Noname manuscript No.
(will be inserted by the editor)

Solving Linear Programs with Complementarity Constraints using Branch-and-Cut

Bin Yu · John E. Mitchell · Jong-Shi Pang

28 October, 2016

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

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 nonconvex, disjunctive optimization problem. With its linear structure, the LPCC plays the same role in the domain of non-convex programming as a linear program does in the domain of convex programming. In §1.1, we present the mathematical formulation of the general LPCC we use throughout this paper. In §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 global resolution of the LPCC, and we achieve this goal through two steps:

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 Number CMMI-1334327. 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.

B. Yu
BNSF Railway, Fort Worth, TX

J.E. Mitchell
Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180. E-mail: mitchj@rpi.edu `http://www.rpi.edu/~mitchj`

J.S. Pang
Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA 90089, USA. E-mail: jongship@usc.edu

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 [38], and we briefly recap the constraints in §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 §1.3, where we also outline the rest of the paper.

1.1 Statement of the Problem

In this paper, we consider a general formulation of the LPCC in the form suggested by Pang and Fukushima [41]. 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 globally

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

where the $a \perp b$ denotes that the perpendicularity between vector a and b, i.e., $a^T b = 0$. Thus, without the orthogonality condition: $y \perp q+Nx+My$, 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.

If 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. To avoid this drawback, in this paper, we present a branch and cut algorithm which branches on the complementarity constraint directly.

Problem (1) generalizes the standard linear complementarity problem (LCP) [9]: $0 \leq y \perp q+My \geq 0$, so the LPCC is NP-Hard. Moreover, affine variational constraints also lead to the problem (1) [36]. Applications of the LPCC are surveyed in [25]. Among these applications, complementarity constraints mainly play three kinds of role during the modelling process:

1. Modelling KKT optimality condition that must be satisfied by some of the variables. Such applications include hierarchical optimization such as Stackelberg games [44], inverse convex quadratic programs, indefinite quadratic programs [21,23], and cross-validated support vector regression [32,33].
2. Modelling equilibrium constraints. See for example the texts [11,36], the survey article [40], or a recent paper on market equilibrium in electric power markets [18].
3. Modelling certain logical conditions that are required by some practical optimization problems. Such applications include non-convex piecewise linear optimization, quantile minimization [42], and ℓ_0-minimization [8,15].

1.2 Previous Work on Solving LPCCs

In this section, we give a brief review of previous work for 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.

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 the reference [30]. 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 [12,20,28] 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 [43]: 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 [8,15]. An alternative is to put a penalty for violation of the complementarity constraints into the objective, and gradually update the penalty to infinity [34]. A homotopy method has also been proposed [46]. 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 methods based on a branch-and-bound scheme have been proposed for solving an LPCC derived from a bilevel linear program. Bard and Moore [7] proposed a pure branch and bound method for solving bilevel linear program. Hansen et al. [22] 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 [26, 29] study alternative ways to solve an LPCC by using a cutting plane method. Audet et al. [3] 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 [18]. It follows from the results of [5] 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 getting global resolution of the LPCC presume the LPCC has a finite optimal value, and this limitation was not resolved until the paper [24]. 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 [4]. The success of this method 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 focus on the study of 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 [14,16]. De Farias et al. [14] considered problems where all the coefficients are nonnegative and their emphasis is on possible families of cutting planes. Fischer and Pfetsch [16] emphasize cuts and branching techniques that are not available in the LPCC case.

1.3 LPCC using Branch-and-Cut

In this paper, we propose a branch-and-cut algorithm for solving the general LPCC problem (1). Notice that for the general LPCC problem, if the initial LP relaxation of the LPCC is feasible, there will be two cases for the initial LP relaxation: the initial LP relaxation is bounded below and the initial LP relaxation is unbounded. We refer to these two cases as the bounded case and the unbounded case of the LPCC. For the bounded case, our algorithm consists of two phases: a preprocessing phase and a branch-and-bound phase; for the unbounded case only the branch-and-bound phase will be applied. In §2, we first describe the preprocessing phase of our algorithm: in §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 §2.2, the strategy of generating and selecting from various types of cutting planes we studied in [38] is discussed, which could sharpen the LP relaxation and improve the initial lower bound of LPCC. In §3, we present the second phase of our algorithm: branch-and-bound. Various node selection strategies and branching complementarity selection strategies are discussed in §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 §4. In §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 [13], nonconvex quadratic programs [45], nonconvex piecewise linear optimization problems [31], and problems with overlapping SOS1 constraints [14,16]. 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

In this section, we will start to present our branch-and-cut algorithm for solving a general LPCC. We divide our algorithm into two phases: a preprocessing phase presented in this section, and a branch-and-bound phase discussed in §3. In §4, we will describe the general scheme of our algorithm.

When the initial LP relaxation is bounded below, the preprocessing phase will be invoked. The preprocessing phase consists 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 [17].

