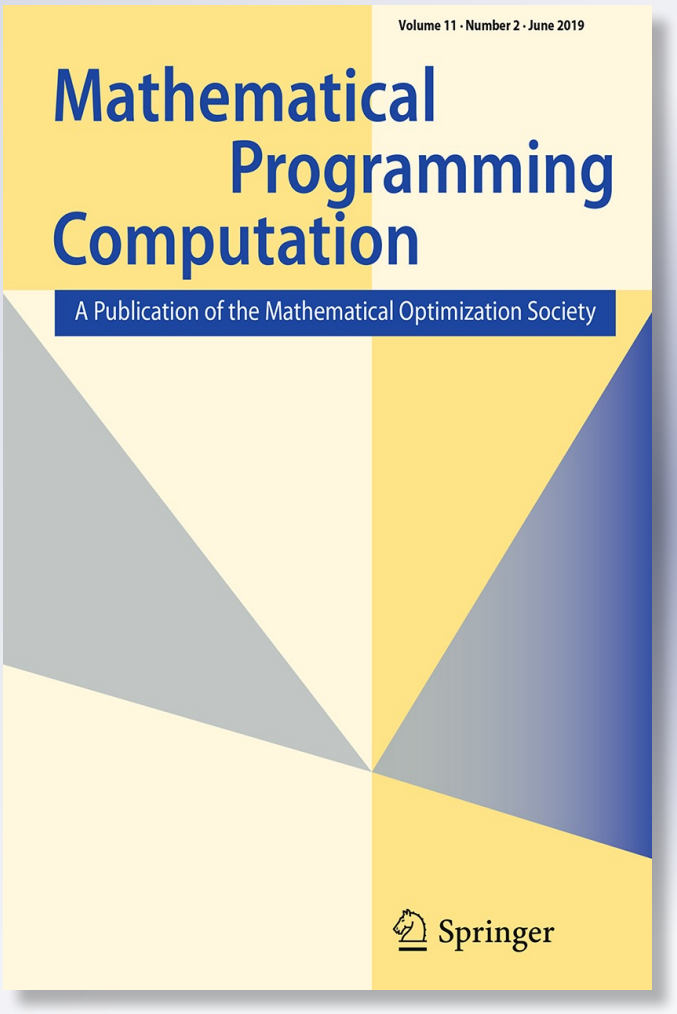
Solving linear programs with complementarity constraints using branch-and-cut

Bin Yu, John E. Mitchell & Jong-Shi Pang

Mathematical Programming Computation
A Publication of the Mathematical Optimization Society

ISSN 1867-2949
Volume 11
Number 2

Math. Prog. Comp. (2019) 11:267-310
DOI 10.1007/s12532-018-0149-2



Springer

Mathematical Programming Computation (2019) 11:267–310
https://doi.org/10.1007/s12532-018-0149-2

FULL LENGTH PAPER

Solving linear programs with complementarity constraints using branch-and-cut

Bin Yu[1] · **John E. Mitchell**[2] · **Jong-Shi Pang**[3]

Received: 5 July 2017 / Accepted: 7 September 2018 / Published online: 27 September 2018

Abstract

A linear program with linear complementarity constraints (LPCC) requires the minimization of a linear objective over a set of linear constraints together with additional linear complementarity constraints. This class has emerged as a modeling paradigm for a broad collection of problems, including bilevel programs, Stackelberg games, inverse quadratic programs, and problems involving equilibrium constraints. The presence of the complementarity constraints results in a nonconvex optimization problem. We develop a branch-and-cut algorithm to find a global optimum for this class of optimization problems, where we branch directly on complementarities. We develop branching rules and feasibility recovery procedures and demonstrate their computational effectiveness in a comparison with CPLEX. The implementation builds on CPLEX through the use of callback routines. The computational results show that our approach is a strong alternative to constructing an integer programming formulation using big-M terms to represent bounds for variables, with testing conducted on general LPCCs as well as on instances generated from bilevel programs with convex quadratic lower level problems.

Keywords Linear programs with complementarity constraints · MPECs · Branch-and-cut

Mathematics Subject Classification 90C33 · 90C57 · 90C26 · 65K10

Yu and Mitchell were supported in part by the Air Force Office of Sponsored Research under Grants FA9550-08-1-0081 and FA9550-11-1-0260 and by the National Science Foundation under Grant Numbers CMMI-1334327 and DMS-1736326. Pang was supported in part by the National Science Foundation under Grant Number CMMI-1333902 and by the Air Force Office of Scientific Research under Grant Number FA9550-11-1-0151.
The software that was reviewed as part of this submission was given in https://doi.org/10.5281/zenodo.1419515.

✉ John E. Mitchell
mitchj@rpi.edu
http://www.rpi.edu/~mitchj

Extended author information available on the last page of the article

1 Introduction

A linear program with linear complementarity constraints (LPCC), which minimizes a linear objective function over a set of linear constraints with additional linear complementarity constraints, is a non-convex, disjunctive optimization problem. In Sect. 1.1, we present the mathematical formulation of the general LPCC we use throughout this paper. In Sect. 1.2, various existing algorithms designed for solving LPCCs are reviewed. Most of these existing methods are only able to obtain a stationary solution and incapable of ascertaining the quality of the solution. This is the major drawback for the existing solvers. In this paper, we mainly focus on finding the globally optimal solution of an LPCC, and we achieve this goal through two steps:

Step 1 Study various valid constraints by exploiting the complementarity constraints directly, and evaluate the benefit of these constraints on the value of the linear relaxation of the LPCC. We have previously discussed valid constraints for the LPCC in [49], and we briefly recap these constraints in Sect. 2.2.

Step 2 Propose a branch-and-cut algorithm to globally solve the LPCC problem, where cuts are derived from various valid constraints studied in Step 1 and branching is imposed on the complementarity constraints. A general LPCC solver has been developed based on this branch-and-cut approach, and it is able to compete with the existing MIP-based solvers like CPLEX.

The branch-and-cut algorithm is introduced in Sect. 1.3, where we also outline the rest of the paper.

1.1 Statement of the problem

We consider a general formulation of the LPCC in the form suggested by Pang and Fukushima [52]. Given vectors and matrices: $c \in \mathbb{R}^n$, $d \in \mathbb{R}^m$, $b \in \mathbb{R}^k$, $q \in \mathbb{R}^m$, $A \in \mathbb{R}^{k\times n}$, $B \in \mathbb{R}^{k\times m}$, $N \in \mathbb{R}^{m\times n}$ and $M \in \mathbb{R}^{m\times m}$, the LPCC is to find $(x, y, w) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m$ in order to solve to global optimality

$$\begin{array}{lll} \underset{(x,y,w)}{\text{minimize}} & c^T x + d^T y & \\ \text{subject to} & Ax + By \geq b & \\ & x \geq 0 & \\ \text{and} & 0 \leq y \perp w := q + Nx + My \geq 0 & \end{array} \tag{1}$$

where $a \perp b$ denotes perpendicularity between vectors a and b, i.e., $a^T b = 0$. Without the orthogonality condition $y \perp w$, the LPCC is a linear program (LP). The global resolution of the LPCC means the generation of a certificate showing that the problem is in one of its 3 possible states: (a) it is infeasible, (b) it is feasible but unbounded below, or (c) it attains a finite optimal solution. Note that problem (1) is equivalent to 2^m linear programs obtained by making each possible assignment for the complementary variables: either $y_i = 0$ or $w_i = 0$ for each $i = 1, \ldots, m$; hence, it is not possible for an LPCC to have a finite optimal value that is not attained.

If the feasible regions for y and w are bounded then there exist diagonal matrices Θ^y and Θ^w with diagonal entries θ_i^y and θ_i^w and problem (1) can be formulated as a mixed integer program:

$$\begin{array}{lll} \underset{(x,y,z)}{\text{minimize}} & c^T x + d^T y & \\ \text{subject to} & Ax + By \geq b & \\ & x \geq 0 & \\ & 0 \leq y \leq \Theta^y z & \\ & 0 \leq q + Nx + My \leq \Theta^w(\mathbf{1} - z) & \\ \text{and} & z \in \{0,1\}^m & \end{array} \tag{2}$$

The obvious drawback of this formulation is that in order to find θ_i^y and θ_i^w we need to compute valid upper bounds of y_i and w_i, not to mention such upper bounds may not exist if the feasible regions for y and/or w are unbounded. To avoid this drawback, in this paper, we present a branch-and-cut algorithm which branches on the complementarity constraint directly. Previous work on branching on complementarity constraints includes [11,17,30].

Problem (1) generalizes the standard linear complementarity problem (LCP) [14]: $0 \leq y \perp q + My \geq 0$, so the LPCC is NP-Hard. Moreover, affine variational constraints also lead to the problem (1) [46]. Applications of the LPCC are surveyed in [33]. Among these applications, complementarity constraints play three principal roles during the modelling process:

1. Modelling KKT optimality conditions that must be satisfied by some of the variables. Such applications include hierarchical optimization such as Stackelberg games [55], inverse convex quadratic programs, indefinite quadratic programs [27,31], and cross-validated support vector regression [42,43].
2. Modelling equilibrium constraints. See for example the texts [19,46], the survey article [51], or a recent paper on market equilibrium in electric power markets [24].
3. Modelling certain logical conditions that are required by some practical optimization problems. Such applications include non-convex piecewise linear optimization, quantile minimization [53], and ℓ_0-minimization [13,21].

1.2 Previous work on solving LPCCs

Research on algorithms for solving an LPCC can be divided into two main areas: one concerns the development of globally convergent algorithms with a guarantee of finding a suitable stationary point; the other concerns the development of exact algorithms for global resolution of an LPCC. See the survey [39] for a more detailed review.

It is noted that the methods which are able to solve a general LCP can also be extended to solve an LPCC by using the so called sequential LCP Method. Such a procedure can be found in detail in [40]. A complementary pivoting algorithm for an LPCC is an extension of a pivoting algorithm for LCP which handles linear complementarity constraints just as the classic simplex algorithm for linear programs.

Such algorithms usually perform in this way: start from a feasible solution, maintain feasibility for all iterations and try to improve the objective function. Under certain constraint qualifications, these methods guarantee convergence to a certain stationary solution. The references [20,26,36] study and implement this type of method to solve the general LPCC. Another way to get a stationary point is through a so called regularization framework [54]: construct a sequence of relaxed problems controlled by some parameter, then obtain a sequence of solutions which converge to a stationary point when the parameter goes to the limit. Each regularized relaxed problem is solved by an NLP based algorithm such as an interior point method. One method of regularization is to introduce a positive parameter ϕ and relax the complementarity constraints in problem (1) using either $\{y, w \geq 0, y^T w \leq \phi\}$ or some other approach [13,21]. An alternative is to put a penalty for violation of the complementarity constraints into the objective, and gradually update the penalty to infinity [44]. A homotopy method has also been proposed [57]. The obvious drawback of these methods is that they are incapable of ascertaining the quality of the computed solution.

The methods for global resolution of an LPCC are mainly based on an enumerative scheme. Several branch-and-bound methods have been proposed for solving an LPCC derived from a bilevel linear program. Bard and Moore [10] proposed a pure branch-and-bound method for solving bilevel linear programs. Hansen et al. [29] enhance this branch-and-bound scheme by exploiting the necessary optimality condition of the inner problem. As opposed to a branch-and-bound method, the references [34,38] study alternative ways to solve an LPCC by using a cutting plane method. Audet et al. [6] proposed a branch-and-cut algorithm for solving bilevel linear programs. An RLT method for finding a feasible solution to a problem with both binary and complementarity constraints is proposed in [24]. It follows from the results of [8] that an LPCC can be lifted to an equivalent convex optimization problem so it can in principle be solved globally using a convex optimization algorithm; the drawback to this approach is that the convexity is over the cone of completely positive matrices which is hard to work with computationally.

It is noted that most of the existing methods for global resolution of the LPCC presume the LPCC has a finite optimal value, and this limitation was not resolved until the paper [32]. In that paper, the authors proposed a minimax integer programming formulation of the LPCC, and solve this system using a Benders decomposition method. The method was extended to quadratic programs with complementarity constraints in [7]. A branching scheme for determining boundedness of the optimal value of a linear program with a bilinear objective function was proposed in [5].

The success of the Benders decomposition method [7,32] heavily depends on a so called sparsification process. If the sparsification process is not successful, in the worst case it will be necessary to check every piece of the LPCC. In this paper, we alternatively use a specialized branch-and-cut scheme which is a more systematic enumerative process to get the global resolution of the LPCC, and our algorithm is also able to characterize infeasible and unbounded LPCC problems as well as solve problems with finite optimal value. Moreover we also discuss various valid constraints for the LPCC by exploiting the complementarity structure; this topic has not been fully exploited in the literature for studying the LPCC.

The complementarity structure of an LPCC can be generalized to SOS1 constraints, a type of special ordered set constraint requiring that at most one of a set of variables is nonzero. Recent work on branch-and-cut approaches to problems with SOS1 constraints include [18,22]. De Farias et al. [18] considered problems where all the coefficients are nonnegative and their emphasis is on possible families of cutting planes using a sequential lifting procedure. Fischer and Pfetsch [22] emphasize cuts and branching techniques for problems with overlapping SOS1 constraints, that is, sets of complementarity constraints that have variables in common; this structure can be represented with conflict graphs and can be exploited in the derivation of valid cutting planes and in the construction of sophisticated branching rules building on the ideas of Beale and Tomlin [11]. In our formulation, each variable appears in at most one complementarity constraint, so the nice techniques of Fischer and Pfetsch would not be helpful.

1.3 LPCC using branch-and-cut

In this paper, we propose a branch-and-cut algorithm for solving the general LPCC problem (1). In Sect. 2, we describe the preprocessing phase of our algorithm: in Sect. 2.1, a heuristic feasibility recovery procedure is developed to recover a feasible solution of the LPCC which provides a valid upper bound of the LPCC; and in Sect. 2.2, the strategy of generating and selecting from various types of cutting planes we studied in [49] is discussed, which could sharpen the LP relaxation and improve the initial lower bound of LPCC. In Sect. 3, we present the second phase of our algorithm: branch-and-bound. Various node selection strategies and branching complementarity selection strategies are discussed in Sects. 3.1 and 3.2. Our proposed algorithm is able to characterize infeasible and unbounded LPCC problems as well as solve problems with finite optimal value. The algorithm is summarized in Sect. 4. In Sect. 5, we show the computational results of our branch-and-cut algorithm on solving randomly generated LPCC instances.

