominates both BB and other BCC methods in terms of solution time. Performance profile at Figure 12 over five different cardinality parameters verifies this observation. DCCs for quadratic bound constrained AAPs are two dimensional, which lead to smaller savings in terms of solution time compared to round-lot and cardinality constrained AAPs. Adding DCCs in the root node, on the other hand, increases the number of nodes in the BCC tree for most of the instances, as presented in Figure 13. The BCC-I and BCC-F strategies perform better than BB in terms of the number of nodes in the BCC tree, but often result in longer solution time. DCCs generated for such smaller constraints are more powerful when added in the root node.

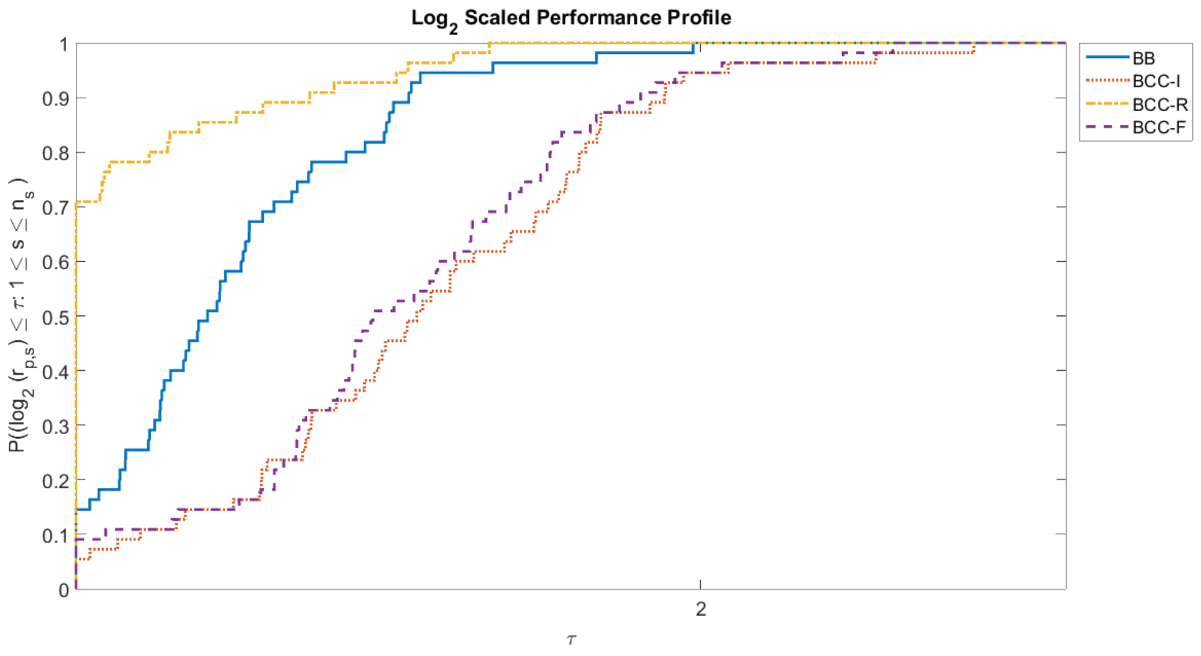

Figure 12: Performance profile of solution time of cut management strategies on quadratic bound constrained AAP.