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:

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

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:

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

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'.

Now we will start to discuss our feasibility recovery process. 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. To resolve this issue, next, we will propose a refined local search feasibility recovery procedure, and the key of this refined procedure is to add constraint $lb_{search} \leq c^T x + d^T y \leq ub_{search}$ to (4) when we compute 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 §5.

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 [38], 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. These cutting planes include: bound cuts, general disjunctive cuts & simple cuts, and linear sub-gradient approximation cuts.

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:

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 ascending 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

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 Simple cuts
– Bound cuts
– Linear cuts derived from second order cone constraints

We use the computational results with these cutting planes in [38] to guide the cut generation process. The details of generation and selection rule are described as follows.

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 [6,3] 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 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 rule of selection for violated complementarity constraints for generating disjunctive cuts is based on the value of $y_i w_i$ in the optimal solution to the LP relaxation: we sort $y_i w_i$ in descending order and 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. Therefore, 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

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

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 problems, 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 [38], 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 [37] to tighten the linearization, and handling the $y^T My$ term appropriately. Details can be found in [39]. 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 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 [38].

An additional preprocessing procedure undertaken at each node is the complementary variable fixing process, which is detailed in §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. Unlike the branch-and-bound routine to solve mixed integer programs, here we branch on complementarity constraints instead of branching on integer variables. The branching is imposed on the complementarity constraint directly, and two sub-problems (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 §3.1, we will first present three classic branching rules used to solve mixed integer programs, and then we will present our branching strategy which is based on the idea of these three classic branching rules and also some new proposed ideas designed for the LPCC problem. Node selection is the procedure to select the next sub-problem from the node tree to be processed. In §3.2, we will present and compare different node selection strategies. Besides these two key ingredients, in §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 §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 [35], Fügenschuh and Martin [19] and Achterberg et al [2] for a comprehensive study of branch-and-bound strategies for mixed integer programming. Then we will present our branching strategy based on the idea of these branching strategies. The computational results that compare various branching strategies will be shown in §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 sub-problems (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. 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.

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

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 $\tilde{I} = \{j \in I \,|\, y_j^* \, w_j^* > 0\}$ 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

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*, *Pseudocost Branching* and *Inference Branching*. 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* 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, and select the one that can lead to the largest lower bound improvement. If we choose to test all the complementarities in the candidate set $\tilde{I}$, and solve the resulting LP relaxation to optimality, then, in the literature, we usually call this branching rule as *Full Strong Branching*. In other words, *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.

3.1.2 Pseudocost Branching

Pseudocost Branching 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 $\sigma_i^w$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 [1] 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 the node 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* 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 §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. We will talk about this in more detail when we describe our branching strategy.

3.1.4 Hybrid Branching Strategy for the LPCC (bounded case)

Next, we will first start to describe our branching strategy for the nodes whose LP relaxation is bounded below. 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 §3.1.2:
$$s^{PC} = \sqrt{max\{\Psi_i^y \cdot y_i^*, \epsilon\} \cdot max\{\Psi_i^w \cdot w_i^*, \epsilon\}}$$
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

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

Note that when we combine these four scores into one branching score, it might be reasonable to scale these scores first since they operate on completely different scales. We scale using the 2-norm of the corresponding score vector, and the following formula is the branching score function that we used to evaluate the score for each violated complementarity:

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

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 §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 sub-problem (node) we should pick to process. There are two major criteria for selecting the next sub-problem 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 First Search strategy.

In our implementation of the branch-and-bound routine we use a ***Best First Search*** 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 First Search*, 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

In this subsection, we will briefly describe the node pre-solving procedure and node warm start that we implemented in our algorithm.

We first talk about the node pre-solving procedure. 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.