In the MIP formulation (2), the binary vector z is only used to model the complementary relationship of the LPCC, and except for the complementarity constraints it does not interact with x and y at all. This observation motivates us to enforce the complementarities through a specialized branching scheme, i.e., branch on complementarities directly without introducing the binary vector z. This kind of specialized branching approach has been studied to solve several problems such as generalized assignment problems [16], nonconvex quadratic programs [56], nonconvex piecewise linear optimization problems [41], and problems with overlapping SOS1 constraints [18,22]. The obvious advantage of using a specialized branching approach for solving the LPCC is that we no longer need θ in the formulation, and therefore this approach is also applicable for the case when y or w is unbounded. In fact, even if we know such a θ exists, the cost of computing a valid θ could be very expensive especially when m is very large. Moreover, introducing the binary vector z will lead to an increase in both the number of variables and the number of constraints, and these Big-M type constraints are usually not tight which will lead to a number of violated complementarities in the solution of the relaxation.

2 Preprocessing phase

When the initial LP relaxation is bounded below, the preprocessing phase will be invoked, consisting of a feasibility recovery process and a cutting plane selection and management process. The feasibility recovery process may provide a valid upper bound for the LPCC, while the cutting plane selection and management process may provide a better lower bound for the LPCC. Both processes may provide a good starting point for the second phase of our algorithm: branch-and-bound.

2.1 LPCC feasibility recovery

Finding a good feasible solution to an LPCC is an essential component of our branch-and-cut algorithm for globally resolving the LPCC. A good upper bound can help prune nodes quickly, and avoid unnecessary branching. Notice that here we assume the initial LP relaxation is bounded when we apply our feasibility recovery procedures. Our feasibility recovery procedures have some similarities to feasibility pumps for MIP and MINLP [23].

For ease of discussion, we first introduce some notation and definitions.

Definition 1 Given any binary vector z with dimension m, we define the linear program LPCC(z) as follows:

$$\begin{aligned} \underset{(x,y)}{\text{minimize}}\ & c^T x + d^T y \\ \text{subject to}\ & Ax + By \geq b \\ & x \geq 0 \\ & 0 \leq y \\ & 0 \leq q + Nx + My \\ & 0 \geq y_i \qquad\qquad \text{if } z_i = 0 \\ & 0 \geq (q + Nx + My)_i \quad \text{if } z_i = 1. \end{aligned} \tag{3}$$

LPCC(z) is a so-called piece of the LPCC corresponding to the binary vector z.

Definition 2 The feasibility gap of the piece of an LPCC corresponding to the binary vector z, denoted by FG-LPCC(z), is the optimal value of the following linear program:

$$\begin{aligned} \underset{(x,y,w)}{\text{minimize}}\ & (\mathbf{1} - z)^T y + z^T w \\ \text{subject to}\ & Ax + By \geq b \\ & x \geq 0 \\ & 0 \leq y \\ & 0 \leq w := q + Nx + My \end{aligned} \tag{4}$$

where $\mathbf{1}$ is the vector with all components equal to 1.

Based on the above two definitions, it is obvious that the following proposition is true:

Proposition 1 *$LPCC(z)$ is feasible if and only if FG-LPCC$(z) = 0$*

Definition 3 Binary vectors z and z' are *adjacent* if there is exactly one component that is different between z and z'.

Definition 4 If binary vectors z and z' are adjacent and FG-LPCC(z) < FG-LPCC(z'), then $\Delta z = z - z'$ is a *feasibility gap descent direction* for z'.

Just like mixed integer programs, it is often a good idea to recover a feasible solution based on the LP relaxation solution. The most intuitive recovery process is to round the LP relaxation solution into a solution that satisfies all the complementarity constraints. We use this rounding procedure to initialize a new local search feasibility recovery process, detailed in Procedure 1. Notice that we define search $breadth$ as the number of candidates that we are going to select from binary vectors which are adjacent to the initial z^*, and search $depth$ as the maximum number of iterations that we are going to perform for each candidate. We can set search $breadth$ and search $depth$ to control the local search process. The proposed local search procedure can be used to find a feasible solution, although the quality of the recovered feasible solution is not guaranteed. We use optimality based bound tightening [28,47,58] to resolve this issue, refining the local search feasibility recovery procedure through the addition of the constraints $lb_{search} \leq c^T x + d^T y \leq ub_{search}$ to (4) when computing the feasibility gap. Procedure 2 describes this refined feasibility recovery procedure. We will demonstrate the computational results of our proposed local search feasibility recovery process in Sect. 5. See Fischer and Pfetsch [22] for primal heuristics that can be used when a variable appears in more than one complementarity constraint.

2.2 Cutting plane generation and selection

The second key step in our preprocessing phase is the generation and selection of cutting planes. We have discussed various valid linear constraints and second order cone constraints that can be used to tighten the initial relaxation of LPCC in [49], and have shown the computational results of these valid constraints individually. As important as finding these cutting planes is the selection of the cuts that actually should be added to the initial LP relaxation. In this section, we will describe our detailed procedure to generate and select our cutting planes. Note that we will only add cutting planes at the root node, and perform the generation of each type of cut in rounds and in the following order:

- Disjunctive cuts and Simple cuts
- Bound cuts
- Linear cuts derived from second order cone constraints

We use the computational results with these cutting planes in [49] to guide the cut generation process. The details of generation and selection rule are described as follows.

input : the LP relaxation solution of the original LPCC: x^*, y^*, w^*, search depth parameter $depth$, search breadth parameter $breadth$
output: recovered feasible LPCC solution or failed to recover the solution

Initialization: Set binary vector $z^* = 0$;
for $i \leftarrow 1$ **to** m **do**
  if $y_i^* < w_i^*$ **then** z_i^*=0;
  else z_i^*=1;
end
Solve (4) to get *FG-LPCC(z^*)*;
if *FG-LPCC(z^*)==0* **then**
  solve *LPCC(z^*)*, and **return** the optimal solution to *LPCC(z^*)*;
end
else
  let $A(z^*)$ denote the set of binary vectors that are adjacent to z^*;
  foreach $z \in A(z^*)$ **do**
    solve (4) to get *FG-LPCC(z)*;
    insert z into a sorted queue Q with nondecreasing order on *FG-LPCC(z)*;
  end
  Let $r_b = 0$;
  while Q *is not empty and* $r_b \leq breadth$ **do**
    $r_b = r_b + 1$;
    pop the top element $\bar{z}$ in Q, and delete this element from Q;
    let $z = \bar{z}$ and $r_d = 0$;
    while *there exists any feasibility gap descent direction* Δz *for* z *and* $r_d \leq depth$ **do**
      pick a feasibility gap descent direction Δz;
      $z = z + \Delta z$;
      $r_d = r_d + 1$;
    end
    if *FG-LPCC(z)==0* **then**
      solve *LPCC(z)*, and **return** the optimal solution to *LPCC(z)*;
    end
  end
end
return feasibility recovery failed;

Procedure 1: Local search feasibility recovery process

2.2.1 Disjunctive cuts and simple cuts

These cuts exploit the disjunctive constraints: for each i, either $y_i \leq 0$ or $w_i \leq 0$. The solution to the LP relaxation typically violates a number of these disjunctions, and disjunctive cuts can either be generated by solving a supplemental linear program, or by examining the optimal tableau for the LP relaxation. Based on our computational experience, it seems that general disjunctive cuts and simple cuts [4,6,9] are the weakest cuts among our three type of cutting planes, but they are the cheapest to generate. Therefore we generate this type of cut first. The solving time of CPLEX for our test instances became worse when we added all of the generated disjunctive cuts or simple cuts to the root node even though the value of the initial LP relaxation was improved by these cuts, because the initial LP became too large. Moreover, due also to the high cost of generating general disjunctive cuts, we only generate $\lfloor m/100 \rfloor$ rounds of general

input : the known valid upper bound of LPCC $ub_{initial}$, parameter $searchGap_{min}$
output: refined feasible LPCC solution or failed to refine the known feasible solution

Initialization: Set lb_{search} =optimal value of the LP relaxation of LPCC and $ub_{search} = ub_{initial}$;
add $lb_{search} \leq c^T x + d^T y \leq ub_{search}$ into (4);
while $ub_{search} - lb_{search} > searchGap_{min}$ **do**
  solve LP relaxation of LPCC with constraints $lb_{search} \leq c^T x + d^T y \leq ub_{search}$;
  apply Procedure 1 to recover a feasible solution;
  if *recovery process succeed* **then**
    update the refined feasible solution with recovered solution;
    update ub_{search} with the newly recovered solution;
    $ub_{search} = (lb_{search} + ub_{search})/2$;
  end
  else
    $lb_{search} = (lb_{search} + ub_{search})/2$;
  end
end
if *refined feasible solution has been updated* **then**
  return refined feasible solution
end
else
  return feasibility refinement failed
end

Procedure 2: Refined local search feasibility recovery process

disjunctive cuts and for each round we only generate at most 3 general disjunctive cuts instead of generating disjunctive cuts for each violated complementarity constraint.

The values of $y_i w_i$ in the optimal solution to the LP relaxation are sorted in nonascending order and we select complementarity constraints with index that corresponds to the largest three products. After each round of generating cuts, we will remove every cut whose corresponding slack variable is basic in the relaxed LP, in order is to keep the size of the relaxed LP small. After generating the general disjunctive cuts, $\lfloor m/10 \rfloor$ rounds of simple cuts will be added. Since a simple cut is derived from the simplex tableau with almost no cost, we will generate simple cuts for every violated complementarity constraint in each round, and also remove every cut whose corresponding slack variable is basic in the relaxed LP after each round of generating cuts.

2.2.2 Bound cuts

Upper bounds u_i^y and u_i^w on y_i and w_i can be used in the bound cut

$$u_i^w y_i + u_i^y w_i \ \leq \ u_i^w u_i^y \tag{5}$$

for any pair of complementary variables y_i and w_i. Strengthening the upper bounds seems very important for the branch-and-bound routine of CPLEX for solving our instances, and the bound cuts also improve the initial lower bound dramatically. However, the major drawback of bound cuts is that they are very expensive to generate, especially when m, the number of complementarity constraints, is very large. There-

fore, we will only compute bounds for at most 5 pairs of complementary variables, and the selection of these complementary variables is the same as the selection of complementarity constraints to generate disjunctive cuts. An upper bound u_i^y for y_i can be found by solving the linear program

$$
\begin{aligned}
u_i^y \;=\; & \max_{(x,y,w)} \; y_i \\
& \text{subject to} \;\; Ax + By \geq b \\
& \qquad\qquad x \;\geq\; 0 \\
& \qquad\qquad 0 \leq y \leq u^y \\
& \qquad\qquad 0 \leq q + Nx + My \;=\; w \;\leq u^w \\
& \qquad\qquad c^T x + d^T y \leq ub \\
& \qquad\qquad u_j^w y_j + u_j^y w_j \;\leq\; u_j^w u_j^y \quad \forall\, j \text{ with known bounds } u_j^y, u_j^w
\end{aligned}
\tag{6}
$$

where ub is a known upper bound on the optimal value of the LPCC. A similar LP can be constructed to get bounds on w.

We also investigated improving the bound cuts by splitting the variables. In particular, two versions of problem (6) could be solved, one with the additional constraint $y_k = 0$ and the other with the additional constraint $w_k = 0$, for some index $k \neq i$. The maximum of the optimal values of these two problems could potentially improve on the initial upper bound. For our test instances, the additional computational work involved in computing these improved bounds did not improve the overall computational time, so this splitting is not included in our results.

2.2.3 Linear cuts from second order cone constraints

Based on the computational results of [49], cuts derived from a certain second order cone constraint can significantly improve the initial lower bound of our instances with relatively low generating cost compared to bound cuts when $n << m$, provided M is positive semidefinite. These cuts arise from linearizing the term $y^T Nx$, using McCormick inequalities [48] to tighten the linearization, and handling the $y^T My$ term appropriately. Details can be found in [50]. The constraints can be tightened by refining bounds. We did not use these cuts in the computational results reported in this paper, because of difficulties with ensuring M was regarded as numerically positive semidefinite by CPLEX.

2.3 Overall flow of the preprocessor

The preprocessor consists of the following steps:

1. Apply the feasibility recovery routine to recover a feasible solution.
2. Generate $\lfloor m/100 \rfloor$ rounds of general disjunctive cuts.
3. Generate $\lfloor m/10 \rfloor$ rounds of simple cuts.
4. Apply 4 bound refinements and generate bound cuts.

We apply Procedures 1 and 2 as the default feasibility recovery procedure due to run time considerations. Other feasibility recovery procedures and refinements can also

be invoked if required for solving special classes of problems. The number of rounds for generating each type of cutting plane can be modified by changing the parameter settings. The current setting is based on the computational experience in [49].

An additional preprocessing procedure undertaken at each node is the complementary variable fixing process, which is detailed in Sect. 3.3.

3 Branch-and-bound phase

The branch-and-bound routine needs to be invoked to solve the problem exactly if the initial LP relaxation is unbounded or the preprocessing phase is unable to close 100% of the gap for the bounded case. The branching is imposed on the complementarity constraint directly, and two subproblems (nodes) will be generated by enforcing either side of the pair of complementary variables to its lower bound zero. Just like a branch-and-bound based MIP solver, there are two key ingredients in our branch-and-bound routine: branching complementarity selection and node selection. Branching complementarity selection is the procedure to select the complementarity constraint to be branched on, and it is the same as the "variable selection" in mixed integer programming. In Sect. 3.1, we present our branching strategy which is based on the ideas of three classic branching rules and also some new proposed ideas designed for the LPCC problem. Node selection is the procedure to select the next subproblem from the node tree to be processed. In Sect. 3.2, we will present and compare different node selection strategies. Besides these two key ingredients, in Sect. 3.3 we will describe the node pre-solving procedure used in our algorithm to pre-process the nodes during the branch-and-bound process. The general branch-and-bound routine for handling the bounded case and unbounded case of LPCC are described in Sects. 4.1 and 4.2 respectively.

3.1 Branching complementarity selection

The branching rule is the key ingredient of any branch-and-bound algorithm. Good branching strategies are extremely important in practice for solving mixed integer programs, although currently there is no existing theoretical best branching strategy. We will first present three classic branching strategies for solving mixed integer programs that have been studied in the literature. The reader can refer to Linderoth and Savelsbergh [45], Fügenschuh and Martin [25] and Achterberg et al [2] for a comprehensive study of branch-and-bound strategies for mixed integer programming. We will present our branching strategy based on the ideas of these branching strategies. The computational results that compare various branching strategies will be shown in Sect. 5.