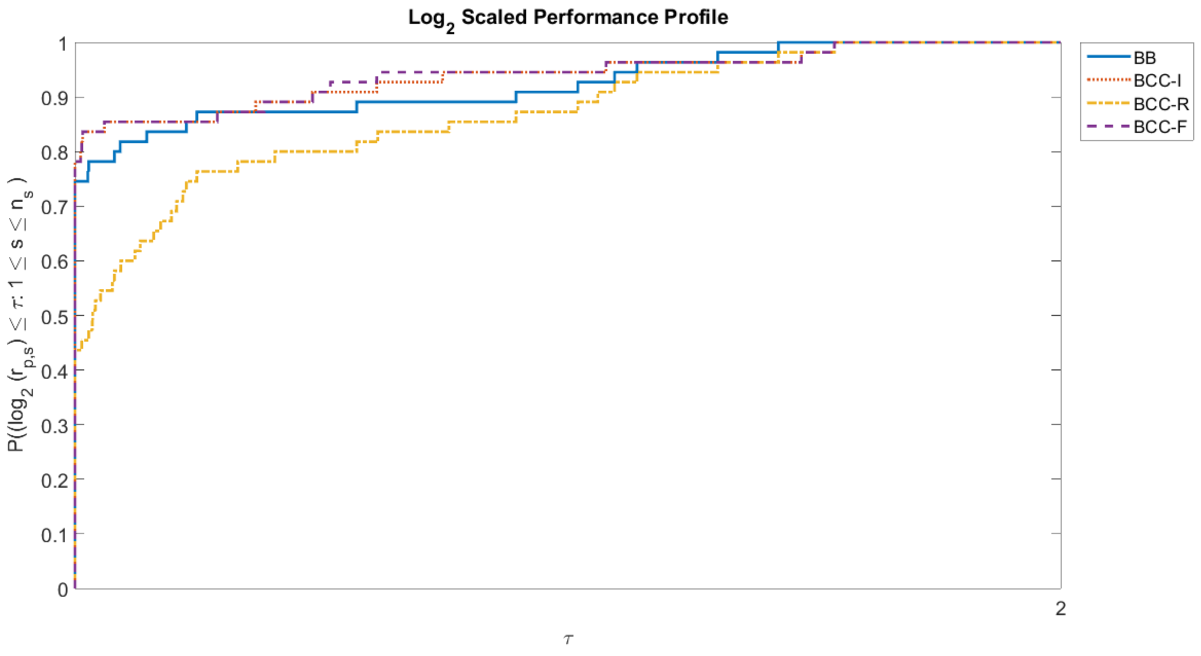

Figure 13: Performance profile of number of nodes of cut management strategies on quadratic bound constrained AAP.

We may summarize the conclusion of these results as follows. First, the BCC-I shows to work best for constraints with general integer variables and BCC-R shows to work best for constraints with binary variables to decrease the number of nodes. In the following subsection, we compare the suggested methodology against

BB and a commercial solver.

4.4 Comparison of solution approaches

To verify our observations from the previous experiments, we tested our suggested BCC framework against pure BB and also a commercial solver, MOSEK. Our aim is to show how much savings we can get in comparison to other methods. We also implemented a new method where DCCs are added as a preprocessing operation for MOSEK, called MOSEK-R. Since MOSEK does not allow adding cuts in the tree, we add these cuts at the root node. We apply this method only for quadratic cardinality and quadratic bound constrained AAPs, due to binary variables in the formulation.

The results are summarized in Table 4, which shows the number of nodes and solution time comparison of BB, BCC-I and MOSEK. From the results we observe that BCC-I outperforms both BB and MOSEK in terms of number of nodes in almost all cases. Performance of the DCCs is very significant for AA, RD2, RD7 and RD9 data sets, where number of nodes are approximately halved. MOSEK dominates BB and BCC-I in terms of solution time. We believe this is a result of MOSEK using its state-of-the-art preprocessing, cut management and warm-starting methods, which we do not utilize in our BCC.

	Number of nodes			Solution Time		
Data	BB	BCC-I	MOSEK	BB	BCC-I	MOSEK
AA	47	15	50	2.254	1.074	0.418
RD0	135	119	73	4.822	3.021	0.196
RD1	83	71	80	3.370	7.353	0.526
RD2	223	127	206	4.336	8.776	0.393
RD3	27	29	143	0.619	1.146	0.444
RD4	6	3	4	0.223	0.205	0.077
RD5	35	36	168	0.668	2.363	0.480
RD6	32	29	51	0.440	1.020	0.217
RD7	17	9	26	0.222	0.472	0.139
RD8	167	183	219	5.460	11.171	0.576
RD9	12	3	4	0.348	0.096	0.076

Table 4: Comparison of number of nodes and solution time of solution approaches for round-lot AAPs.

For quadratic cardinality and bound constrained AAPs, we compare BB, BCC-R, MOSEK and MOSEK-R, where the latter uses MOSEK after all possible cuts are added in the root node. Figure 14 shows the tree size performance profile for cardinality constrained AAPs. These results show that DCCs reduce BCC tree size significantly for cardinality constrained AAPs when added at the root node. Both comparisons among BB with BCC-R, and MOSEK with MOSEK-R reinforce this claim. This example clearly illustrates the power of DCCs in terms of decreasing BCC tree size.