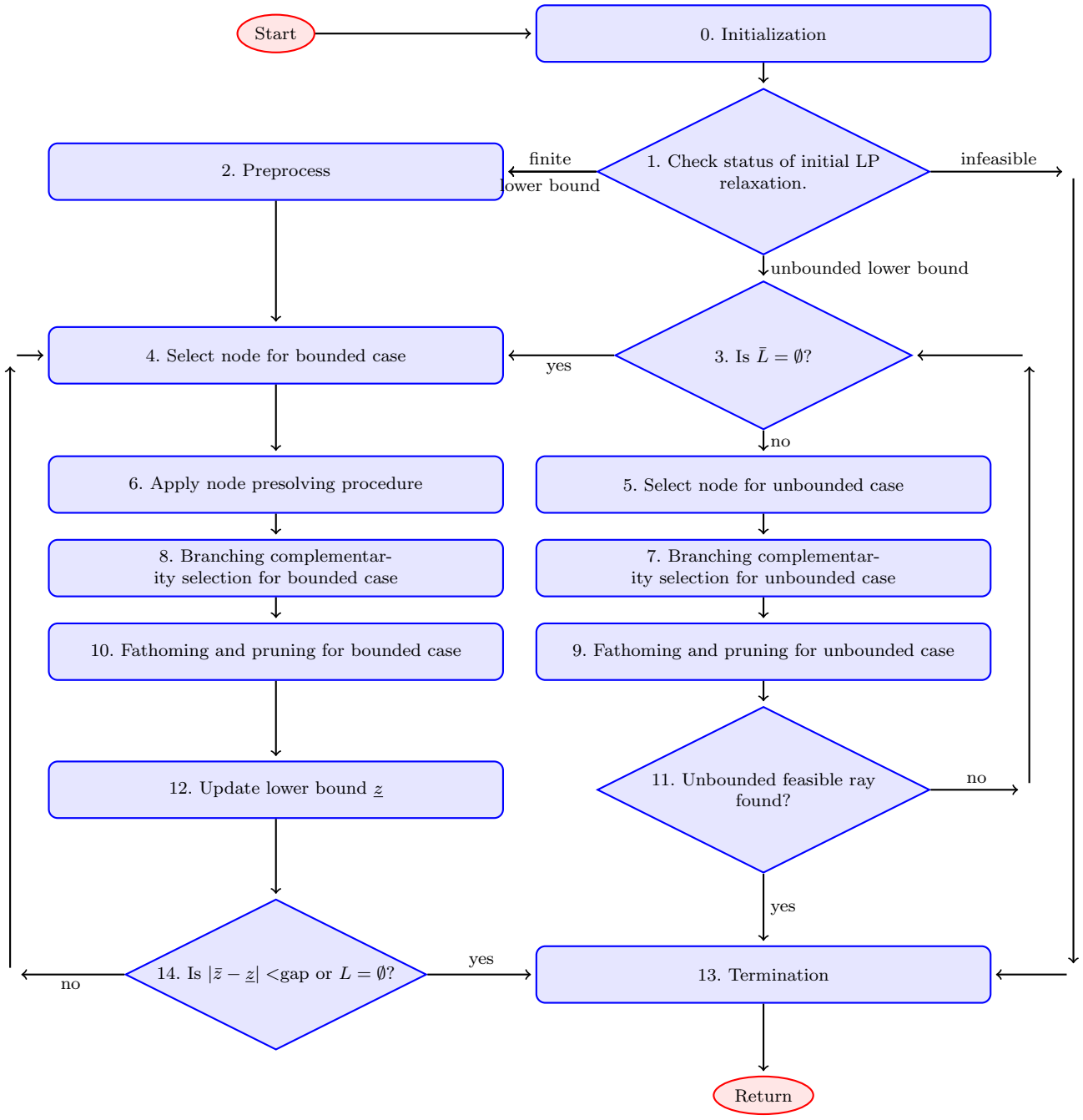

Fig. 1 Flow chart of branch-and-bound procedure

4 General Scheme of the Branch-and-Cut Algorithm for Solving LPCC

Now we are ready to explain the overall flow of our branch-and-cut algorithm. As we mentioned before, we will handle the bounded and unbounded cases of the LPCC in different ways. 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 handles the unbounded nodes with higher priority than the bounded nodes. A flow diagram of the overall algorithm is given in Figure 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 §4.1, with the unbounded case boxes 3, 5, 7, 9, and 11 explained in §4.2. Box 13 is discussed in §4.3.

4.1 Overall Flow of Branch-and-Bound for LPCC (Bounded Case)

In this subsection, we will describe the overall flow of our branch-and-bound routine for handling the bounded case of the LPCC. In fact, this process 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 above. In box 4, we apply *Best First Search* to pick the next node $LPCC^i$ from L to be processed and delete $LPCC^i$ from L. The node presolving procedure from §3.3 is implemented in box 6. Branching complementarity selection in box 8 consists of applying the branching strategy of Section 3.1.4 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:

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 the existing MIP solver 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 2, 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 complementarity selection of box 7 applies the branching strategy of Section 3.1.5 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 Figure 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. We compare the computational performance of our algorithm with that of using unmodified CPLEX 12.6.2 to solve MIP formulations of these LPCC instances, with our preprocessor used for both approaches. We used the CPLEX construct of *indicator constraints* to model the complementarities in the MIP formulations. Except for a few preliminary tests discussed in §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 unmodified CPLEX 12.6.2 MIP formulation can use all 12 available threads. The relative gap for optimality is 10^{-6}, here the relative gap is defined as $\frac{upperbound - lowerbound}{\max(1, |lowerbound|)}$. The tolerance of complementarity is 10^{-6}, i.e., either y_i or w_i for $i = 1, ..., m$ should be less than 10^{-6} for any feasible LPCC solution. All runtimes are reported in seconds.