We first give some definitions related to our branching routine for the LPCC problem. For easy discussion, if the LP relaxation of the LPCC is unbounded below, we represent its lower bound as $-\infty$. Suppose that we have an LPCC problem Q and the set I is the index set of complementarity constraints. If the current solution to the LP relaxation of Q is not a feasible solution to LPCC (for the unbounded case,we consider

an unbounded ray of the LP relaxation instead of solution to the LP relaxation), then we can pick an index $i \in I$ with $y_i w_i > 0$ and obtain two subproblems (nodes): one by adding the constraint $y_i \leq 0$ (named the left child node, denoted by Q_i^y) and one by adding the constraint $w_i \leq 0$ (named the right child node, denoted by Q_i^w). We refer to this as branching on complementarity i. For the bounded case, if we denote the objective value of the LP relaxation of Q as c_Q and the objective value of the LP relaxation of its two child nodes as $c_{Q_i^y}$ and $c_{Q_i^w}$ respectively, then the objective value changes caused by branching on the ith complementarity are $\Delta_i^y = c_{Q_i^y} - c_Q$ and $\Delta_i^w = c_{Q_i^w} - c_Q$. We usually use the improvement of objective value of the LP relaxation to measure the quality of branching on the ith complementarity. Our implementation supports fixing multiple complementarity constraints at one time, but by default we will only select to branch on one complementarity. Based on the results of testing our instances and the computational results of solving various MIP problems in the literature, multiple way branching is rarely better than two way branching.

input : the LP relaxation solution of the current processing node Q or the unbounded ray to the LP relaxation if the LP relaxation is unbounded: x^*, y^*, w^*
output: the selected branching index $i \in I$ of a complementarity constraint

1. Let $\tilde{I} = \{j \in I \mid y_j^* w_j^* > 0\}$ denote the index set of violated complementarity constraints.
2. Compute a branching score $s_j \in \mathbb{R}^+$ for all candidates $j \in \tilde{I}$.
3. Select the selected branching index $i \in \tilde{I}$ with $s_i = \max_{k \in \tilde{I}}\{s_k\}$.

Return selected branching index i.

Procedure 3: Generic complementarity selection procedure

The generic procedure for selecting the branching complementarity can be described in Procedure 3. The score function in Step 2 of this procedure needs to evaluate the two child nodes that could be generated by the branching, and map these two effectiveness values onto a single score value. Different choices for the effectiveness values are given later. Suppose q^y and q^w are the effectiveness values of the two child nodes generated by a branching. In the literature, the score function usually has one of the following forms:

$$score(q^y, q^w) = (1 - \mu) \cdot \min\{q^y, q^w\} + \mu \cdot \max\{q^y, q^w\} \tag{7}$$

or

$$score(q^y, q^w) = \max\{q^y, \epsilon\} \cdot \max\{q^w, \epsilon\} \tag{8}$$

Here μ is a number between 0 and 1, and it is usually an empirically determined constant or a dynamic parameter adjusted through the course of branching process. We chose $\epsilon = 10^{-6}$ to enable the comparison when either q^y or q^w is zero. Based on the computational experience in [1], the product form is superior to the weighted sum form for solving MIP problems. Therefore, in our algorithm, we chose to use the product form to map the effectiveness values from two child nodes onto a single value.

In the following we will present three classic branching strategies for solving an MIP in terms of our branching on complementarity scheme: *Strong Branching* (apparently originally developed in the work leading up to [3]), *Pseudocost Branching* [12] and *Inference Branching* [1]. In fact, all of these branching routines are just variants of Procedure 3 with different score functions.

3.1.1 Strong branching

The idea of *Strong Branching* [3] is to test the branching candidates by temporarily enforcing either side of a complementarity constraint and solving the resulting LP relaxation to a certain level, then select the one that can lead to the largest lower bound improvement. *Full Strong Branching* will compute Δ_i^y and Δ_i^w for each branching complementarity candidate $i \in \tilde{I}$, and use the score(Δ_i^y, Δ_i^w) as the effectiveness values in the form of either (7) or (8) as its score function. *Full Strong Branching* can be seen as the locally best branching strategy in terms of lower bound improvement. However the computational cost of *Full Strong Branching* is very high, since in order to evaluate the score function for each complementarity candidate, we need to solve two resulting LP relaxations to optimality. There are usually two ways to speed up *Full Strong Branching*: one is to only test a subset of the candidate set instead of considering all the candidates, and another is to perform a limited number of simplex iterations and estimate the objective value change based on that. In our branch-and-bound algorithm, we have implemented the *Full Strong Branching* routine, and also we adopt the former idea to speed up the *Full Strong Branching*: as long as the objective value of LP relaxation of either side of the child nodes hits some threshold, we will select this branching candidate and exit the selection routine; we set the median value of the lower bound of unsolved nodes in the current search tree as this threshold.

A version of strong branching was used by Fischer and Pfetsch [22] in their branch-and-cut approach for problems with overlapping SOS1 constraints.

3.1.2 Pseudocost branching

Pseudocost Branching [12] uses the branching history to estimate the two objective changes of the child nodes without actually solving them. In other words, *Pseudocost Branching* is a branching rule based on the historical performance of complementarity branching on complementarities which have already been branched. Let ς_i^y and ς_i^w be the objective gain per unit change at node Q after branching on complementarity i by enforcing y_i or w_i to zero, that is

$$\varsigma_i^y = \frac{\Delta_i^y}{y_i^*} \text{ and } \varsigma_i^w = \frac{\Delta_i^w}{w_i^*} \tag{9}$$

where y_i^* and w_i^* are the violation of complementarity i corresponding to the LP relaxation solution of Q. Let σ_i^y denote the sum of ς_i^y over all the processed nodes where complementarity i has been selected as the branching complementarity and resulting child node Q_i^y has been solved and was feasible. Let η_i^y denote the number

of these problems, and define σ_i^w and η_i^w in the same way for the other side of the complementarity. Then the pseudocost of branching on complementarity i can be calculated as the arithmetic mean of objective gain per unit change:

$$\Psi_i^y = \frac{\sigma_i^y}{\eta_i^y} \text{ and } \Psi_i^w = \frac{\sigma_i^w}{\eta_i^w} \tag{10}$$

Therefore given the violated complementarity i corresponding to the LP relaxation of Q, it is reasonable to use $\Psi_i^y \cdot y_i^*$ and $\Psi_i^w \cdot w_i^*$ to estimate Δ_i^y and Δ_i^w respectively. We call the branching rule that uses the score function $score(\Psi_i^y \cdot y_i^*, \Psi_i^w \cdot w_i^*)$ in step 2 of Procedure 3 as *Pseudocost Branching*. Notice that at the beginning of the branch-and-bound procedure, the pseudocost is uninitialized for all the complementarities. One way to handle a complementarity with an uninitialized pseudocost is to replace its pseudocost with the average of the pseudocosts of the complementaries whose pseudocosts have been initialized, and set the pseudocost as 1 if all the complementarities are uninitialized. Applying strong branching to the nodes whose tree depth level is less than a given level is another way to initialize the pseudocosts. More recently, Achterberg et al [2] proposed a more general pseudocost initialization method, and named the corresponding branching rule as *Reliability Branching*. In our implementation, we include the pseudocost as part of our branching score, and we choose to apply strong branching to nodes whose tree depth level is less than 7 to initialize the pseudocost.

3.1.3 Inference branching

The branching decision of strong branching and pseudocost branching are both based on the change of objective value of the LP relaxation, while *Inference Branching* [1] is quite different from the above two branching strategies. *Inference Branching* checks the impact of branching on changing the bounds of other variables. As with pseudocosts, historical information is typically used to estimate the deductions on bounds of the variables, and the inference value can be calculated as the arithmetic mean of the number of bound deductions. The domain propagation process is a node pre-solving process to detect the bound change of the variables and is discussed in Sect. 3.3. In our implementation, we use a similar idea to inference branching: instead of evaluating the inference value, we estimate the *complementarity satisfaction level* after branching on a complementarity, leading to the quantity s_i^{SL} below.

3.1.4 Hybrid branching strategy for the LPCC (bounded case)

Our branching strategy for the bounded case combines the ideas of the above three classic branching strategies, and additionally we also include some new score values into our branching score function which are specialized for the LPCC problem.

In our implementation, the default branching strategy will apply the full strong branching strategy for the nodes whose depth level are no larger then 7. The reason for doing that is because it is usually quite important to make the right branching decision at the beginning, and also we can use strong branching to initialize the pseudocosts

and another score value that we will propose next. For the nodes whose tree depth are larger than 7, we will use a weighted sum formula to combine four score values for each violated complementarity. Among these four score values, two of them are only based on the current node Q, and the other two are based on historical branching information. For the violated complementarity i, these four score values are listed as follows:

1. s_i^{VL}: score of Violation Level. Suppose y_i^* and w_i^* are the violation of complementarity i corresponding to the LP relaxation of Q, then we define

$$s_i^{VL} = \sqrt{y_i^* \cdot w_i^*}$$

2. s_i^{ED}: score of Euclidean Distances from the LP relaxation solution of Q to the two hyperplanes corresponding to $y_i = 0$ and $w_i = 0$. Recall that since $y_i^* \cdot w_i^* > 0$, we can represent the complementary variables y_i and w_i with the non-basic variables in the optimal simplex tableau of Q

$$y_i = y_i^* - \sum_{j \in NB} a_j^{yi} \xi_j \tag{11}$$

$$w_i = w_i^* - \sum_{j \in NB} a_j^{wi} \xi_j \tag{12}$$

We use the Euclidean distance from the LP relaxation solution to the two hyperplanes

$$\sum_{j \in NB} a_j^{yi} \xi_j = y_i^* \text{ and } \sum_{j \in NB} a_j^{wi} \xi_j = w_i^*$$

to define s_i^{ED} as follows:

$$s_i^{ED} = \sqrt{\frac{y_i^* \cdot w_i^*}{\sqrt{\|a^{yi}\| \cdot \|a^{wi}\|}}}$$

3. s_i^{PC}: score of Pseudo Cost. We use the following small modification to the pseudcost calculation of Sect. 3.1.2:

$$s_i^{PC} = \sqrt{\max\{\Psi_i^y \cdot y_i^*, \epsilon\} \cdot \max\{\Psi_i^w \cdot w_i^*, \epsilon\}}$$

4. s_i^{SL}: score of complementarity Satisfaction Level. We define the complementarity satisfaction level as the proportion of the satisfied complementarities corresponding to the LP relaxation solution of the child node after branching. Intuitively we want to select a branching complementarity that will lead to more satisfied complementarities. To estimate this complementarity satisfaction level, we collected

the historical information to compute the average complementarity satisfaction level for both sides of the complementarity

$$\Phi_i^y = \frac{\varphi_i^y}{\eta_i^y} \text{ and } \Phi_i^w = \frac{\varphi_i^w}{\eta_i^w}$$

Here φ_i^y is the sum of the proportion of complementarity satisfaction levels over all the prior nodes, where complementarity i has been selected as the branching complementarity, and η_i^y is the total number of these nodes. We define φ_i^w and η_i^w to be the analogous value for the other side of complementarity. Then the score of the complementarity Satisfaction Level can be calculated as

$$s_i^{SL} = \sqrt{\Phi_i^y \cdot \Phi_i^w}$$

We scale the score vectors using their 2-norms, and the following formula is the branching score function that we used to evaluate the score for each violated complementarity:

$$s_i = \omega^{VL}\left(\frac{s_i^{VL}}{\|s^{VL}\|}\right) + \omega^{ED}\left(\frac{s_i^{ED}}{\|s^{ED}\|}\right) + \omega^{PC}\left(\frac{s_i^{PC}}{\|s^{PC}\|}\right) + \omega^{SL}\left(\frac{s_i^{SL}}{\|s^{SL}\|}\right) \quad (13)$$

By default, the weight is set as $\omega^{VL} = 1$, $\omega^{ED} = 0.5$, $\omega^{PC} = 0.25$ and $\omega^{SL} = 0.5$. Note that setting different weights for each score value will lead to different branching behaviour. In Sect. 5.2, we will show the computational results of solving our LPCC instances with different weights of the score value.

3.1.5 Hybrid branching strategy for LPCC (unbounded case)

Our branching strategy for the unbounded case is slightly simpler than the one for the bounded case. We will still apply full strong branching to the nodes whose tree depth level is no larger than 7. However, for the remaining unbounded nodes we will only use s_i^{VL} as the branching score to make the branching decision.

3.2 Node selection

In addition to selecting which complementarity to branch on, another question is which subproblem (node) we should pick to process. There are two major criteria for selecting the next subproblem to be processed.

1. Finding feasible LPCC solutions to improve the upper bound of the LPCC problem which leads to pruning the nodes by bounding, leading to a Depth First Search strategy.
2. Improving the lower bound as fast as possible, leading to a Best-Bound strategy.

In our implementation of the branch-and-bound routine we use a ***Best-Bound*** strategy to select the next node to be processed, since we want to solve the problem to optimality as fast as possible. Notice that for the *Best-Bound*, it is possible that there are several nodes with the same lower bound. For that case, we will select the most recently generated node as the next node to be processed.

3.3 Node pre-solving

The major task of our node pre-solving procedure is to tighten the domains of complementary variables y_i and w_i and try to fix the complementary variables. In order to facilitate the discussion, here we can assume that each ξ_i in (11) and (12) is a non-negative variable with zero lower bound. Therefore we have the following result: if $a_j^{yi} \leq 0, \forall j \in NB$, then we have $y_i \geq \hat{y}_i$, and therefore $w_i = 0$; if $a_j^{wi} \leq 0, \forall j \in NB$, then we have $w_i \geq \hat{w}_i$, and therefore $y_i = 0$. This complementary variable fixing check is performed before we branch on the complementarity constraint.

4 General scheme of the branch-and-cut algorithm for solving LPCC

The preprocessing routines are only invoked if the the initial LP relaxation of the LPCC has a bounded optimal value; we refer to this as the "bounded case". If the initial relaxation does not have a finite optimal value then we are in the "unbounded case". For the bounded case, the preprocessing procedure is applied first to tighten the initial LP relaxation, then the branch-and-bound routine is invoked to solve the LPCC to optimality; for the unbounded case, we will only apply the branch-and-bound routine, which gives unbounded nodes higher priority than bounded nodes. A flow diagram of the overall algorithm is given in Fig. 1. The initialization step 0 sets the upper bound $\bar{z} = +\infty$, the lower bound $\underline{z} = -\infty$, the unbounded node list $\bar{L} = \emptyset$, and the bounded node list $L = \emptyset$. If the LP relaxation of the initial problem is feasible then the initial problem is added to L or $\bar{L}$ in box 1, as appropriate. Boxes 2, 4, 6, 8, 10, 12, and 14 corresponding to the bounded case are the subject of Sect. 4.1, with the unbounded case boxes 3, 5, 7, 9, and 11 explained in Sect. 4.2. Box 13 is discussed in Sect. 4.3.