Quadratic bound constrained AAPs show a similar performance to their cardinality constrained counterpart. Figure 15 shows the solution time performance profile. We observe that BCC-R outperforms BB in terms of solution time, although the same improvement is not observed for MOSEK. Remember that BCC-I and BCC-F work better for reducing the number of nodes in the BCC tree for quadratic bound constrained AAPs. Due to limitations of MOSEK's User API, it is not possible to implement these cuts inside the solver, but we would expect a similar effect if implemented.

4.5 Effects of cuts as a preprocessing step

DCCs can be added to MISOCO problems as a preprocessing step. As mentioned earlier, an example of this approach can be seen in Góez [20] on constrained layout problems. These problems include quadratic constraints with a single binary variable. Replacing those quadratic constraints by the generated DCCs

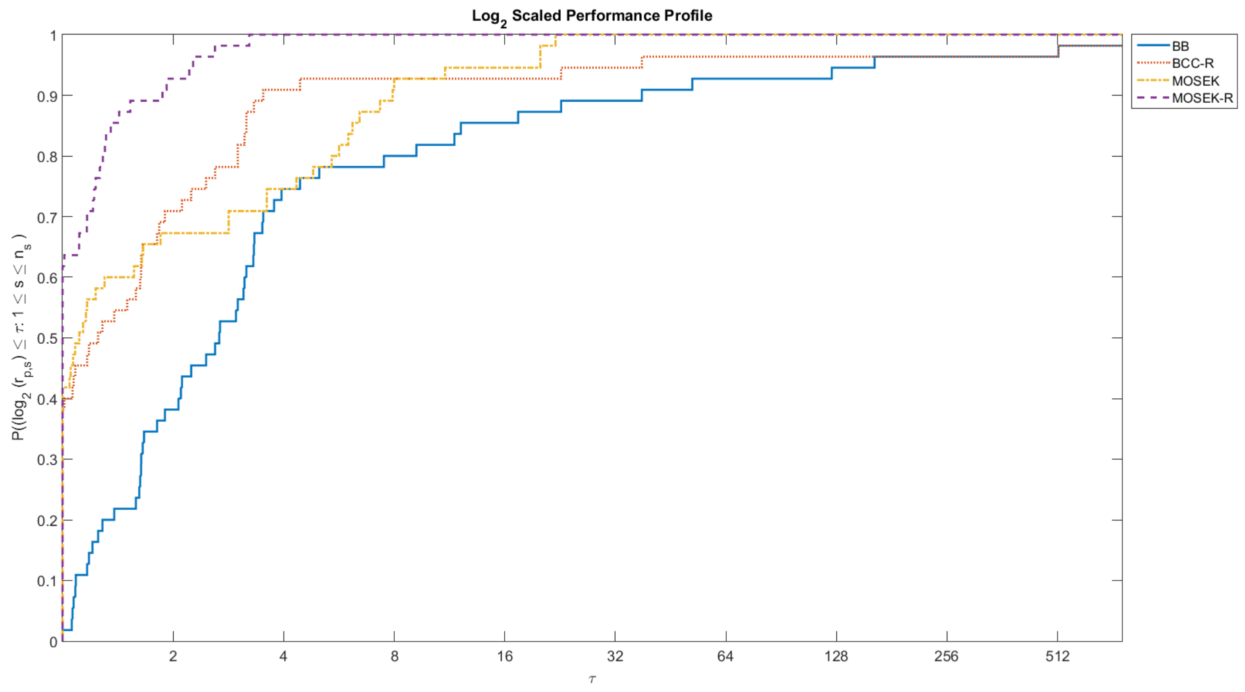

Figure 14: Performance profile of number of nodes on quadratic cardinality constrained AAPs.

based on the binary variable disjunctions is shown to be effective for these problems. DCCs tighten the formulation and reduce solution time significantly.

As an illustration, consider the oversimplified example of quadratic bound constraint

$$x_j^2 \leq z_j.$$

Generating a DCC for this constraint brings back the original linear constraint $x_j \leq z_j$ into formulation, which provides better numerical accuracy. To compare accuracy, we solve the instance RD2 with BB and BCC-R for $k = 1$. We set our integer feasibility tolerance to $\epsilon = 10^{-6}$ and solved the continuous relaxations within BCC with MOSEK in both strategies. Table 5 shows that the sum of the investment into assets with $z_j < 10^{-6}$ sums up to 0.75% of the total investment when solved with BB. This leads to reported optimal objective value to differ by 1.47% of the true optimal value. Adding DCCs as a preprocessing restores numerical accuracy and the solver gives the true optimal solution when solving with BCC-R. In general, we can replace the quadratic constraint that has a single binary variable with the corresponding DCC using the binary disjunction. This preprocessing step at the root node often decrease number of nodes in BCC tree significantly.