We used two sets of test problems. The first set consists of 60 LPCC problems 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 sections 5.1 and 5.2. The second set of test problems are LPCC formulations of bilevel programs, where the lower level problem is a convex quadratic program. Information about these instances can be found in Appendix B and discussion of the results can be found in Section 5.3.

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. A generator for the bilevel problems can be found on the website; the generator uses AMPL to construct the instances.

5.1 Computational Results of the Feasibility Recovery Process

In this section, we will discuss the computational results of applying the feasibility recovery procedures to the 60 LPCC instances. 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.

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 various LPCC instances with finite global optimal values.

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 1 Average Computational Results of Feasibility Recovery with $n = 2$, $k = 20$. The column "*Gap*" is calculated as $\frac{LB_{recovered} - LPCC_{opt}}{LPCC_{opt}}$. Detailed results can be found in Table 5 in Appendix A.

m	$Time_{R_1}$ (sec)	$Time_{R_2}$ (sec)	$Time_{R_3}$ (sec)	$Time_{R_4}$ (sec)	$Time_{CPLEX}$ (sec)
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

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

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

We conducted preliminary experiments with 4 different weight settings to choose a score function (13):

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. Based on these results, R_4 is the best branching rule in terms of the number of nodes. Since in terms of solving 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 problems is given in Figure 2. Performance profiles [10] are given in Figure 3. The preprocessing times have been excluded from these plots. All the LPCC instances can be solved by our algorithm within one hour, most of them are solved within half an hour or much less. Each instance requires considerably less processing time with our algorithm than with unmodified CPLEX. Notice that unmodified CPLEX is only able to solve 42 of the 60 instances within 3600 seconds. In particular, it is unable to solve most of our LPCC instances when $m = 200$ within this time limit.

5.3 Bilevel test problems

We also tested our algorithm on bilevel problems of the form

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

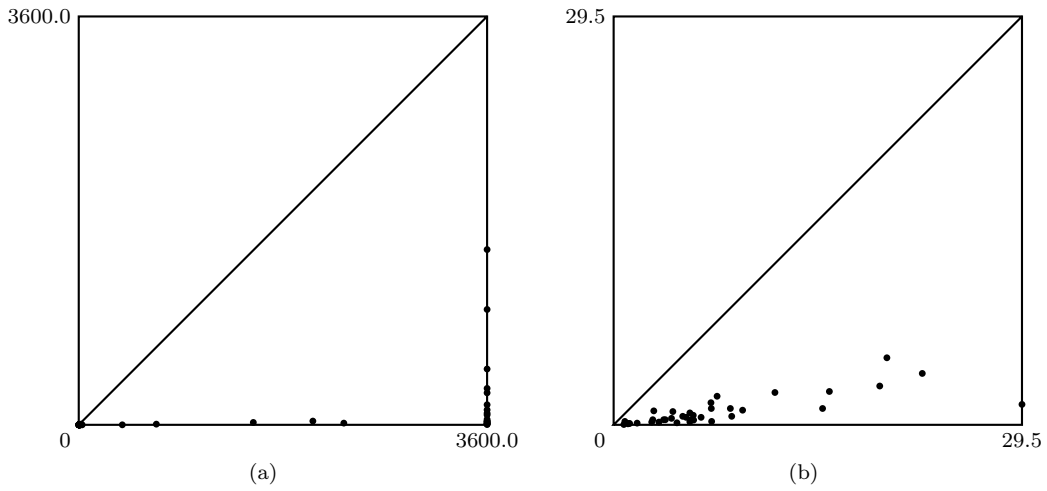

Fig. 2 Scatter plots for CPU time (seconds) for solution of LPCCs. Horizontal axis is time for the unmodified CPLEX MIP solver, vertical axis is time for our branch and cut algorithm. Processing times excluded. (a) All 60 instances. (b) 37 LPCCs where unmodified CPLEX MIP required no more than 30 seconds.