4.1 Overall flow of branch-and-bound for LPCC (bounded case)

In the bounded case, the algorithm is quite similar to the branch-and-bound routine for a mixed integer program. If it is determined in box 1 that the initial LP relaxation is bounded then we implement a more detailed preprocessing step in box 2, as discussed in Sect. 2. In box 4, we apply *Best-Bound* to pick the next node $LPCC^i$ from L to be processed and delete $LPCC^i$ from L. The node presolving procedure from Sect. 3.3 is implemented in box 6. The branching strategy of Sect. 3.1.4 is used in box 8 to select branching complementarity j. Fathoming and pruning is performed in box 10 as follows:

Fathoming and pruning Generate two child nodes by enforcing either $y_j = 0$ or $w_j = 0$ and solve LP relaxations. For each child node:

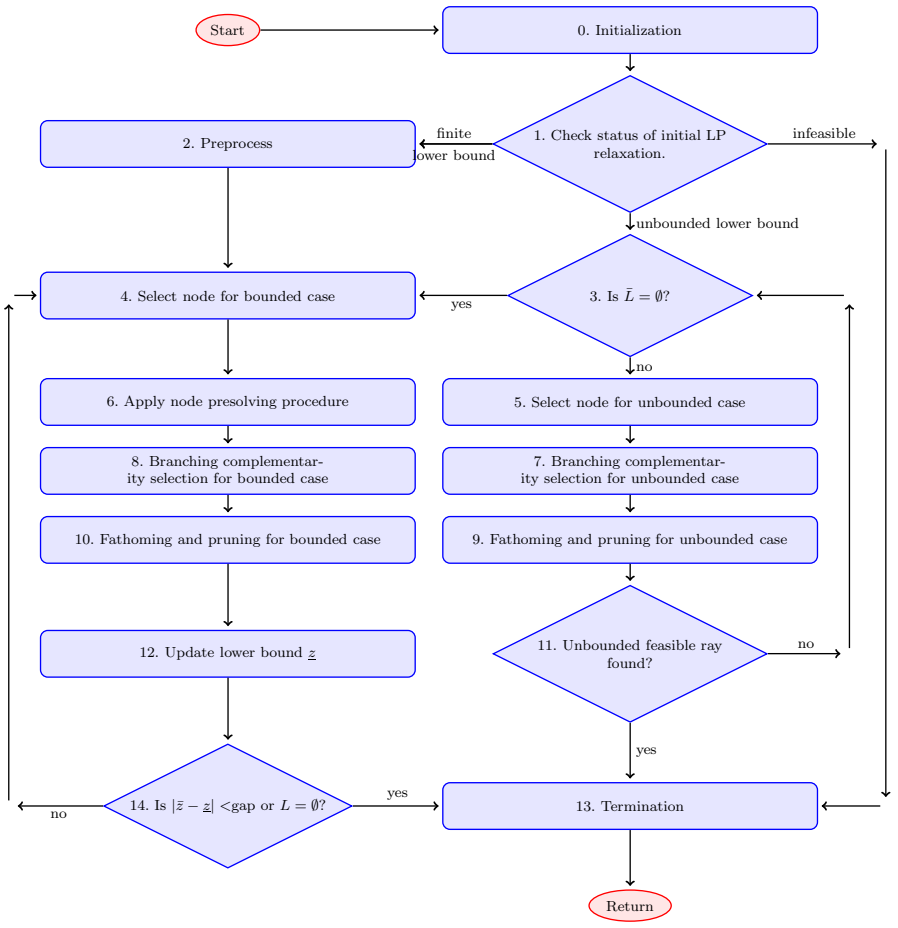


Fig. 1 Flow chart of branch-and-bound procedure

1. If LP relaxation solution is feasible in LPCC with objective z^* then delete child node. Set $\bar{z} \leftarrow \min\{\bar{z}, z^*\}$.
2. If LP relaxation is feasible with objective $z^* < \bar{z}$ then set the lower bound of child node as z^* and add child node to L.
3. If LP relaxation feasible with objective $z^* \geq \bar{z}$ or infeasible then delete child node.

The lower bound is updated in box 12. The procedure is terminated in box 14 if there are no more nodes in the set L or if the gap between the upper and lower bound is sufficiently small.

4.2 Overall flow of branch-and-bound for LPCC (unbounded case)

The branch-and-bound routines for solving mixed integer programs in existing MIP solvers like CPLEX usually assume the initial LP relaxation is bounded below. Even

if the initial LP relaxation is unbounded, it is still treated as bounded below by adding an objective lower bound constraint with a very large negative number (-10^{20}) as its lower bound. However, our branch-and-bound routine for handling the unbounded case of the LPCC is quite different. If the LP relaxation of a node is unbounded, we will treat this node as an unbounded node and add it to the unbounded node list. If the unbounded node list is non-empty, our branch-and-bound routine will always process a node in the unbounded node list first. Notice that when we find an unbounded ray that satisfies all the complementarities, we need to check whether this is a feasible ray to the LPCC. The LPCC is feasible with unbounded objective value if and only if we find an unbounded feasible ray to the LPCC.

If the set $\bar{L}$ of unbounded nodes is empty in box 3 then we return to the bounded case in box 4, constructing an appropriate lower bound $\underline{z}$. In box 5, we select the node $LPCC^i$ that is the most recently generated from $\bar{L}$ to be processed and delete $LPCC^i$ from $\bar{L}$. The branching strategy of Sect. 3.1.5 is used in box 7 to select branching complementarity j. Fathoming and pruning for an unbounded node is performed in box 9 as follows:

Fathoming and pruning Generate two child nodes by enforcing either $y_j = 0$ or $w_j = 0$ and solve LP relaxations. For each child node:

1. If LP relaxation solution is feasible in LPCC with objective z^* then delete child node. Set $\bar{z} \leftarrow \min\{\bar{z}, z^*\}$.
2. If LP relaxation is feasible with objective $z^* < \bar{z}$ then set the lower bound of child node as z^* and add child node to the bounded node list L.
3. If LP relaxation feasible with objective $z^* \geq \bar{z}$ or infeasible then delete child node.
4. If LP relaxation is unbounded and the unbounded ray is not a feasible ray to LPCC then add this child node to the unbounded node list $\bar{L}$.
5. If LP relaxation is unbounded and the piece of LPCC corresponding to that ray is feasible then the LPCC is unbounded.

If an unbounded piece is found in box 11 then the algorithm can be terminated; otherwise we loop back to box 3.

4.3 The complete overall scheme

A flow chart of the algorithm is exhibited in Fig. 1. Each of the three possible problem states can be returned in the termination box 13. If an unbounded feasible ray to the LPCC is found then the LPCC is feasible with unbounded objective value. If the LPCC is not unbounded and an LPCC feasible solution is found then the LPCC attains a finite optimal solution with optimal objective $\bar{z}$. Otherwise, the problem is infeasible.

5 Computational results

In this section, we will present the computational results of using our proposed branch-and-cut algorithm to solve various LPCC instances. All procedures and algorithms are

developed in the C language with the CPLEX callable library, and all LPs and convex quadratic constraint programs are solved using CPLEX 12.6.2. We implement our algorithm through the addition of callback routines to CPLEX. As an alternative to our approach, CPLEX allows the modeling of complementarity constraints through the use of *indicator constraints*; we compare the computational performance of our algorithm with that of using default CPLEX 12.6.2 to solve indicator constraint formulations of these LPCC instances, with our preprocessor used for both approaches. Except for a few preliminary tests discussed in Sect. 5.2, all the computational testing is performed on a Mac Pro with 6 dual processor Intel Xeon E5 cores and 16GB of memory. Our branch-and-cut routine uses just one thread, while the default CPLEX 12.6.2 indicator constraint formulation can use all 12 available threads. The relative gap for optimality is 10^{-6}, here the relative gap is defined as $\frac{upperbound-lowerbound}{\max(1,|lowerbound|)}$. This is smaller than CPLEX's default MIP relative gap optimality tolerance of 10^{-4}; many of the optimal values are non-integral, so we found a tolerance of 10^{-4} to be insufficiently accurate. The tolerance of complementarity is 10^{-6}, i.e., either y_i or w_i for $i = 1, \ldots, m$ should be less than 10^{-6} for any feasible LPCC solution. All runtimes are reported in seconds.

We used three sets of test instances. The first set consists of 60 LPCC instances with $n = 2$ and between 100 and 200 complementarities. The generation scheme for these problems and computational results can be found in "Appendix A", with the results discussed in Sects. 5.1 and 5.2. The second set of test instances are LPCC formulations of bilevel programs, where the lower level problem is a convex quadratic program; the formulation and results are presented in Sect. 5.3, with more extensive results in "Appendix B". The final set of results in Sect. 5.4 are for inverse quadratic programming problems, with detailed results in "Appendix C".

Source code and test instances can be found online at https://github.com/mitchjrpi/LPCCbnc Also included with the source code is a Makefile. A user needs to have access to CPLEX in order to be able to compile the code. Generators for the bilevel and inverse QP problems can be found on the website; the generator uses AMPL to construct the instances.

5.1 Computational results of the feasibility recovery process

We will first apply the local search feasibility recovery process (procedure 1); if this procedure successfully recovers a feasible solution, then the refinement procedure (Procedure 2) will be applied to refine that feasible solution. We set the depth parameter as 5 and breadth parameter as m, i.e. the number of complementarities, in Procedure 1. Table 1 summarizes the feasibility recovery result of the 60 LPCC instances. The computational results show that our proposed feasibility recovery procedures can successfully recover a feasible solution for all of the 60 LPCC instances with very good quality. For most instances, the recovered feasible solution is in fact an optimal solution. Note that as m increases, the feasibility recovery processing time increases as well. Therefore in practice, as a preprocessing procedure, we need to control the depth and breadth parameters in procedure 2 to reduce the time spent on the feasibility recovery procedure.

Table 1 Average computational results of feasibility recovery with $n = 2$, $k = 20$. The column "average gap" is calculated as $\frac{LB_{recovered} - LPCC_{opt}}{LPCC_{opt}}$. Detailed results can be found in Table 5 in "Appendix A"

m	$RankM$	Average gap (%)	Optimal found out of 10
100	30	0.09	5
100	60	0.22	2
150	30	0.0	10
150	100	0.06	3
200	30	0.0	10
200	120	0.07	2

Table 2 Comparison of geometric means of solving time, using our four different branching rules and using default CPLEX

m	$Time_{R_1}$ (s)	$Time_{R_2}$ (s)	$Time_{R_3}$ (s)	$Time_{R_4}$ (s)	$Time_{CPLEX}$ (s)
100	19.185	18.790	18.805	18.917	38.021
150	75.213	73.623	76.071	74.285	1688.160
200	308.293	296.663	287.232	291.057	5043.017

5.2 Computational results of branch-and-cut algorithm

In this section, we will show the computational results of using our proposed branch-and-cut algorithm to solve the 60 LPCC instances with finite global optimal values from "Appendix A".

We conducted preliminary experiments with 4 different weight settings of increasing sophistication to choose a score function (13). Each rule is a refinement of the previous one, and led to improvement on the largest instances tested:

R_1: $\omega^{VL} = 1, \omega^{ED} = 0, \omega^{PC} = 0$ and $\omega^{SL} = 0$;
R_2: $\omega^{VL} = 1, \omega^{ED} = 0.5, \omega^{PC} = 0$ and $\omega^{SL} = 0.5$;
R_3: $\omega^{VL} = 1, \omega^{ED} = 0.5, \omega^{PC} = 0.25$ and $\omega^{SL} = 0.5$;
R_4: $\omega^{VL} = 1, \omega^{ED} = 0.5, \omega^{PC} = 0.25$ and $\omega^{SL} = 0.5$ and apply strong branching rule to the node whose tree depth is less or equal to 7.

These results were obtained using CPLEX 11.4 using a single core of AMD Phenom II X4 955 CPU @ 3.2GHZ, 4GB memory and are contained in Tables 2 and 3. All four rules required far fewer nodes than default CPLEX. Based on these results, R_4 is the best branching rule in terms of the number of nodes. Since in terms of solving

Table 3 Comparison of geometric means of number of nodes in branch-and-cut tree, using our four different branching rules and using default CPLEX

m	$Node_{R_1}$	$Node_{R_2}$	$Node_{R_3}$	$Node_{R_4}$	$Node_{CPLEX}$
100	213	215	212	196	39114
150	831	863	848	757	1078311
200	3408	3301	3161	2837	1494577

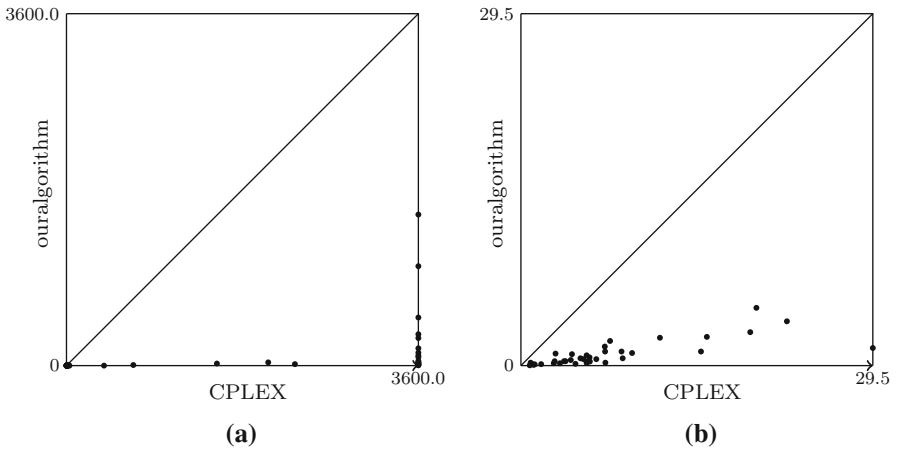


Fig. 2 Scatter plots for CPU time (seconds) for solution of LPCCs. Horizontal axis is time for the default CPLEX indicator constraint solver, vertical axis is time for our branch and cut algorithm. Processing times excluded. **a** All 60 instances. **b** 37 LPCCs where default CPLEX MIP required no more than 30 s

time, these 4 routines are quite close, we chose R_4 as our default branch-and-bound routine.

All remaining results in the paper were obtained using CPLEX 12.6.2 with detailed results contained in Table 6 in "Appendix A". A scatter plot of the CPU time for solving the instances is given in Fig. 2. Performance profiles [15] are given in Fig. 3. The preprocessing times have been excluded from these plots. All the LPCC instances can be solved by our algorithm within 30 min, with 90% of them (54/60) solved within 150 s. Each instance requires considerably less processing time with our algorithm than with default CPLEX. Notice that default CPLEX is only able to solve 42 of the 60 instances within 3600 s. In particular, it is unable to solve 11 of our 20 LPCC instances when $m = 200$ within this time limit.

The determination of a valid disjunctive cut or bound cut requires the solution of a linear programming problem. The parameter choices given in Sect. 2.2 result in $0.3m$ disjunctive cuts, approximately $5m$ simple cuts, and 15 bound cuts for each instance. We also experimented with not adding cutting planes in the preprocessor, in which case both codes performed slightly worse for the larger instances (a difference of perhaps 10% in average runtime for our branch-and-cut code).