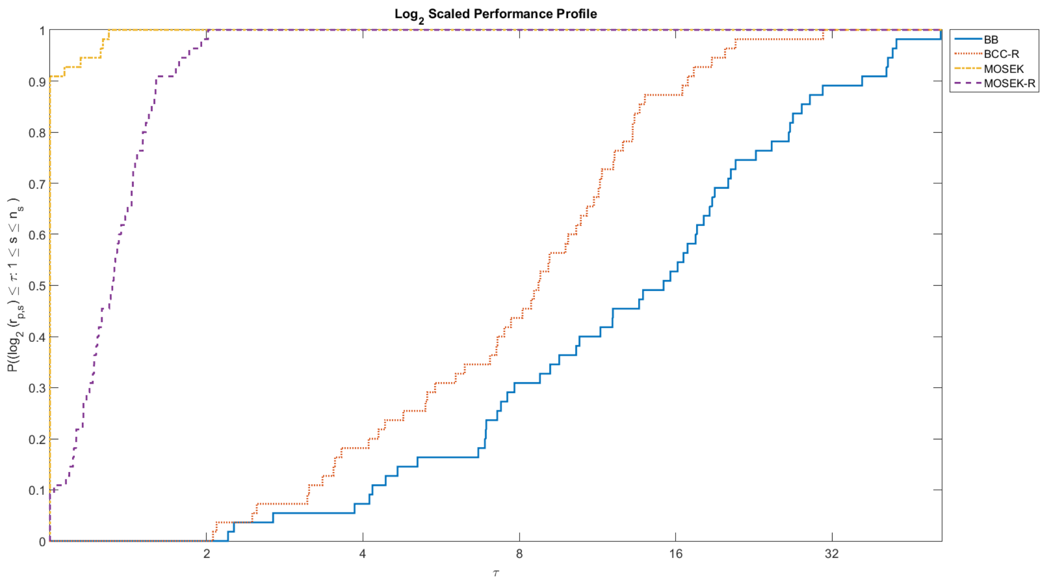

Figure 15: Performance profile of solution time on quadratic bound constrained AAPs.

	BB	BCC-R
Reported optimal objective value	27.84864	28.26441
Primal feasibility at best solution	$2 \cdot 10^{-6}$	$2 \cdot 10^{-9}$
Dual infeasibility at best solution	$5 \cdot 10^{-8}$	$6 \cdot 10^{-11}$

Table 5: Comparison of numerical accuracy of solution without DCCs (BB) versus when DCCs are added in the preprocessing (BCC-R).

5 Conclusions

In this paper, we study the effect of DCCs and DCyCs on solving asset allocation problems. We focused on AAP problems where integer variables appear, more specifically we consider round-lot and cardinality constrained AAPs. We present all cut generation steps for these problems, as well as the details on how we implement DCCs in a complete BCC framework.

Our purpose for studying these problems was to illustrate the positive effect of conic cuts for an important real-world problem. A key contribution of this paper is that it moves the recently developed theory of DCCs from theory to computational practice. We illustrate how the recent theoretical development of DCCs can be used in general purpose MISOCO solvers. We developed a BCC framework around the recently developed DCCs and DCyCs. The proposed software was able to solve numerical instances of AAP in reasonable time and enables us to compare various strategies in terms of managing conic and cylindrical cuts within a BCC framework.

We solved a real-world and several randomly generated data sets. By experimenting with different sized conic and cylindrical cuts, we had a chance to observe their effects. We show that adding DCCs and DCyCs help to reduce the BCC tree size significantly for the majority of the experiments, although solution time may increase in many instances due to increasing size of the problems. We tested several strategies for choosing and adding DCCs and DCyCs within a BCC tree. Our experiments show that BCC-I was the best method for generating DCCs for MISOCO formulations with general integer variables. On the other hand, BCC-R provides significant reduction in tree size for MISOCO formulations with binary variables. We saw significant improvements in the BCC tree size by applying DCCs when they are ordered with the HC rule. It should be emphasized that solution time can be greatly reduced within a full BCC framework with warm-starting, preprocessing, and cut management, which was beyond the scope in this paper.