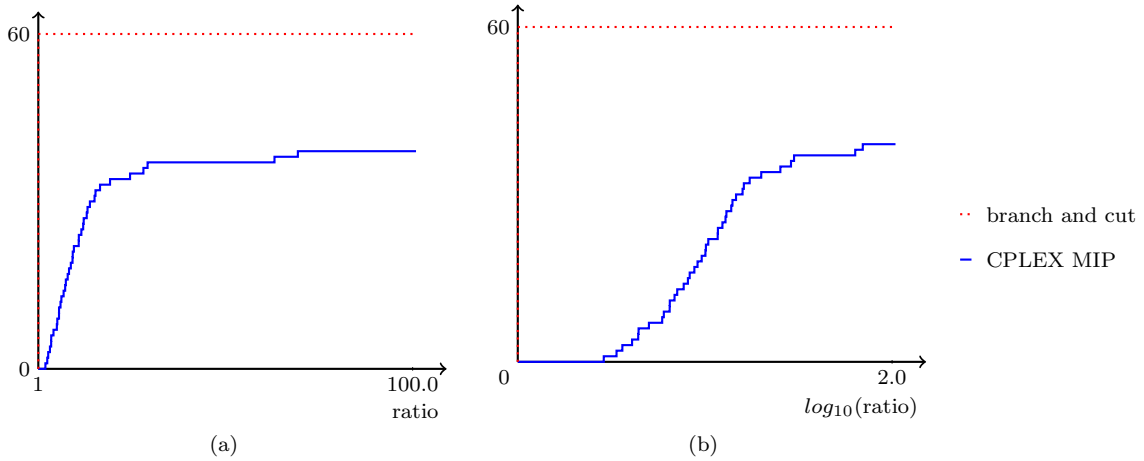

Fig. 3 Performance profile for CPU time (seconds) for solution of 60 LPCCs (preprocessing time excluded). Vertical axis is the number of instances. Horizontal axis is ratio of time required by the given algorithm to the time required by the better algorithm. (a) Linear scale. (b) Log scale.

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:

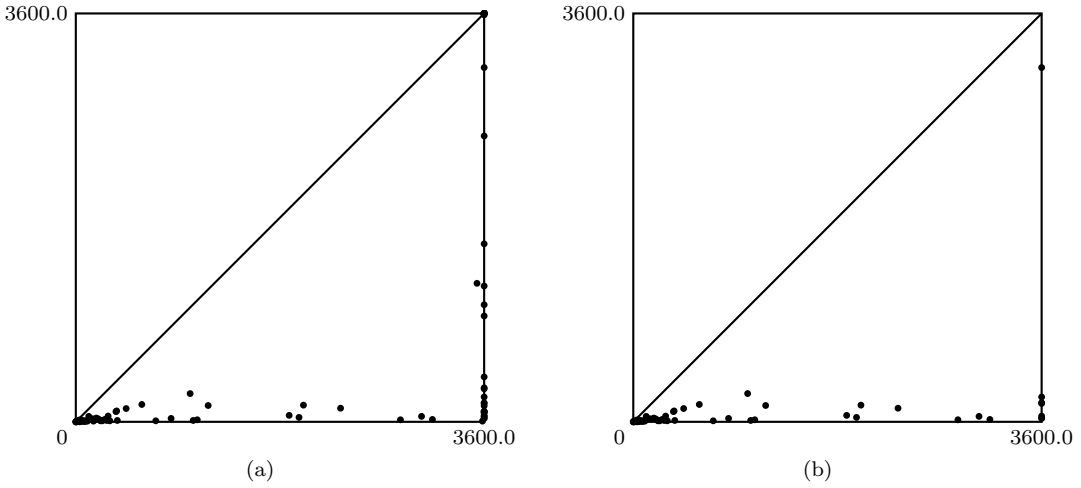

Fig. 4 Scatter plots for CPU time (seconds) for solution of LPCCs based on bilevel problems Horizontal axis is time for unmodified CPLEX MIP solver, vertical axis is time for our branch and cut solver. Processing times excluded. (a) All 98 instances. (b) 63 problems with $n = 50$.

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			

Table 4 Average performance on bilevel problems 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.

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, 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.

We gave each algorithm a time limit of 3600 seconds in addition to the preprocessing time. More details of the characteristics of the instances can be found in Appendix B in Tables 7 and 8, with performance data 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 3/11 of the instances with the dimension of v equal to 100. The corresponding numbers for the unmodified CPLEX MIP code were 56/63, 2/24, and 1/11. Our algorithm was considerably faster than unmodified CPLEX MIP on every instance. Further, it had a smaller final gap than unmodified CPLEX MIP for each instance where neither code could solve the problem. There was no instance that could be solved by unmodified CPLEX MIP which could not be solved by our algorithm. A scatter plot of the CPU time for solving the problems (ignoring the common preprocessing time) is given in Figure 4 and a performance profile is in Figure 5.