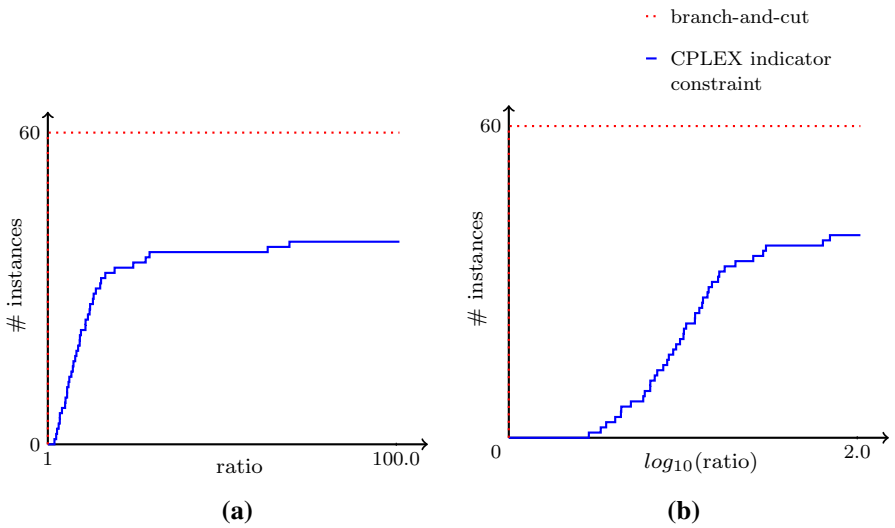


Fig. 3 Performance profile for CPU time (seconds) for solution of 60 LPCCs (preprocessing time excluded). Vertical axis is the number of instances. Horizontal axis is ratio of time required by the given algorithm to the time required by the better algorithm. **a** Linear scale. **b** Log scale

5.3 Bilevel test problems

We further tested our algorithm on bilevel problems of the form

$$\begin{array}{ll} \min_{x,v} & c^T x + d^T v \\ \text{subject to} & Ax + Bv \geq b \\ & 0 \leq v \leq u \\ & x \in \operatorname{argmin}_x \{\frac{1}{2} x^T Q x + v^T x : Hx \geq g,\ x \geq 0\} \end{array} \tag{14}$$

where Q is positive semidefinite. The variables v are first stage variables, with the second stage variables x chosen to optimize a convex quadratic subproblem that depends on v. Both sets of variables appear in the linear objective. In addition, the first and second stage variables must satisfy the linking constraint $Ax + Bv \geq b$. By introducing KKT multipliers y and λ for the constraints in the subproblem, we can model this problem equivalently as the LPCC

$$\begin{array}{lrcl} \min_{x,v,y,\lambda,w} & c^T x + d^T v & & \\ \text{subject to} & Ax + Bv & & \geq b \\ & Qx + v - H^T y - \lambda & & = 0 \\ & 0 \leq v & & \leq u \\ & 0 \leq \lambda \perp x & & \geq 0 \\ & 0 \leq y \perp w := Hx - g & & \geq 0, \end{array}$$

a problem equivalent to one in our standard form (1). The relationship between the dimensions in (1) and the dimensions of the variables and constraints in (14) is as follows:

Dimensions	
(1)	(14)
m	dimension(g) + dimension(v)
n	2 × dimension(v)
k	dimension(b) + 3 × dimension(v)

Thus, the number of complementarity constraints is equal to the sum of the dimensions of g and v.

In our experiments, all parameters in b, c, d, g, A, B, and H were uniformly generated in the interval (0, 1). The matrix Q was equal to the matrix product LL^T, where the number of columns in L is equal to the required rank of Q and each entry in L is chosen uniformly from the interval $(-1, 1)$. Each entry of u was equal to 1. Repeated problem dimensions in the table correspond to different instances. The dimension of g varied from 50 to 200, the dimension of v and x varied from 50 to 100, the number of complementarity constraints varied from 100 to 250, the dimension of b varied from 25 to 100, and the rank of Q varied between 0.5 of the dimension of v and the dimension of v. Problem data for the 90 bilevel test instances can be found in Tables 7 and 8.

We gave each algorithm a time limit of 3600 s in addition to the preprocessing time. Detailed performance data can be found in Tables 9 and 10. Our algorithm was able to solve all 63 instances with dimension of v equal to 50, 16/24 of the instances with the dimension of v equal to 75, and 3/3 of the instances with the dimension of v equal to 100. The corresponding numbers for the default CPLEX indicator constraint code were 56/63, 2/24, and 1/3. Our algorithm was considerably faster than default CPLEX indicator constraint code on every instance. Further, it had a smaller final gap than default CPLEX indicator constraint code for each instance where neither code could solve the problem. There was no instance that could be solved by default CPLEX indicator constraint code which could not also be solved by our algorithm. A scatter plot of the CPU time for solving the instances (ignoring the common preprocessing time) is given in Fig. 4 and a performance profile is in Fig. 5.

The instances become more difficult as the dimensions of v, b, and g increase, as might be expected. The instances also become more difficult as the rank of Q increases. Table 4 contains averages of solution times over these different parameters for the instances with the dimension of v equal to 50.

The parameter choices given in Sect. 2.2 result in $0.3m$ disjunctive cuts, approximately $3m$ simple cuts, and 15 bound cuts for each instance. Also as in Sect. 5.2, we experimented with not adding cutting planes in the preprocessor. Both codes performed similarly to their respective performance with the preprocessor. Thus, based on the results in this section and Sect. 5.2, our default implementation is to generate cutting planes in the preprocessor.

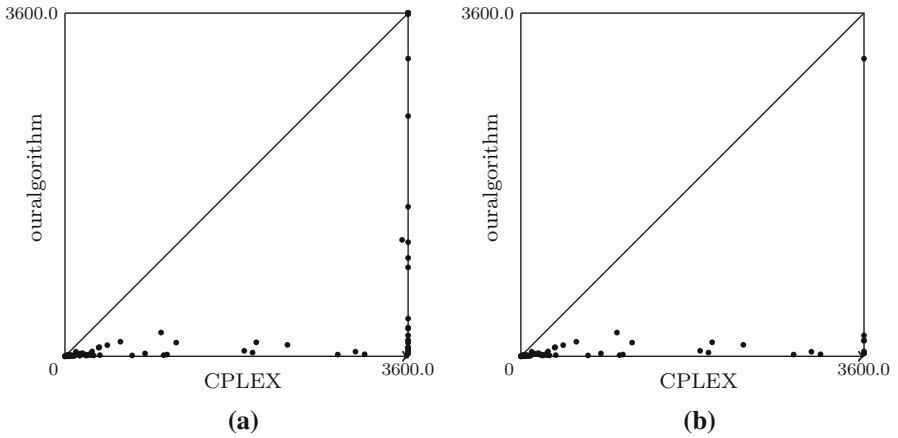


Fig. 4 Scatter plots for CPU time (seconds) for solution of LPCCs based on bilevel instances Horizontal axis is time for default CPLEX indicator constraint solver, vertical axis is time for our branch-and-cut solver. Processing times excluded. **a** All 90 instances. **b** 63 instances with $n = 50$

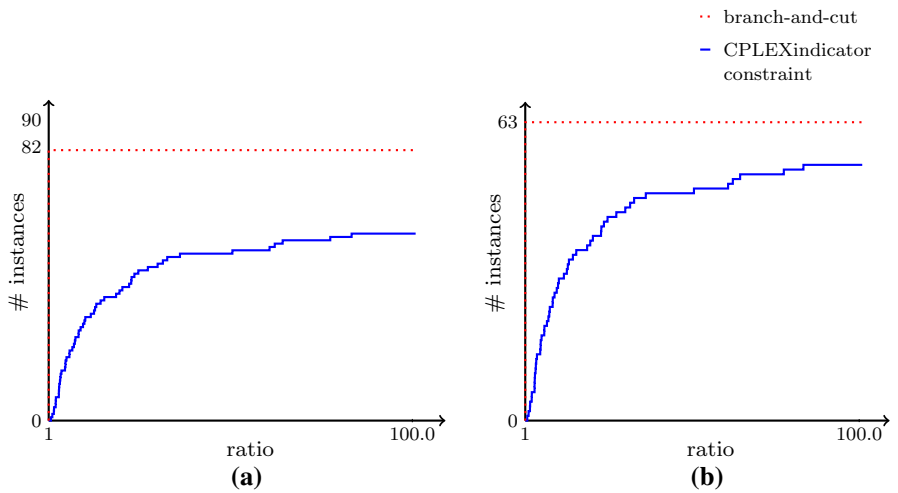


Fig. 5 Performance profile with linear scale for CPU time (seconds) for solution of LPCCs based on bilevel instances (preprocessing time excluded). Vertical axis is the number of instances. Horizontal axis is ratio of time required by the given algorithm to the time required by the better algorithm. **a** All 90 instances. **b** 63 instances with $n = 50$

5.4 Inverse quadratic programs

Jara-Moroni et al. [37] presented a DC method for finding local optima for LPCCs arising from inverse quadratic programs [33]. The problem of interest has the form

$$\begin{array}{ll} \min_{x,b,c} & ||(x,b,c) - (\bar{x},\bar{b},\bar{c})||_1 \\ \text{s.t.} & x \in \operatorname{argmin}_y\{\frac{1}{2}y^T Qy + c^T y \ : \ Ay \geq b\} \\ & (x,b,c) \in P \end{array} \tag{15}$$

Table 4 Average performance on bilevel instances with 50 first stage variables. Each column contains results from all 63 instances. Each average is taken over instances where the other parameters are varied

Dimension of g			Dimension of b			Rank of Q		
Dim	Time	# Instances	Dim	Time	# Instances	Rank	Time	# Instances
50	12.76	16	25	39.98	16	25	19.94	31
100	30.29	15	50	51.06	16	50	161.66	32
150	41.91	16	75	15.11	16			
200	266.45	16	100	259.57	15			

where $\bar{x}$, $\bar{b}$, and $\bar{c}$ are observations of the parameters and solution of a quadratic program and P is a polyhedron. The objective is to find (x, b, c) close to the observed values where x does solve the lower level quadratic program. In our computational testing, we varied the number of rows $\tilde{m}$ and columns $\tilde{n}$ of A between 100 and 400 and between 5 and 90, respectively; the dimensions of all other vectors and matrices are determined by the dimensions of A. When the matrix Q is positive definite, the inverse QP is equivalent to the following LPCC:

$$\begin{array}{ll} \min_{x,b,c,z^x,z^b,z^c,\lambda} & \mathbf{1}^T z^x + \mathbf{1}^T z^b + \mathbf{1}^T z^c \\ \text{s.t.} & Qx + c - A^T\lambda = 0 \\ & x + z^x \geq \bar{x},\ -x + z^x \geq -\bar{x} \\ & b + z^b \geq \bar{b},\ -b + z^b \geq -\bar{b} \\ & c + z^c \geq \bar{c},\ -c + z^c \geq -\bar{c} \\ & (x, b, c) \in P \\ & 0 \leq \lambda \perp w := Ax - b \geq 0 \end{array} \tag{16}$$

where λ is the vector of KKT variables for the inner QP, the variables z^x, z_b, z^c are used to represent the L_1 objective function in (15), and $\mathbf{1}$ represents a vector of ones of an appropriate dimension.

The instances in [37] were generated in MATLAB, whereas our instances were generated using AMPL. Nonetheless, we closely followed their procedures except for the generation of Q. Our matrix $Q \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ was formed as the product MM^T, where $M \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ was a square matrix with exactly three nonzeroes per row, with diagonal entries uniformly distributed between 0.5 and 1 and two off-diagonal entries uniformly distributed between 0 and 1; this results in a positive definite matrix Q, with about 9 entries per row on average (similar to the number of nonzeroes in a row of Q from [37]). Other parameters were generated as in [37]: the matrix $A \in \mathbb{R}^{\tilde{m}\times\tilde{n}}$ has an average of approximately 10 nonzero entries per row which are uniformly distributed between 0 and 1; a vector $\tilde{x} \in \mathbb{R}^{\tilde{n}}$ has components distributed as Normal(0,1); vectors $\hat{\lambda} \in \mathbb{R}^{\tilde{m}}$ and $\hat{w} \in \mathbb{R}^{\tilde{m}}$ have components uniformly distributed between 0 and 10; a binary vector $v \in \mathbb{B}^{\tilde{m}}$ is generated and $\tilde{\lambda} \in \mathbb{R}^{\tilde{m}}$ and $\tilde{w} \in \mathbb{R}^{\tilde{m}}$ are constructed as the Hadamard products $\tilde{\lambda} := \hat{\lambda} \bullet v$ and $\tilde{w} := \hat{w} \bullet (\mathbf{1} - v)$; vectors $\tilde{b} \in \mathbb{R}^{\tilde{m}}$ and $\tilde{c} \in \mathbb{R}^{\tilde{n}}$ are defined as $\tilde{b} := A\tilde{x} - \tilde{w}$ and $\tilde{c} = A^T\tilde{\lambda} - Q\tilde{x}$; vectors $\bar{x} \in \mathbb{R}^{\tilde{n}}$, $\bar{b} \in \mathbb{R}^{\tilde{m}}$, and $\bar{c} \in \mathbb{R}^{\tilde{n}}$ are obtained by perturbing $\tilde{x}$, $\tilde{b}$, and $\tilde{c}$ respectively, using Normal (0,1) noise; the polyhedron P is constructed as a box using simple bounds $-u^x\mathbf{1} \leq x \leq u^x\mathbf{1}$, $-u^b\mathbf{1} \leq b \leq u^b\mathbf{1}$,

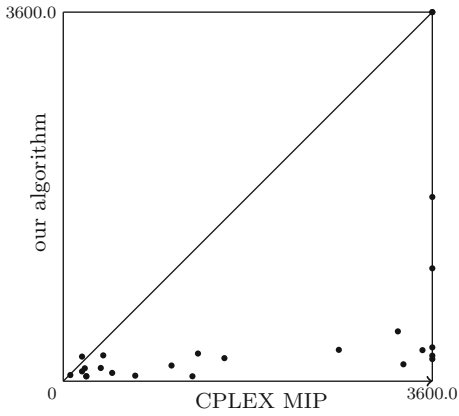


Fig. 6 Scatter plots for CPU time (seconds) for solution of inverse quadratic programs. Horizontal axis is time for CPLEX MIP called from AMPL and run on a single thread, vertical axis is time for our branch and cut algorithm

$-u^c\mathbf{1} \le c \le u^c\mathbf{1}$ with $u^x = 10\max\{|\tilde{x}_i|\}$, $u^b = 10\max\{|\tilde{b}_i|\}$, $u^c = 10\max\{|\tilde{c}_i|\}$; finally, upper bounds are also imposed on λ with $u^\lambda = 10\max\{|\tilde{\lambda}_i|\}$. The point $(\tilde{x}, \tilde{b}, \tilde{c})$ with $\tilde{\lambda}$ is feasible in the resulting problem instances of (16).

It is easy to generate explicit upper bounds on $w = Ax - b$ from the upper bounds on x and b. Also, explicit upper bounds on λ are imposed following [37]. Thus, this problem can be formulated directly as a mixed integer program of the form (2). Because of this observation, our comparisons in this section are somewhat different from the previous experiments. In particular, we make the following two changes:

- Since bounds are already available, we do not use the cutting plane generation features of the preprocessor.
- We compare our LPCC branch-and-cut code with the CPLEX MIP solver invoked from AMPL, run with a single thread.