Many important questions remain for future research, such as cut pooling, cut removal and cut recovery. It should also be noted that we have not used any DCCs or DCyCs based on general disjunctions. The development of warm-start and preprocessing techniques remain for future research, too. As these crucial element are developed, our aim is to develop a comprehensive, efficient, general purpose BCC methodology to solve MISOCO problems by integrating DCCs and DCyCs.

Acknowledgment

The authors acknowledge the support of the Air Force Office of Scientific Research grant #FA9550-15-1-0222 for the development of this research.

References

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1): 42–54, 2005.

[2] M. S. Aktürk, A. Atamtürk, and S. Gürel. Aircraft rescheduling with cruise speed control. *Operations Research*, 62(4):829–845, 2014.

[3] E. D. Andersen and K. D. Andersen. The MOSEK optimization software. *EKA Consulting ApS, Denmark*, 2000.

[4] K. Andersen and A. N. Jensen. Intersection cuts for mixed integer conic quadratic sets. In *Proceedings of the 16th international conference on Integer Programming and Combinatorial Optimization*, pages 37–48. Springer-Verlag, 2013.

[5] A. Atamtürk and V. Narayanan. Conic mixed-integer rounding cuts. *Mathematical Programming*, 122 (1):1–20, 2010.

[6] A. Atamtürk and V. Narayanan. Lifting for conic mixed-integer programming. *Mathematical Programming*, 126(2):351–363, 2011.

[7] A. Atamtürk, L. F. Muller, and D. Pisinger. Separation and extension of cover inequalities for secondorder conic knapsack constraints with generalized upper bounds. Technical report, Technical report, Department of Management Engineering, Technical University of Denmark, Denmark, 2011.

[8] P. Bales, B. Geißler, O. Kolb, J. Lang, A. Martin, and A. Morsi. Comparison of nonlinear and linear optimization of transient gas networks. Technical Report 2552, Darmstadt University of Technology, Department of Mathematics, 2008.

[9] P. Belotti, J. C. Góez, I. Pólik, T. K. Ralphs, and T. Terlaky. Disjunctive conic cuts for mixed integer second order cone optimization. Technical Report 14T-008, Department of ISE, Lehigh University, 2013.

[10] P. Belotti, J. C. Góez, I. Pólik, T. K. Ralphs, and T. Terlaky. On families of quadratic surfaces having fixed intersections with two hyperplanes. *Discrete Applied Mathematics*, 161(16):2778–2793, 2013.

[11] P. Belotti, J. C. Góez, I. Pólik, T. K. Ralphs, and T. Terlaky. A conic representation of the convex hull of disjunctive sets and conic cuts for integer second order cone optimization. In *Numerical Analysis and Optimization*, pages 1–35. Springer, 2015.

[12] D. Bertsimas and R. Shioda. Algorithm for cardinality-constrained quadratic optimization. *Computational Optimization and Applications*, 43(1):1–22, 2009.

[13] D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74(2):121–140, 1996.

[14] F. Black and R. Litterman. Global portfolio optimization. *Financial Analysts Journal*, 48(5):28–43, 1992.

[15] P. Bonami and M. A. Lejeune. An exact solution approach for portfolio optimization problems under stochastic and integer constraints. *Operations Research*, 57(3):650–670, 2009.

[16] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, et al. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[17] S. B. Çay. Random portfolio dataset generator. `http://sertalpbilal.github.io/randomportfolio/`. URL `http://dx.doi.org/10.5281/zenodo.53204`. Accessed: 04/25/2016.

[18] M. T. Çezik and G. Iyengar. Cuts for mixed 0-1 conic programming. *Mathematical Programming*, 104 (1):179–202, 2005.

[19] G. Cornuejols and R. H. Tütüncü. *Optimization Methods in Finance*, volume 5. Cambridge University Press, 2006.

[20] J. C. Góez. *Mixed Integer Second Order Cone Optimization - Disjunctive Conic Cuts: Theory and experiments*. PhD thesis, Department of Industrial and Systems Engineering, Lehigh University, 2013.

[21] Z. Gu, E. Rothberg, and R. Bixby. Gurobi optimizer reference manual, version 6.0. *Gurobi Optimization Inc., Houston, USA*, 2014.