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

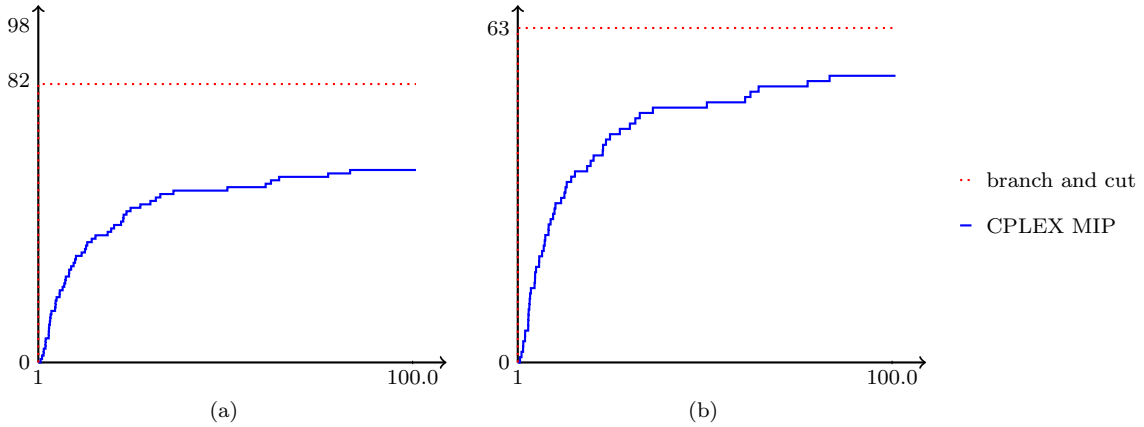

Fig. 5 Performance profile with linear scale for CPU time (seconds) for solution of LPCCs based on bilevel problems (preprocessing time excluded). Vertical axis is the number of instances. Horizontal axis is ratio of time required by the given algorithm to the time required by the better algorithm. (a) All 98 problems. (b) 63 problems with $n = 50$.

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 an unmodified version of CPLEX, at least for our test set of problems. It is able to solve problems with up to 250 complementarity constraints in reasonable amounts of time, and can reliably solve problems with 100 complementarity constraints in less than a minute.

References

1. Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische University Berlin (2007)
2. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters **33**(1), 42–54 (2005)
3. Audet, C., Savard, G., Zghal, W.: New branch-and-cut algorithm for bilevel linear programming. Journal of Optimization Theory and Applications **38**(2), 353–370 (2007)
4. Bai, L., Mitchell, J.E., Pang, J.S.: On convex quadratic programs with linear complementarity constraints. Computational Optimization and Applications **54**(3), 517–554 (2013)
5. Bai, L., Mitchell, J.E., Pang, J.S.: On conic QPCCs, conic QCQPs and completely positive programs. Mathematical Programming **159**(1), 109–136 (2016)
6. Balas, E.: Disjunctive programming. Annals of Discrete Mathematics **5**, 3–51 (1979)
7. Bard, J.F., Moore, J.T.: A branch and bound algorithm for the bilevel programming program. SIAM Journal on Scientific and Statistical Computing **11**(2), 281–292 (1990)
8. Burdakov, O., Kanzow, C., Schwartz, A.: Mathematical programs with cardinality constraints: reformulation by complementarity-type conditions and a regularization method. SIAM Journal on Optimization **26**(1), 397–425 (2016)
9. Cottle, R.W., Pang, J., Stone, R.S.: The Linear Complementarity Problem. Academic Press (1992)
10. Dolan, E., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical Programming **91**, 201–213 (2002)
11. Facchinei, F., Pang, J.: Finite-dimensional variational inequalities and complementarity problems: Volumes I and II. Springer-Verlag, New York (2003)
12. Fang, H., Leyffer, S., Munson, T.S.: A pivoting algorithm for linear programs with complementarity constraints. Optimization Methods and Software **27**(1), 89–114 (2012)
13. Farias Jr., I.R., Johnson, E.L., Nemhauser, G.L.: A generalized assignment problem with special ordered sets: a polyhedral approach. Mathematical Programming **89**(1), 187–203 (2000)
14. Farias Jr., I.R., Kozyreff, E., Zhao, M.: Branch-and-cut for complementarity-constrained optimization. Mathematical Programming Computation **6**(4), 365–403 (2014)
15. Feng, M., Mitchell, J.E., Pang, J., Shen, X., Wächter, A.: Complementarity formulations of l0-norm optimization problems. Tech. rep., Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY (2013)
16. Fischer, T., Pfetsch, M.E.: Branch-and-cut for linear programs with overlapping SOS1 constraints. Tech. rep., TU Darmstadt, Department of Mathematics, Research Group Optimization, Dolivostraße 15, 64293 Darmstadt, Germany (2015)
17. Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. Mathematical Programming Computation **1**(2–3), 201–222 (2009)