Our testbed consisted of 5 sets of 5 instances: $(\tilde{m}, \tilde{n})$ equal to (100, 75), (120, 90), (150, 20), (200, 15), and (400, 5). A scatter plot of the results can be found in Fig. 6 and performance profiles can be found in Fig. 7. Detailed computational results are contained in the Appendix, in Table 11. Our algorithm was able to solve 23 of the 25 instances within the 3600 s time limit; the corresponding figure for CPLEX was 18 out of 25. There was only one instance where CPLEX outperformed our code. Our algorithm solved 20 of the 25 instances within 360 s, while CPLEX only solved 6 of the instances within this time window.

6 Conclusions

The optimal solution to a linear program with complementarity constraints can in principle be found directly using CPLEX. However, far better performance can often be obtained by adding good cutting planes, by incorporating a specialized feasibility recovery routine, and especially by designing good branching routines. Our computational results demonstrate that our code is at least an order of magnitude faster than a default version of CPLEX, at least for our test set of instances. It is able to solve instances with up to 400 complementarity constraints in reasonable amounts of time,

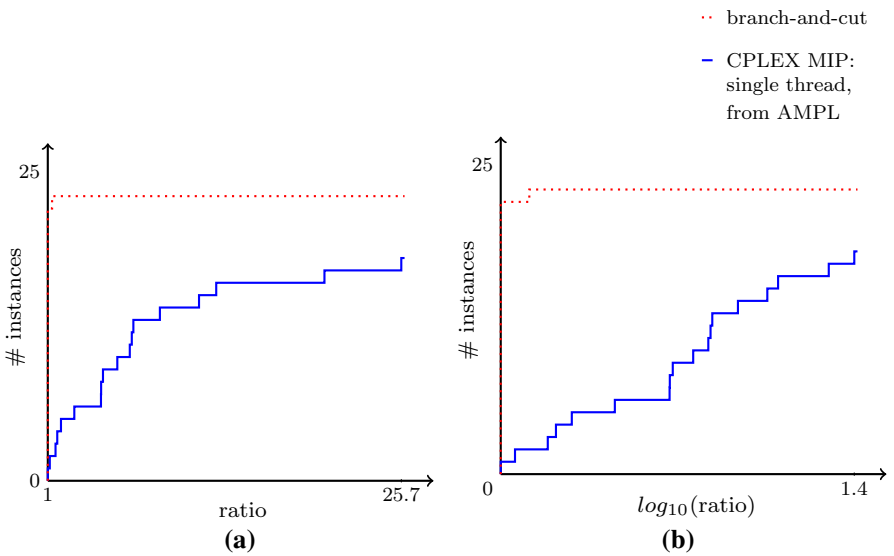


Fig. 7 Performance profile for CPU time (seconds) for solution of 25 inverse quadratic programs. Vertical axis is the number of instances. Horizontal axis is ratio of time required by the given algorithm to the time required by the better algorithm. Time limit 3600 s. **a** Linear scale. **b** Log scale

and can reliably solve instances with 100 complementarity constraints in less than a minute.

A LPCC test instances

See Tables 5 and 6.

Table 5 Objective function data for the 60 instances. All instances have $n = 2$ and $k = 20$. The number of complementarities is m. The rank of M and the density of each matrix are indicated. Three relative gaps are given as percentages: (1) the gap between the upper and lower bounds obtained through preprocessing, (2) the gap between the upper bound obtained from feasibility recovery and the optimal value of the LPCC, and (3) the improvement in the gap between upper and lower bound effected by the improvement in the LP relaxation obtained through preprocessing

#	m	Rank	Dense	Optimal value	LP relaxation	Preprocessed bounds		Relative gaps (percentages)		
						Lower	Upper	UB-LB	UB-opt	%Closed
1	100	30	70	769.911528	629.002874	669.22439	770.287	13.13	0.05	71.46
2	100	30	70	752	650.929154	723.79249	754.658	4.10	0.35	27.91
3	100	30	70	690.306012	627.332027	657.571025	691	4.84	0.10	51.98
4	100	30	70	543	531.188245	539.856029	544.497	0.85	0.28	26.62
5	100	30	70	930	771.820799	896.57115	930.917	3.69	0.10	21.13
6	100	30	20	589	583.487434	588.868287	589	0.02	0.00	2.39
7	100	30	20	488	425.717966	459.942655	488	5.75	0.00	45.05
8	100	30	20	771	687.744893	745.909078	771	3.25	0.00	30.14
9	100	30	20	628	524.270776	620.866694	628	1.14	0.00	6.88
10	100	30	20	732	705.051229	729.547378	732	0.34	0.00	9.10
11	100	60	70	612.145738	606.45432	609.833638	622.283	2.03	1.66	40.62
12	100	60	70	686.130259	649.068458	675.208522	686.212	1.60	0.01	29.47
13	100	60	70	734	722.033536	733.174515	734	0.11	0.00	6.90
14	100	60	70	665.868588	657.703283	661.460391	666	0.68	0.02	53.99
15	100	60	70	984.588193	818.248599	855.932906	986	13.21	0.14	77.34
16	100	60	20	691	629.620621	664.054558	691	3.90	0.00	43.90
17	100	60	20	666.995818	631.110603	655.171515	667	1.77	0.00	32.95
18	100	60	20	756.780603	725.103749	746.527684	758	1.52	0.16	32.37
19	100	60	20	763	626.529227	722.010148	763.971	5.50	0.13	30.04
20	100	60	20	532.218697	521.894551	528.196096	533	0.90	0.15	38.96

Table 5 continued

#	m	Rank	Dense	Optimal value	LP relaxation	Preprocessed bounds		Relative gaps (percentages)		
						Lower	Upper	UB-LB	UB-opt	%Closed
21	150	30	70	1029	946.929565	1010.002422	1029	1.85	0.00	23.15
22	150	30	70	1160	1075.719667	1143.215912	1160	1.45	0.00	19.91
23	150	30	70	965	929.722695	957.060812	965	0.82	0.00	22.51
24	150	30	70	1242	1170.744571	1232.634488	1242	0.75	0.00	13.14
25	150	30	70	1149	1013.045865	1063.947928	1149	7.40	0.00	62.56
26	150	30	20	822.333333	790.161133	813.932095	822.333	1.02	0.00	26.11
27	150	30	20	1046	991.351886	1039.478766	1046	0.62	0.00	11.93
28	150	30	20	922	851.085225	899.489258	922	2.44	0.00	31.74
29	150	30	20	992	855.028214	921.051941	992	7.15	0.00	51.80
30	150	30	20	848	729.617101	775.254605	848	8.58	0.00	61.45
31	150	100	70	1377.072388	1263.798462	1344.135656	1377.072	2.39	0.00	29.08
32	150	100	70	837	833.238632	835.993215	837	0.12	0.00	26.77
33	150	100	70	972.779519	912.297933	951.089989	972.804	2.23	0.00	35.86
34	150	100	70	1260.57242	1206.833191	1238.300018	1261.188	1.82	0.05	41.45
35	150	100	70	1087.08492	1040.170448	1077.111477	1089	1.09	0.18	21.26
36	150	100	20	921.273479	893.518557	904.290053	923	2.03	0.19	61.19
37	150	100	20	923.772654	774.71571	879.664636	925	4.91	0.13	29.59
38	150	100	20	1139	1111.79451	1126.884941	1139	1.06	0.00	44.53
39	150	100	20	879.582356	812.660589	852.096526	879.605	3.13	0.00	41.07
40	150	100	20	1158.383138	1063.017814	1119.548217	1158.432	3.36	0.00	40.72

Table 5 continued

#	m	Rank	Dense	Optimal value	LP relaxation	Preprocessed bounds		Relative gaps (percentages)		
						Lower	Upper	UB-LB	UB-opt	%Closed
41	200	30	70	1580	1098.044624	1196.5995	1580	24.27	0.00	79.55
42	200	30	70	1057	1025.39776	1050.433637	1057	0.62	0.00	20.78
43	200	30	70	1577	1467.609941	1541.862973	1577	2.23	0.00	32.12
44	200	30	70	1535	1462.36974	1524.019988	1535	0.72	0.00	15.12
45	200	30	70	1153	1122.856763	1145.503839	1153	0.65	0.00	24.87
46	200	30	20	1229	1148.301545	1192.605532	1229	2.96	0.00	45.10
47	200	30	20	1350	1251.324462	1318.973919	1350	2.30	0.00	31.44
48	200	30	20	1451	1115.387691	1208.887517	1451	16.69	0.00	72.14
49	200	30	20	1345	1261.123305	1337.276135	1345	0.57	0.00	9.21
50	200	30	20	1249	1164.340236	1195.763472	1249	4.26	0.00	62.88
51	200	120	70	1726.526853	1649.267937	1701.696186	1728	1.52	0.09	32.14
52	200	120	70	1403	1337.168109	1394.142467	1403	0.63	0.00	13.45
53	200	120	70	1144.989488	1126.310832	1143.367197	1145	0.14	0.00	8.69
54	200	120	70	1542	1500.576683	1532.123758	1542	0.64	0.00	23.84
55	200	120	70	1096.255705	951.763018	1009.459289	1097	7.99	0.07	60.07
56	200	120	20	1235.593203	1183.04243	1203.799741	1237	2.69	0.11	60.50
57	200	120	20	1224.764683	1100.94521	1188.734874	1226	3.04	0.10	29.10
58	200	120	20	1145.969792	1132.996319	1140.093917	1147	0.60	0.09	45.29
59	200	120	20	1426	1399.225251	1415.85159	1429.364	0.95	0.24	37.90
60	200	120	20	1371.901959	1340.784415	1358.035244	1372	1.02	0.01	44.56
Means								3.28	0.07	35.40

Table 6 Performance data for the 60 instances. The final gap obtained by default CPLEX is indicated for the 18 instances it didn't solve (DNF) within the time limit of 3600 s

#	m	Rank	Dense	Preprocess time	Our algorithm		Default CPLEX indicator constraint		
					Time	Nodes	Time	Nodes	%Gap
1	100	30	70	5.93	1.19	404	7.04	2704	
2	100	30	70	4.71	5.70	4340	DNF	6572052	0.485
3	100	30	70	9.35	0.61	260	4.98	1775	
4	100	30	70	1.79	0.13	38	1.70	185	
5	100	30	70	4.80	0.46	210	4.18	1330	
6	100	30	20	1.02	0.02	18	0.73	43	
7	100	30	20	4.52	0.24	44	0.81	103	
8	100	30	20	5.55	0.55	224	5.24	3305	
9	100	30	20	3.56	0.08	32	1.04	204	
10	100	30	20	0.81	0.10	30	1.16	208	
11	100	60	70	1.30	0.24	88	7.08	3940	
12	100	60	70	6.38	0.73	588	384.44	353435	
13	100	60	70	1.21	0.10	20	1.09	49	
14	100	60	70	1.21	1.06	534	9.32	4671	
15	100	60	70	7.25	3.69	1322	DNF	9693517	0.125
16	100	60	20	5.29	1.01	444	2.91	907	
17	100	60	20	6.02	2.06	1324	7.47	5723	
18	100	60	20	1.21	0.35	206	5.55	2672	
19	100	60	20	4.76	0.37	194	3.73	1011	
20	100	60	20	1.00	0.37	226	2.83	708	
Means				3.88	0.95	527			
21	150	30	70	24.66	2.40	448	15.58	3307	
22	150	30	70	23.08	0.96	124	4.27	0	
23	150	30	70	5.11	0.16	56	4.58	211	
24	150	30	70	24.49	0.61	98	8.52	2163	
25	150	30	70	24.69	6.17	942	685.24	207429	
26	150	30	20	3.68	0.20	92	3.27	232	
27	150	30	20	16.81	0.20	32	2.76	124	
28	150	30	20	18.82	0.36	78	3.62	370	
29	150	30	20	14.23	1.60	188	7.03	1256	
30	150	30	20	18.49	3.70	682	22.30	10877	
31	150	100	70	26.75	101.60	28538	DNF	13175000	0.146
32	150	100	70	4.86	0.54	192	6.31	580	
33	150	100	70	27.11	29.50	8744	DNF	2323005	0.114
34	150	100	70	14.16	132.03	32124	DNF	1674151	0.206
35	150	100	70	5.21	10.90	2888	DNF	2443005	0.272
36	150	100	20	4.42	16.12	4602	DNF	15955452	0.437
37	150	100	20	22.16	15.45	3064	2338.14	845218	

Table 6 continued

#	m	Rank	Dense	Preprocess time	Our algorithm		Default CPLEX indicator constraint		
					Time	Nodes	Time	Nodes	%Gap
38	150	100	20	4.84	2.32	584	11.66	1427	
39	150	100	20	22.38	4.84	976	19.74	4403	
40	150	100	20	23.75	32.48	10376	2063.57	1202357	
Means				15.49	16.53	4249			
41	200	30	70	126.63	1546.32	181008	DNF	1368269	0.612
42	200	30	70	12.97	0.22	38	5.50	0	
43	200	30	70	76.18	1.19	66	8.43	189	
44	200	30	70	15.66	1.48	262	29.50	3328	
45	200	30	70	14.40	0.33	64	5.77	0	
46	200	30	20	44.60	0.85	42	5.51	0	
47	200	30	20	49.62	1.17	120	15.08	1714	
48	200	30	20	58.34	22.07	930	1538.63	405689	
49	200	30	20	39.58	0.69	48	5.75	0	
50	200	30	20	48.30	2.80	196	19.23	2749	
51	200	120	70	13.90	176.16	27046	DNF	748850	0.292
52	200	120	70	14.87	319.53	53786	DNF	635814	0.048
53	200	120	70	15.17	25.74	5014	DNF	1721873	0.036
54	200	120	70	13.97	47.07	7736	DNF	1350106	0.028
55	200	120	70	87.88	86.08	8646	DNF	4416595	0.168
56	200	120	20	14.61	283.36	42010	DNF	1371796	0.248
57	200	120	20	82.14	1017.50	97688	DNF	651199	0.736
58	200	120	20	12.90	11.19	1834	DNF	1233196	0.160
59	200	120	20	14.43	31.69	5188	DNF	856913	0.101
60	200	120	20	13.87	491.75	85978	DNF	710977	0.278
Means				58.50	203.36	25885			

In order to test the effectiveness of different type of valid constraints, a series of LPCC instances was randomly generated, and Procedure 4 gives a detailed description of the generator.

Remark 1 In the initialization step of the procedure, n is the dimension of x variable; m is the dimension of y variable; k is the dimension of b; $rankM$ is the rank of matrix M; $dense$ is the density of generated matrices; we assume all instances have the non-negativity constraint $x \geq 0$ which are not included in the constraint $Ax + By \geq b$; *step 1* and *step 2* are used to generate a feasible LPCC solution; step 8 is to generate matrix M to be a non-symmetric positive semidefinite matrix with rank $rankM$.