[22] M. Hirschberger, Y. Qi, and R. E. Steuer. Randomly generating portfolio-selection covariance matrices with specified distributional characteristics. *European Journal of Operational Research*, 177(3):1610–1625, 2007.

[23] IBM ILOG CPLEX. V12. 1: Users manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[24] R. A. Jabr, R. Singh, and B. C. Pal. Minimum loss network reconfiguration using mixed-integer convex programming. *Power Systems, IEEE Transactions on*, 27(2):1106–1115, 2012.

[25] R. Mansini and M. G. Speranza. Heuristic algorithms for the portfolio selection problem with minimum transaction lots. *European Journal of Operational Research*, 114(2):219–233, 1999.

[26] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[27] S. Modaresi, M. R. Kılınç, and J. P. Vielma. Split cuts and extended formulations for mixed integer conic quadratic programming. *Operations Research Letters*, 43(1):10–15, 2015.

[28] I. Pólik and T. Terlaky. Interior point methods for nonlinear optimization. In *Nonlinear optimization*, pages 215–276. Springer, 2010.

[29] Ü. Saglam. *Advanced Optimization and Statistical Methods in Portfolio Optimization and Supply Chain Management*. PhD thesis, Drexel University, 2014.

[30] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999.

[31] K.-C. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3 – a MATLAB software package for semidefinite programming. *Optimization methods and software*, 11(1-4):545–581, 1999.

[32] S. Yıldız and G. Cornuéjols. Disjunctive cuts for cross-sections of the second-order cone. *Operations Research Letters*, 43(4):432–437, 2015.

Instance	Strategy	Nodes	SOCO	Time	DCCs	Objective	Node Impr.	Time Impr.
AA	BB	47	47	0.416236	0	2.92E+04		
	BCC-I	15	30	0.284016	6	2.92E+04	68.09%	13.22%
	BCC-F	15	29	0.358755	6	2.92E+04	68.09%	5.75%
	BCC-D	29	30	0.987672	5	2.92E+04	38.30%	-57.14%
RD0	BB	135	135	0.387379	0	1.91E+01		
	BCC-I	119	178	0.883457	4	1.91E+01	11.85%	-49.61%
	BCC-F	119	178	0.833958	4	1.91E+01	11.85%	-44.66%
	BCC-D	119	120	0.747257	3	1.91E+01	11.85%	-35.99%
RD1	BB	83	83	0.821809	0	1.58E+01		
	BCC-I	71	102	2.485459	11	1.58E+01	14.46%	-166.37%
	BCC-F	71	102	2.477008	11	1.58E+01	14.46%	-165.52%
	BCC-D	77	78	1.714736	3	1.58E+01	7.23%	-89.29%
RD2	BB	223	223	1.282353	0	2.49E+00		
	BCC-I	127	207	0.844828	23	2.49E+00	43.05%	34.12%
	BCC-F	127	207	1.246768	23	2.49E+00	43.05%	2.77%
	BCC-D	157	158	1.733978	5	2.49E+00	29.60%	-35.22%
RD3	BB	27	27	0.086293	0	5.14E+00		
	BCC-I	29	47	0.231375	5	5.14E+00	-7.41%	-14.51%
	BCC-F	29	47	0.313216	5	5.14E+00	-7.41%	-22.69%
	BCC-D	41	42	0.186247	2	5.14E+00	-51.85%	-10.00%
RD4	BB	6	6	0.036818	0	3.29E+00		
	BCC-I	3	5	0.041812	2	3.29E+00	50.00%	-0.50%
	BCC-F	3	5	0.030821	2	3.29E+00	50.00%	0.60%
	BCC-D	3	4	0.035436	1	3.29E+00	50.00%	0.14%
RD5	BB	35	35	0.190645	0	1.14E+01		
	BCC-I	36	57	0.292348	9	1.14E+01	-2.86%	-10.17%
	BCC-F	36	57	0.298587	9	1.14E+01	-2.86%	-10.79%
	BCC-D	35	36	0.266011	2	1.14E+01	0.00%	-7.54%
RD6	BB	32	32	0.168585	0	2.10E+00		
	BCC-I	29	43	0.235703	5	2.10E+00	9.38%	-6.71%
	BCC-F	29	43	0.238947	5	2.10E+00	9.38%	-7.04%
	BCC-D	31	32	0.132671	2	2.10E+00	3.12%	3.59%
RD7	BB	17	17	0.075680	0	7.32E+00		
	BCC-I	9	16	0.098087	4	7.32E+00	47.06%	-2.24%
	BCC-F	9	16	0.099855	4	7.32E+00	47.06%	-2.42%
	BCC-D	9	10	0.053402	1	7.32E+00	47.06%	2.23%
RD8	BB	167	167	0.47820	0	1.64E+01		
	BCC-I	183	242	1.642379	16	1.64E+01	-9.58%	-116.42%
	BCC-F	183	242	2.414955	16	1.64E+01	-9.58%	-193.68%
	BCC-D	167	168	0.620489	2	1.64E+01	0.00%	-14.23%
RD9	BB	12	12	0.074879	0	2.09E+01		
	BCC-I	3	5	0.030705	1	2.09E+01	75.00%	4.42%
	BCC-F	3	5	0.026440	1	2.09E+01	75.00%	4.84%
	BCC-D	3	4	0.038644	1	2.09E+01	75.00%	3.62%