18. 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). URL `https://www.gerad.ca/en/papers/G-2015-60`
19. Fügenschuh, A., Martin, A.: Computational integer programming and cutting planes. In: Handbooks in Operations Research and Management, vol. 12, chap. 2, pp. 69–122. Elsevier (2005)
20. Fukushima, M., Tseng, P.: An implementable active-set algorithm for computing a B-stationary point of a mathematical program with linear complementarity constraints. SIAM Journal on Optimization **12**(3), 724–739 (2002)
21. Giannessi, F., Tomasin, E.: Nonconvex quadratic programs, linear complementarity problems, and integer linear programs. In: R. Conti, A. Ruberti (eds.) Fifth Conference on Optimization Techniques (Rome 1973), Part I, *Lecture Notes in Computer Science*, vol. 3, pp. 437–449. Springer, Berlin (1973)
22. Hansen, P., Jaumard, B., Savard, G.: New branch-and-bound rules for linear bilevel programming. SIAM Journal on Scientific and Statistical Computing **13**(5), 1194–1217 (1992)
23. Hu, J., Mitchell, J.E., Pang, J.: An LPCC approach to nonconvex quadratic programs. Mathematical Programming **133**(1–2), 243–277 (2012)
24. Hu, J., Mitchell, J.E., Pang, J., Bennett, K.P., Kunapuli, G.: On the global solution of linear programs with linear complementarity constraints. SIAM Journal on Optimization **19**(1), 445–471 (2008)
25. Hu, J., Mitchell, J.E., Pang, J., Yu, B.: On linear programs with linear complementarity constraints. Journal of Global Optimization **53**(1), 29–51 (2012)
26. Ibaraki, T.: The use of cuts in complementary programming. Operations Research **21**, 353–359 (1973)
27. ILOG Inc, Mountain View, California: ILOG CPLEX Callable Library C API 11.0 Reference Manual (2007)
28. Izmailov, A.F., Solodov, M.V.: An active-set Newton method for mathematical programs with complementarity constraints. SIAM Journal on Optimization **19**(3), 1003–1027 (2009)
29. Jeroslow, R.G.: Cutting-planes for complementarity constraints. SIAM Journal on Control and Optimization **16**(1), 56–62 (1978)
30. Júdice, J.J., Faustino, A.M.: A sequential LCP method for bilevel linear programming. Annals of Operations Research **34**, 89–106 (1992)
31. Keha, A.B., de Farias Jr, I.R., Nemhauser, G.L.: A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization. Operations Research **54**(5), 847–858 (2006)
32. Kunapuli, G., Pang, J., Bennett, K.P.: Bilevel cross-validation-based model selection. In: I. Guyon, G. Crawley, G. Dror, A. Saffari (eds.) Hands-On Pattern Recognition: Challenges in Machine Learning, vol. 1, chap. 15, pp. 345–370. Mikrotone Publishing, Brookline, MA (2011)
33. Lee, Y., Pang, J., Mitchell, J.E.: Global resolution of the support vector machine regression parameters selection problem with LPCC. EURO Journal on Computational Optimization **3**(1), 197–261 (2015)
34. Leyffer, S., Lopez-Calva, G., Nocedal, J.: Interior methods for mathematical programs with complementarity constraints. SIAM Journal on Optimization **17**(1), 52–77 (2006)
35. Linderoth, J.T., Savelsbergh, M.W.P.: A computational study of strategies for mixed integer programming. INFORMS Journal on Computing **11**, 173–187 (1999)
36. Luo, Z.Q., Pang, J., Ralph, D.: Mathematical Programs with Equilibrium Constraints. Cambridge University Press, Cambridge, England (1996)
37. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: part I — convex underestimating problems. Mathematical Programming **10**, 147–175 (1976). See also fortet1960a and fortet1960b
38. Mitchell, J.E., Pang, J., Yu, B.: Obtaining tighter relaxations of mathematical programs with complementarity constraints. In: T. Terlaky, F. Curtis (eds.) Modeling and Optimization: Theory and Applications, *Springer Proceedings in Mathematics and Statistics*, vol. 21, chap. 1, pp. 1–23. Springer, New York (2012)
39. Mitchell, J.E., Pang, J., Yu, B.: Convex quadratic relaxations of nonconvex quadratically constrained quadratic programs. Optimization Methods and Software **29**(1), 120–136 (2014)
40. Pang, J.: Three modeling paradigms in mathematical programming. Mathematical Programming **125**(2), 297–323 (2010)
41. Pang, J., Fukushima, M.: Some feasibility issues in mathematical programs with equilibrium constraints. SIAM Journal on Optimization **8**, 673–681 (1998)
42. Pang, J., Leyffer, S.: On the global minimization of the Value-at-Risk. Optimization Methods and Software **19**(5), 611–631 (2004)
43. Scholtes, S.: Convergence properties of a regularisation scheme for mathematical programs with complementarity constraints. SIAM Journal on Optimization **11**(4), 918–936 (2001)
44. Stackelberg, H.V.: The Theory of the Market Economy. Oxford University Press, Oxford (1952)
45. Vandenbussche, D., Nemhauser, G.L.: A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. Mathematical Programming **102**(3), 559–575 (2005)
46. 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 Journal on Optimization **23**(2), 1167–1188 (2013)

A LPCC Test Instances

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 $rankM$.

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 $m \times m$ skew symmetric matrix ΔM with value between -2 and 3, integer; Let $m \times m$ matrix $M = LL^T + \Delta M$;
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 between 1 and 11, integer; let m dimension vector $q = -N\bar{x} - M\bar{y} + \Delta q$;

Procedure 4: LPCC instances generator

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 library [27] to formulate the complementarity constraints, and the CPLEX setting is default. The time limit for CPLEX is 3600 seconds. Notice that unmodified CPLEX is unable to solve most of our LPCC instances when $m = 200$ within 3600 seconds.