We generated 60 LPCC instances with 100, 150, 200 complementaries, 20 instances of each size, and with the same parameter, we randomly generated 5 instances. For CPLEX solving LPCC instances, we used indicator constraints in CPLEX C callable

input : n,m,k,$rankM$,$dense$
output: vector c,d,b,q; matrix A,B,N,M

1: generate n dimension vector $\bar{x}$ with value between 0 and 10, integer;
2: generate m dimension vector $\bar{y}$ with value between 0 and 10, integer if index $< \frac{m}{3}$; 0 otherwise;
3: generate n dimension vector c with value between 0 and 10, integer;
4: generate m dimension vector d with value between 0 and 10, integer;
5: generate $k \times n$ matrix A with value between -5 and 6, integer, and the matrix density is $dense$;
6: generate $k \times m$ matrix B with value between -5 and 6, integer, and the matrix density is $dense$;
7: generate $m \times n$ matrix N with value between -5 and 6, integer, and the matrix density is $dense$;
8: generate $m \times rankM$ matrix L with value between -5 and 6, integer, and the matrix density is $dense$; generate $m \times m$ upper triangular matrix ΔM with value between -2 and 2, integer; Let $m \times m$ matrix $M = LL^T + \Delta M - \Delta M^T$;
9: generate k dimension vector Δb with value between 1 and 11, integer; let k dimension vector $b = A\bar{x} + B\bar{y} - \Delta b$;
10: generate m dimension vector Δq with value 0 if index $< \frac{2m}{3}$; integer between 1 and 11 otherwise; let m dimension vector $q = -N\bar{x} - M\bar{y} + \Delta q$;

Procedure 4: LPCC instances generator

library [35] to formulate the complementarity constraints, and the CPLEX setting is default. The time limit for CPLEX is 3600 s. Notice that default CPLEX is unable to solve most of our LPCC instances when $m = 200$ within 3600 s.

Table 5 contains objective function value information for the 60 instances, including the effectiveness of the preprocessing routines. Table 6 contains performance data.

B Bilevel test instances

See Tables 7, 8, 9, and 10.

Our code solved all 63 of the instances with dimension of v equal to 50 and 18/35 of the larger instances. With extended time, default CPLEX was able to solve all but one problem with $n = 50$; it still has a gap of 16.56% for problem 60 after more than 7200 s of wall clock time and 47304 s of processor time. It solved just 6/35 of the larger instances. Run time information can be found in Tables 9 and 10.

Table 7 Values of bilevel instances with dimension of v equal to 50

#	Dimensions			Rank(Q)	LP relaxation	Preprocess lower bound	Optimal value	%Gap shrunk
	v	b	g					
1	50	25	50	25	0.644247	0.67043	0.708016	41.06
2	50	25	50	25	0.691488	0.742312	0.817536	40.32
3	50	25	100	25	0.758156	0.853512	0.90403	65.37
4	50	25	100	25	0.508003	0.719332	1.030281	40.46
5	50	25	150	25	0.634865	0.768284	0.930512	45.13
6	50	25	150	25	0.746323	0.939574	1.179523	44.61
7	50	25	200	25	0.834849	0.882683	0.983496	32.18
8	50	25	200	25	0.552456	0.742789	1.113869	33.90
9	50	50	50	25	0.74926	0.833068	0.974266	37.25
10	50	50	50	25	0.596532	0.680096	0.777676	46.13
11	50	50	100	25	0.74352	0.85814	1.025396	40.66
12	50	50	100	25	0.713623	0.897497	1.007106	62.65
13	50	50	150	25	0.734739	0.788571	0.884411	35.97
14	50	50	150	25	0.521193	0.682435	0.990508	34.36
15	50	50	200	25	0.754768	0.782534	0.783561	96.43
16	50	50	200	25	0.754747	0.847708	1.054281	31.04
17	50	75	50	25	0.649778	0.773429	0.945024	41.88
18	50	75	50	25	0.607476	0.880724	1.046959	62.17
19	50	75	100	25	0.720769	0.812158	0.971363	36.47
20	50	75	100	25	0.529814	0.667117	0.949297	32.73
21	50	75	150	25	0.909594	0.93072	0.933821	87.20
22	50	75	150	25	0.710307	0.836857	1.103089	32.22
23	50	75	200	25	0.718915	0.979733	1.326493	42.93
24	50	75	200	25	0.794803	0.916565	0.950861	78.02
25	50	100	50	25	0.766494	0.894661	1.086423	40.06
26	50	100	50	25	0.485909	0.627154	0.962494	29.64
27	50	100	100	25	0.767284	0.838957	0.95196	38.81
28	50	100	150	25	0.578038	0.699594	0.793066	56.53
29	50	100	150	25	0.713984	0.743964	0.760946	63.84
30	50	100	200	25	0.616827	0.867557	1.064487	56.01
31	50	100	200	25	0.651844	0.751472	0.793559	70.30
32	50	25	50	50	0.644746	0.770222	0.897356	49.67
33	50	25	50	50	0.602	0.759655	0.928742	48.25
34	50	25	100	50	0.660691	0.816037	1.03487	41.52
35	50	25	100	50	0.578159	0.804343	1.031041	49.94
36	50	25	150	50	0.69423	0.856205	1.065649	43.61
37	50	25	150	50	0.790053	0.903077	1.01511	50.22
38	50	25	200	50	0.619887	0.766884	0.977958	41.05
39	50	25	200	50	0.661197	0.876876	1.104905	48.61

Table 7 continued

#	Dimensions			Rank(Q)	LP relaxation	Preprocess lower bound	Optimal value	%Gap shrunk
	v	b	g					
40	50	50	50	50	0.57275	0.793187	1.07134	44.21
41	50	50	50	50	0.54549	0.696252	0.886063	44.27
42	50	50	100	50	0.656456	0.837083	1.073918	43.27
43	50	50	100	50	0.75742	0.844813	0.963771	42.35
44	50	50	150	50	0.67609	0.907669	1.180604	45.90
45	50	50	150	50	0.671718	0.897607	1.117394	50.68
46	50	50	200	50	0.664089	0.816423	1.043968	40.10
47	50	50	200	50	0.571399	0.856775	1.074	56.78
48	50	75	50	50	0.514044	0.699661	0.899591	48.14
49	50	75	50	50	0.647895	0.741642	0.978757	28.33
50	50	75	100	50	0.623007	0.861603	1.058687	54.76
51	50	75	100	50	0.607434	0.803321	0.940491	58.81
52	50	75	150	50	0.742589	1.022707	1.157709	67.48
53	50	75	150	50	0.706354	0.844787	0.980656	50.47
54	50	75	200	50	0.719345	0.877183	1.064428	45.74
55	50	75	200	50	0.690228	0.845069	0.968196	55.70
56	50	100	50	50	0.642142	0.838403	1.021128	51.79
57	50	100	50	50	0.530841	0.864219	1.13489	55.19
58	50	100	100	50	0.647984	0.832686	1.119443	39.18
59	50	100	100	50	0.70828	0.925214	1.182871	45.71
60	50	100	150	50	0.544611	0.755445	1.086004	38.94
61	50	100	150	50	0.71695	0.891654	0.996318	62.54
62	50	100	200	50	0.48787	0.771204	1.251992	37.08
63	50	100	200	50	0.64742	0.812777	0.970732	51.14

Table 8 Values of bilevel instances with larger dimensions of v. The final gaps obtained by each code are indicated for the instances it did not solve

#	Dimensions			Rank(Q)	LP relaxation	Preprocess lower bound	Optimal value	% Gap shrunk	Final % gaps	
	v	b	g						CPLEX	Our code
64	75	25	50	50	0.70269	0.796622	1.015407	30.04	5.42	
65	75	25	50	50	0.523934	0.662829	0.925725	34.57	Solved	
66	75	25	100	50	0.590782	0.727175	1.04984	29.71	17.98	
67	75	25	100	50	0.619884	0.743261	1.056703	28.24	18.16	
68	75	50	50	50	0.561663	0.78647	1.011143	50.01	Solved	
69	75	50	50	50	0.560373	0.695674	0.980867	32.18	19.20	
70	75	50	100	50	0.801978	0.889381	1.041901	36.43	Solved	
71	75	50	100	50	0.754207	0.881563	0.959749	61.96	Solved	
72	75	75	50	50	0.700723	0.815533	1.026476	35.24	7.71	
73	75	75	50	50	0.601257	0.699462	0.930676	29.81	6.95	
74	75	75	100	50	0.346691	0.519392	0.86743	33.16	29.05	
75	75	75	100	50	0.594098	0.738111	1.094845	28.76	25.26	
76	100	25	50	50	0.882787	0.94306	0.992266	55.05	Solved	
77	100	25	50	50	0.567423	0.613049	0.732117	27.70	3.08	
78	100	25	75	50	0.603912	0.639672	0.842811	14.97	20.18	
79	75	25	50	75	0.475835	0.699877			No UB	5.19
80	75	25	50	75	0.524895	0.732925	0.98902	44.82	7.86	
81	75	25	100	75	0.603074	0.783103			16.26	12.05
82	75	25	100	75	0.512566	0.680474			16.44	12.09
83	75	50	50	75	0.546256	0.697274	1.092166	27.66	25.04	
84	75	50	50	75	0.519369	0.68752			18.61	10.53
85	75	50	100	75	0.508331	0.6817			22.22	5.29
86	75	50	100	75	0.619981	0.80781			16.37	4.28
87	75	75	50	75	0.485787	0.630083			27.69	13.95
88	75	75	50	75	0.553843	0.737812			7.31	7.04
89	75	75	100	75	0.416464	0.615691			18.33	13.76
90	75	75	100	75	0.63856	0.783523	1.02771	37.25	Solved	

Table 9 Performance on bilevel instances with dimension of v equal to 50

#	Dimensions			Rank(Q)	Preprocess time	Our code		Default CPLEX	
	v	b	g			Time	Nodes	Time	Nodes
1	50	25	50	25	4.03	0.53	152	3.52	1395
2	50	25	50	25	2.51	3.66	834	28.79	8785
3	50	25	100	25	5.50	1.13	154	36.29	21962
4	50	25	100	25	7.06	11.90	2706	364.99	512865
5	50	25	150	25	10.10	4.31	974	39.37	20765
6	50	25	150	25	14.56	16.56	4622	1072.49	966602
7	50	25	200	25	19.37	9.09	2284	706.38	463141
8	50	25	200	25	25.27	116.34	16582	444.74	245964
Means					11.05	20.44	3538	337.07	280184
9	50	50	50	25	2.86	1.75	308	5.17	1897
10	50	50	50	25	3.89	2.68	872	43.18	22949
11	50	50	100	25	5.86	5.53	1118	22.96	12752
12	50	50	100	25	7.83	1.61	222	39.19	26333
13	50	50	150	25	11.11	6.04	1356	84.34	38741
14	50	50	150	25	13.93	16.21	3194	216.39	136792
15	50	50	200	25	7.85	0.01	2	0.46	0
16	50	50	200	25	19.89	38.52	8810	1966.71	1297588
Means					9.15	9.04	1985	297.30	192131
17	50	75	50	25	3.68	3.07	652	8.96	7615
18	50	75	50	25	3.11	3.07	672	78.15	64029
19	50	75	100	25	8.48	9.26	2748	217.66	192307
20	50	75	100	25	8.81	12.40	3996	1035.67	833881
21	50	75	150	25	5.71	0.04	14	0.43	0
22	50	75	150	25	16.55	21.92	3628	3143.11	2896080
23	50	75	200	25	21.39	17.50	2552	2861.97	1606015
24	50	75	200	25	21.42	6.18	692	26.36	6211
Means					11.14	9.18	1869	921.54	700767
25	50	100	50	25	4.52	8.85	1636	59.19	46175
26	50	100	50	25	5.36	9.94	2614	234.31	224258
27	50	100	100	25	5.81	5.98	716	49.56	31727
28	50	100	150	25	13.10	27.25	4164	148.25	45614
29	50	100	150	25	8.00	0.97	42	9.44	2528
30	50	100	200	25	23.57	48.67	10332	3046.81	1566417
31	50	100	200	25	13.62	8.13	824	299.12	135438
Means					10.57	15.68	2904	549.53	293165
32	50	25	50	50	2.92	5.96	1556	64.01	36311
33	50	25	50	50	2.70	7.72	2560	70.79	55449
34	50	25	100	50	7.66	46.89	12238	112.51	56827
35	50	25	100	50	8.15	29.99	5240	840.10	686855

Table 9 continued

#	Dimensions			Rank(Q)	Preprocess time	Our code		Default CPLEX	
	v	b	g			Time	Nodes	Time	Nodes
36	50	25	150	50	14.20	34.38	7064	≥ 3600	2042811
37	50	25	150	50	9.58	20.46	5170	255.14	74572
38	50	25	200	50	17.34	161.83	34378	≥ 3600	2001153
39	50	25	200	50	18.56	168.98	39694	≥ 3600	3371704
Means					10.14	59.53	13488		
40	50	50	50	50	3.41	51.01	20124	283.70	277842
41	50	50	50	50	3.61	9.07	2420	93.89	79268
42	50	50	100	50	7.53	29.02	7926	≥ 3600	4014451
43	50	50	100	50	7.43	13.78	4422	208.67	71494
44	50	50	150	50	13.48	96.14	14774	361.15	223345
45	50	50	150	50	15.29	153.99	21876	581.26	360898
46	50	50	200	50	17.46	248.31	48162	1008.91	465121
47	50	50	200	50	25.74	143.38	26886	1165.71	648962
Means					11.74	93.09	18324		
48	50	75	50	50	3.48	9.82	3334	43.88	23923
49	50	75	50	50	3.67	22.58	6612	131.80	87870
50	50	75	100	50	9.70	14.07	2166	28.00	8436
51	50	75	100	50	7.12	4.84	662	101.97	46016
52	50	75	150	50	13.43	2.39	232	3.83	926
53	50	75	150	50	12.51	32.03	5386	178.99	118337
54	50	75	200	50	21.96	56.51	5864	1881.67	904828
55	50	75	200	50	21.35	26.14	4890	194.52	64333
Means					11.65	21.05	3643		
56	50	100	50	50	4.87	13.45	3284	272.87	190537
57	50	100	50	50	5.85	51.07	19014	≥ 3600	30944839
58	50	100	100	50	8.08	120.59	33178	2334.10	1597429
59	50	100	100	50	9.86	147.33	28058	2008.30	1283510
60	50	100	150	50	16.61	218.98	48542	≥ 3600	3310389
61	50	100	150	50	11.40	18.82	2662	46.03	13536
62	50	100	200	50	21.63	3122.24	323810	≥ 3600	13525363
63	50	100	200	50	22.05	91.29	9660	353.94	122675
Means					12.54	472.97	58526		