Table 6: Performance of cut application strategies on round-lot constrained AAPs.

Instance	Strategy	Nodes	SOCO	Time	DCCs	Objective	Node Impr.	Time Impr.
AA	BB	401	401	2.171734	0	3.16E+04		
	BCC-I	35	53	0.636078	18	3.16E+04	91.27%	70.71%
	BCC-F	111	136	1.573680	24	3.16E+04	72.32%	27.54%
	BCC-R	33	34	1.666077	20	3.16E+04	91.77%	23.28%
RD0	BB	43	43	0.203653	0	1.48E+01		
	BCC-I	21	30	0.167166	9	1.48E+01	51.16%	3.65%
	BCC-F	21	30	0.181225	9	1.48E+01	51.16%	2.24%
	BCC-R	13	14	0.149044	10	1.48E+01	69.77%	5.46%
RD1	BB	71	71	0.598589	0	1.46E+01		
	BCC-I	633	655	8.731575	22	1.46E+01	-791.55%	-813.30%
	BCC-F	1841	1863	20.690974	22	1.46E+01	-2492.96%	-2009.24%
	BCC-R	67	68	5.860951	30	1.46E+01	5.63%	-526.24%
RD2	BB	275	275	0.737180	0	1.42E+01		
	BCC-I	183	258	1.345173	45	1.42E+01	33.45%	-60.80%
	BCC-F	183	258	1.182424	45	1.42E+01	33.45%	-44.52%
	BCC-R	73	74	0.529362	10	1.42E+01	73.45%	20.78%
RD3	BB	261	261	0.874414	0	1.45E+01		
	BCC-I	173	241	1.076240	44	1.45E+01	33.72%	-20.18%
	BCC-F	173	239	1.084397	42	1.45E+01	33.72%	-21.00%
	BCC-R	89	90	0.536467	10	1.45E+01	65.90%	33.79%
RD4	BB	75	75	0.267137	0	1.48E+01		
	BCC-I	55	73	0.475982	13	1.48E+01	26.67%	-20.88%
	BCC-F	55	73	0.408759	13	1.48E+01	26.67%	-14.16%
	BCC-R	19	20	0.137141	10	1.48E+01	74.67%	13.00%
RD5	BB	85	85	0.297274	0	1.45E+01		
	BCC-I	137	186	0.989608	36	1.45E+01	-61.18%	-69.23%
	BCC-F	137	186	0.876514	36	1.45E+01	-61.18%	-57.92%
	BCC-R	51	52	0.500918	10	1.45E+01	40.00%	-20.36%
RD6	BB	243	243	0.803095	0	1.45E+01		
	BCC-I	165	224	1.155648	39	1.45E+01	32.10%	-35.26%
	BCC-F	165	224	1.172764	39	1.45E+01	32.10%	-36.97%
	BCC-R	73	74	0.525372	10	1.45E+01	69.96%	27.77%
RD7	BB	25	25	0.094158	0	1.45E+01		
	BCC-I	25	38	0.208818	7	1.45E+01	0.00%	-11.47%
	BCC-F	25	38	0.198640	7	1.45E+01	0.00%	-10.45%
	BCC-R	23	24	0.209816	10	1.45E+01	8.00%	-11.57%
RD8	BB	51	51	0.173997	0	1.47E+01		
	BCC-I	85	118	0.619426	27	1.47E+01	-66.67%	-44.54%
	BCC-F	85	118	0.647055	27	1.47E+01	-66.67%	-47.31%
	BCC-R	19	20	0.173520	10	1.47E+01	62.75%	0.05%
RD9	BB	105	105	0.298593	0	1.76E+01		
	BCC-I	43	64	0.305285	17	1.76E+01	59.05%	-0.67%
	BCC-F	43	64	0.271417	17	1.76E+01	59.05%	2.72%
	BCC-R	9	10	0.101984	10	1.76E+01	91.43%	19.66%

Table 7: Performance of cut application strategies on quadratic cardinality constrained AAPs.

Instance	Strategy	Nodes	SOCO	Time	DCCs	Objective	Node Impr.	Time Impr.
AA	BB	41	41	0.499925	0	3.13E+04		
	BCC-I	25	36	0.252023	11	3.15E+04	39.02%	24.79%
	BCC-F	25	36	0.349584	11	3.15E+04	39.02%	15.03%
	BCC-R	41	42	0.383218	20	3.16E+04	0.00%	11.67%
RD0	BB	13	13	0.082730	0	1.47E+01		
	BCC-I	13	20	0.096691	7	1.47E+01	0.00%	-1.40%
	BCC-F	13	20	0.098724	7	1.47E+01	0.00%	-1.60%
	BCC-R	13	14	0.074484	10	1.48E+01	0.00%	0.82%
RD1	BB	67	67	0.800091	0	1.45E+01		
	BCC-I	67	79	0.881537	12	1.45E+01	0.00%	-8.14%
	BCC-F	67	79	0.706502	12	1.45E+01	0.00%	9.36%
	BCC-R	67	68	0.566881	30	1.46E+01	0.00%	23.32%
RD2	BB	73	73	0.377500	0	1.41E+01		
	BCC-I	71	86	0.430077	15	1.42E+01	2.74%	-5.26%
	BCC-F	71	86	0.450477	15	1.42E+01	2.74%	-7.30%
	BCC-R	73	74	0.265387	10	1.42E+01	0.00%	11.21%
RD3	BB	89	89	0.419125	0	1.45E+01		
	BCC-I	73	88	0.461414	15	1.45E+01	17.98%	-4.23%
	BCC-F	73	88	0.477966	15	1.45E+01	17.98%	-5.88%
	BCC-R	89	90	0.355050	10	1.45E+01	0.00%	6.41%
RD4	BB	19	19	0.100325	0	1.47E+01		
	BCC-I	19	24	0.129128	5	1.47E+01	0.00%	-2.88%
	BCC-F	19	24	0.137089	5	1.47E+01	0.00%	-3.68%
	BCC-R	19	20	0.091926	10	1.48E+01	0.00%	0.84%
RD5	BB	51	51	0.268953	0	1.45E+01		
	BCC-I	51	66	0.297394	15	1.45E+01	0.00%	-2.84%
	BCC-F	51	66	0.322543	15	1.45E+01	0.00%	-5.36%
	BCC-R	51	52	0.215907	10	1.45E+01	0.00%	5.30%
RD6	BB	73	73	0.322950	0	1.44E+01		
	BCC-I	67	82	0.443591	15	1.45E+01	8.22%	-12.06%
	BCC-F	67	82	0.407107	15	1.45E+01	8.22%	-8.42%
	BCC-R	73	74	0.292802	10	1.45E+01	0.00%	3.01%
RD7	BB	23	23	0.147263	0	1.45E+01		
	BCC-I	23	31	0.177965	8	1.45E+01	0.00%	-3.07%
	BCC-F	23	31	0.145714	8	1.45E+01	0.00%	0.15%
	BCC-R	23	24	0.106840	10	1.45E+01	0.00%	4.04%
RD8	BB	19	19	0.114138	0	1.46E+01		
	BCC-I	19	28	0.140629	9	1.46E+01	0.00%	-2.65%
	BCC-F	19	28	0.127456	9	1.46E+01	0.00%	-1.33%
	BCC-R	19	20	0.099747	10	1.47E+01	0.00%	1.44%
RD9	BB	7	7	0.045051	0	1.76E+01		
	BCC-I	7	10	0.058126	3	1.76E+01	0.00%	-1.31%
	BCC-F	7	10	0.061234	3	1.76E+01	0.00%	-1.62%
	BCC-R	7	8	0.064291	10	1.76E+01	0.00%	-1.92%

Table 8: Performance of cut application strategies on quadratic bound constrained AAPs.