Table 10 Performance on bilevel instances with larger dimensions of v

#	Dimensions			Rank(Q)	Preprocess time	Our code		Default CPLEX	
	v	b	g			Time	Nodes	Time	Nodes
64	75	25	50	50	11.47	290.70	75616	$\geq$ 3600	12455384
65	75	25	50	50	7.73	299.37	68770	$\geq$ 3600	17447435
66	75	25	100	50	20.20	396.67	86638	$\geq$ 3600	14159416
67	75	25	100	50	16.82	934.03	159358	$\geq$ 3600	7263261
Means					14.06	480.19	97596		
68	75	50	50	50	12.75	89.09	19502	$\geq$ 3600	31475242
69	75	50	50	50	10.46	139.84	28866	$\geq$ 3600	1747600
70	75	50	100	50	13.46	51.33	7478	$\geq$ 3600	4579441
71	75	50	100	50	17.91	5.75	362	3585.51	1415201
Means					13.65	71.50	14052		
72	75	75	50	50	9.32	94.15	16830	$\geq$ 3600	4006474
73	75	75	50	50	11.50	86.90	15076	$\geq$ 3600	3636937
74	75	75	100	50	21.87	2519.64	267744	$\geq$ 3600	1185965
75	75	75	100	50	24.30	1569.69	197996	$\geq$ 3600	1144374
Means					16.75	1067.60	124412		
76	100	25	50	50	8.99	6.78	408	155.29	27742
77	100	25	50	50	8.18	79.48	5450	$\geq$ 3600	2647874
78	100	25	75	50	26.23	1030.75	100044	$\geq$ 3600	1049577
Means					14.47	372.34	35301		
79	75	25	50	75	8.50	$\geq$ 3600	542572	$\geq$ 3600	3248691
80	75	25	50	75	11.67	1197.30	212828	$\geq$ 3600	2730181
81	75	25	100	75	16.93	$\geq$ 3600	449218	$\geq$ 3600	13374465
82	75	25	100	75	22.19	$\geq$ 3600	268486	$\geq$ 3600	13553753
83	75	50	50	75	10.94	3587.62	445828	$\geq$ 3600	2126574
84	75	50	50	75	12.60	$\geq$ 3600	488384	$\geq$ 3600	1758366
85	75	50	100	75	22.05	$\geq$ 3600	408186	$\geq$ 3600	1269948
86	75	50	100	75	19.24	$\geq$ 3600	429360	$\geq$ 3600	1389433
87	75	75	50	75	12.25	$\geq$ 3600	442694	$\geq$ 3600	1984975
88	75	75	50	75	11.15	$\geq$ 3600	463366	$\geq$ 3600	2227245
89	75	75	100	75	24.20	$\geq$ 3600	276876	$\geq$ 3600	1390365
90	75	75	100	75	20.69	1221.35	107572	3537.29	1604647

C Inverse QP instances

See Table 11.

Computational results on 25 inverse QP instances can be found in Table 11. For each set of 5 instances, the average CPU time is listed if all the instances were solved or the number of solved instances is noted.

Table 11 Performance on 25 inverse quadratic programs. Mean solution time is listed for each set of five problems solved successfully by a code; otherwise, the number of solved instances is given. The number of cutting planes added by CPLEX MIP is also reported; GF are Gomory fractional cuts, MIR are mixed integer rounding cuts, L&P are lift-and-project cuts, and IB are implicit bound cuts

$\tilde{m}$	$\tilde{n}$	Instance	Time for our code	CPLEX MIP time	CPLEX MIP cuts			
					GF	MIR	L&P	IB
100	75	a	303.58	3504.84	5	3	2	
100	75	b	304.58	2686.84	22	1		
100	75	c	60.38	69.09	11			1
100	75	d	328.71	3600.00	19			
100	75	e	94.91	182.61	11	1		
Mean or success			218.43	4 of 5				
120	90	a	271.24	1314.80	9	7		
120	90	b	250.26	3600.00	14	2		
120	90	c	126.13	209.95	7	5		
120	90	d	215.09	3600.00	16			
120	90	e	1101.49	3864.48	11	8		
Mean or success			392.84	2 of 5				
150	20	a	55.12	702.95	5			
150	20	b	486.00	3263.58	7			
150	20	c	163.54	3319.01	4			
150	20	d	49.13	1260.59	6			
150	20	e	3605.77	3781.06	1			
Success			4 of 5	4 of 5				
200	15	a	154.11	1057.09	7			
200	15	b	81.65	477.30	1			
200	15	c	3604.85	3600.00	4			
200	15	d	128.41	365.59	7		1	
200	15	e	47.71	224.61	5			
Success			4 of 5	4 of 5				
400	5	a	225.54	1573.05	7			
400	5	b	1797.89	3600.00	17			
400	5	c	238.79	183.92	21			
400	5	d	252.45	388.98	20			
400	5	e	47.45	224.18	5			
Mean or success			512.42	4 of 5				

References

1. Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische University Berlin (2007)
2. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Oper. Res. Lett. **33**(1), 42–54 (2005)
3. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton (2006)
4. Audet, C., Haddad, J., Savard, G.: Disjunctive cuts for continuous bilevel programming. Optim. Lett. **1**(3), 259–267 (2006)
5. Audet, C., Hansen, P., Jaumard, B., Savard, G.: A symmetrical linear maxmin approach to disjoint bilinear programming. Math. Program. **85**(3), 573–592 (1999)
6. Audet, C., Savard, G., Zghal, W.: New branch-and-cut algorithm for bilevel linear programming. J. Optim. Theory Appl. **38**(2), 353–370 (2007)
7. Bai, L., Mitchell, J.E., Pang, J.: On convex quadratic programs with linear complementarity constraints. Comput. Optim. Appl. **54**(3), 517–554 (2013)
8. Bai, L., Mitchell, J.E., Pang, J.: On conic QPCCs, conic QCQPs and completely positive programs. Math. Program. **159**(1–2), 109–136 (2016)
9. Balas, E.: Disjunctive programming. Ann. Discrete Math. **5**, 3–51 (1979)
10. Bard, J.F., Moore, J.T.: A branch and bound algorithm for the bilevel programming program. SIAM J. Sci. Stat. Comput. **11**(2), 281–292 (1990)
11. Beale, E.M.L., Tomlin, J.A.: Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In: Lawrence, J. (ed.) Proceedings of the Fifth International Conference on Operational Research, pp. 447–454. Travistock Publications, London (1970)
12. Benichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribiere, G., Vincent, O.: Experiments in mixed-integer linear programming. Math. Program. **1**(1), 76–94 (1970)
13. Burdakov, O., Kanzow, C., Schwartz, A.: Mathematical programs with cardinality constraints: reformulation by complementarity-type conditions and a regularization method. SIAM J. Optim. **26**(1), 397–425 (2016)
14. Cottle, R.W., Pang, J., Stone, R.S.: The Linear Complementarity Problem. Academic Press, Cambridge (1992)
15. Dolan, E., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
16. de Farias Jr., I.R., Johnson, E.L., Nemhauser, G.L.: A generalized assignment problem with special ordered sets: a polyhedral approach. Math. Program. **89**(1), 187–203 (2000)
17. de Farias Jr., I.R., Johnson, E.L., Nemhauser, G.L.: Branch-and-cut for combinatorial optimisation problems without auxiliary binary variables. Knowl. Eng. Rev. **16**(1), 25–39 (2001)
18. de Farias Jr., I.R., Kozyreff, E., Zhao, M.: Branch-and-cut for complementarity-constrained optimization. Math. Program. Comput. **6**(4), 365–403 (2014)
19. Facchinei, F., Pang, J.: Finite-Dimensional Variational Inequalities and Complementarity Problems, vol. I,II. Springer, New York (2003)
20. Fang, H., Leyffer, S., Munson, T.S.: A pivoting algorithm for linear programs with complementarity constraints. Optim. Methods Softw. **27**(1), 89–114 (2012)
21. Feng, M., Mitchell, J.E., Pang, J., Shen, X., Wächter, A.: Complementarity formulations of ℓ_0-norm optimization problems. Pac. J. Optim. **14**(2), 273–305 (2018)
22. Fischer, T., Pfetsch, M.E.: Branch-and-cut for linear programs with overlapping SOS1 constraints. Math. Program. Comput. **10**(1), 33–68 (2018)
23. Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. Math. Program. Comput. **1**(2–3), 201–222 (2009)
24. Fomeni, F.D., Gabriel, S.A., Anjos, M.F.: An RLT approach for solving the binary-constrained mixed linear complementarity problem. Tech. Rep. G-2015-60, GERAD, HEC Montréal, Canada (2015). https://www.gerad.ca/en/papers/G-2015-60
25. Fügenschuh, A., Martin, A.: Computational integer programming and cutting planes. In: Handbooks in Operations Research and Management, vol. 12, chapter. 2, pp. 69–122. Elsevier (2005)
26. Fukushima, M., Tseng, P.: An implementable active-set algorithm for computing a B-stationary point of a mathematical program with linear complementarity constraints. SIAM J. Optim. **12**(3), 724–739 (2002)

27. Giannessi, F., Tomasin, E.: Nonconvex quadratic programs, linear complementarity problems, and integer linear programs. In: Conti, R., Ruberti, A. (eds.) Fifth Conference on Optimization Techniques (Rome 1973), Part I, Lecture Notes in Computer Science, vol. 3, pp. 437–449. Springer, Berlin (1973)
28. Gleixner, A.M., Berthold, T., Müller, B., Weltge, S.: Three enhancements for optimization-based bound tightening. J. Glob. Optim. **67**, 731–757 (2017)
29. Hansen, P., Jaumard, B., Savard, G.: New branch-and-bound rules for linear bilevel programming. SIAM J. Sci. Stat. Comput. **13**(5), 1194–1217 (1992)
30. Hooker, J.N., Osorio, M.A.: Mixed logical-linear progrmaming. Discrete Appl. Math. **96–97**, 395–442 (1999)
31. Hu, J., Mitchell, J.E., Pang, J.: An LPCC approach to nonconvex quadratic programs. Math. Program. **133**(1–2), 243–277 (2012)
32. Hu, J., Mitchell, J.E., Pang, J., Bennett, K.P., Kunapuli, G.: On the global solution of linear programs with linear complementarity constraints. SIAM J. Optim. **19**(1), 445–471 (2008)
33. Hu, J., Mitchell, J.E., Pang, J., Yu, B.: On linear programs with linear complementarity constraints. J. Glob. Optim. **53**(1), 29–51 (2012)
34. Ibaraki, T.: The use of cuts in complementary programming. Oper. Res. **21**, 353–359 (1973)
35. ILOG Inc, Mountain View, California: ILOG CPLEX Callable Library C API 11.0 Reference Manual (2007)
36. Izmailov, A.F., Solodov, M.V.: An active-set Newton method for mathematical programs with complementarity constraints. SIAM J. Optim. **19**(3), 1003–1027 (2009)
37. Jara-Moroni, F., Pang, J.S., Wächter, A.: A study of the difference-of-convex approach for solving linear programs with complementarity constraints. Math. Program. **169**(1), 221–254 (2018)
38. Jeroslow, R.G.: Cutting-planes for complementarity constraints. SIAM J. Control Optim. **16**(1), 56–62 (1978)
39. Júdice, J.J.: Algorithms for linear programming with linear complementarity constraints. TOP **20**(1), 4–25 (2012)
40. Júdice, J.J., Faustino, A.M.: A sequential LCP method for bilevel linear programming. Ann. Oper. Res. **34**, 89–106 (1992)
41. Keha, A.B., de Farias Jr, I.R., Nemhauser, G.L.: A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization. Oper. Res. **54**(5), 847–858 (2006)
42. Kunapuli, G., Pang, J., Bennett, K.P.: Bilevel cross-validation-based model selection. In: Guyon, I., Crawley, G., Dror, G., Saffari, A. (eds.) Hands-On Pattern Recognition: Challenges in Machine Learning, vol. 1, chapter. 15, pp. 345–370. Mikrotone Publishing, Brookline, MA (2011)
43. Lee, Y., Pang, J., Mitchell, J.E.: Global resolution of the support vector machine regression parameters selection problem with LPCC. EURO J. Comput. Optim. **3**(1), 197–261 (2015)
44. Leyffer, S., Lopez-Calva, G., Nocedal, J.: Interior methods for mathematical programs with complementarity constraints. SIAM J. Optim. **17**(1), 52–77 (2006)
45. Linderoth, J.T., Savelsbergh, M.W.P.: A computational study of strategies for mixed integer programming. INFORMS J. Comput. **11**, 173–187 (1999)
46. Luo, Z.Q., Pang, J., Ralph, D.: Mathematical Programs with Equilibrium Constraints. Cambridge University Press, Cambridge (1996)
47. Maranas, C.D., Floudas, C.A.: Global optimization in generalized geometric programming. Comput. Chem. Eng. **21**, 351–569 (1997)
48. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: part I—convex underestimating problems. Math. Program. **10**, 147–175 (1976)
49. Mitchell, J.E., Pang, J., Yu, B.: Obtaining tighter relaxations of mathematical programs with complementarity constraints. In: Terlaky, T., Curtis, F. (eds.) Modeling and Optimization: Theory and Applications, Springer Proceedings in Mathematics and Statistics, vol. 21, chapter. 1, pp. 1–23. Springer, New York (2012)
50. Mitchell, J.E., Pang, J., Yu, B.: Convex quadratic relaxations of nonconvex quadratically constrained quadratic programs. Optim. Methods Softw. **29**(1), 120–136 (2014)
51. Pang, J.: Three modeling paradigms in mathematical programming. Math. Program. **125**(2), 297–323 (2010)
52. Pang, J., Fukushima, M.: Some feasibility issues in mathematical programs with equilibrium constraints. SIAM J. Optim. **8**, 673–681 (1998)
53. Pang, J., Leyffer, S.: On the global minimization of the Value-at-Risk. Optim. Methods Softw. **19**(5), 611–631 (2004)

54. Scholtes, S.: Convergence properties of a regularisation scheme for mathematical programs with complementarity constraints. SIAM J. Optim. **11**(4), 918–936 (2001)
55. Stackelberg, H.V.: The Theory of the Market Economy. Oxford University Press, Oxford (1952)
56. Vandenbussche, D., Nemhauser, G.L.: A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. Math. Program. **102**(3), 559–575 (2005)
57. Watson, L.T., Billups, S.C., Mitchell, J.E., Easterling, D.R.: A globally convergent probability-one homotopy for linear programs with linear complementarity constraints. SIAM J. Optim. **23**(2), 1167–1188 (2013)
58. Zamora, J.M., Grossmann, I.E.: A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. J. Glob. Optim. **14**(3), 217–249 (1999)

Affiliations

Bin Yu[1] · John E. Mitchell[2] · Jong-Shi Pang[3]

Jong-Shi Pang
jongship@usc.edu

1 JD.com, 675 E Middlefield Rd, Mountain View, CA 94043, USA

2 Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

3 Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA 90089, USA