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

B Bilevel test instances

Problem data for the bilevel test instances can be found in Tables 7 and 8. 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.

Our code solved all 63 of the problems with dimension of v equal to 50 and 18/35 of the larger instances. With extended time, unmodified 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 seconds of wall clock time and 47304 seconds of processor time. It solved just 6/35 of the larger instances. Run time information can be found in Tables 9 and 10.

				Optimal	LP	Preprocessed bounds		Relative gaps (percentages)		
#	m	rank	dense	Value	relaxation	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
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
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 5 Objective function data for the 60 problems. 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: (i) the gap between the upper and lower bounds obtained through preprocessing, (ii) the gap between the upper bound obtained from feasibility recovery and the optimal value of the LPCC, and (iii) the improvement in the gap between upper and lower bound effected by the improvement in the LP relaxation obtained through preprocessing.

				Preprocess	Our algorithm		unmodified CPLEX MIP		
#	m	rank	dense	time	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	
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	
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	
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

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

#	Dimensions			rank(Q)	LP relaxation	Preprocess lower bound	Optimal value	% Gap shrunk
	v	b	g					
1	50	25	50	50	0.644746	0.770222	0.897356	49.67
2	50	25	50	50	0.602	0.759655	0.928742	48.25
3	50	25	50	25	0.644247	0.67043	0.708016	41.06
4	50	25	50	25	0.691488	0.742312	0.817536	40.32
5	50	25	100	50	0.660691	0.816037	1.03487	41.52
6	50	25	100	50	0.578159	0.804343	1.031041	49.94
7	50	25	100	25	0.758156	0.853512	0.90403	65.37
8	50	25	100	25	0.508003	0.719332	1.030281	40.46
9	50	25	150	50	0.69423	0.856205	1.065649	43.61
10	50	25	150	50	0.790053	0.903077	1.01511	50.22
11	50	25	150	25	0.634865	0.768284	0.930512	45.13
12	50	25	150	25	0.746323	0.939574	1.179523	44.61
13	50	25	200	50	0.619887	0.766884	0.977958	41.05
14	50	25	200	50	0.661197	0.876876	1.104905	48.61
15	50	25	200	25	0.834849	0.882683	0.983496	32.18
16	50	25	200	25	0.552456	0.742789	1.113869	33.90
17	50	50	50	50	0.57275	0.793187	1.07134	44.21
18	50	50	50	50	0.54549	0.696252	0.886063	44.27
19	50	50	50	25	0.74926	0.833068	0.974266	37.25
20	50	50	50	25	0.596532	0.680096	0.777676	46.13
21	50	50	100	50	0.656456	0.837083	1.073918	43.27
22	50	50	100	50	0.75742	0.844813	0.963771	42.35
23	50	50	100	25	0.74352	0.85814	1.025396	40.66
24	50	50	100	25	0.713623	0.897497	1.007106	62.65
25	50	50	150	50	0.67609	0.907669	1.180604	45.90
26	50	50	150	50	0.671718	0.897607	1.117394	50.68
27	50	50	150	25	0.734739	0.788571	0.884411	35.97
28	50	50	150	25	0.521193	0.682435	0.990508	34.36
29	50	50	200	50	0.664089	0.816423	1.043968	40.10
30	50	50	200	50	0.571399	0.856775	1.074	56.78
31	50	50	200	25	0.754768	0.782534	0.783561	96.43
32	50	50	200	25	0.754747	0.847708	1.054281	31.04
33	50	75	50	50	0.514044	0.699661	0.899591	48.14
34	50	75	50	50	0.647895	0.741642	0.978757	28.33
35	50	75	50	25	0.649778	0.773429	0.945024	41.88
36	50	75	50	25	0.607476	0.880724	1.046959	62.17
37	50	75	100	50	0.623007	0.861603	1.058687	54.76
38	50	75	100	50	0.607434	0.803321	0.940491	58.81
39	50	75	100	25	0.720769	0.812158	0.971363	36.47
40	50	75	100	25	0.529814	0.667117	0.949297	32.73
41	50	75	150	50	0.742589	1.022707	1.157709	67.48
42	50	75	150	50	0.706354	0.844787	0.980656	50.47
43	50	75	150	25	0.909594	0.93072	0.933821	87.20
44	50	75	150	25	0.710307	0.836857	1.103089	32.22
45	50	75	200	50	0.719345	0.877183	1.064428	45.74
46	50	75	200	50	0.690228	0.845069	0.968196	55.70
47	50	75	200	25	0.718915	0.979733	1.326493	42.93
48	50	75	200	25	0.794803	0.916565	0.950861	78.02
49	50	100	50	50	0.642142	0.838403	1.021128	51.79
50	50	100	50	50	0.530841	0.864219	1.13489	55.19
51	50	100	50	25	0.766494	0.894661	1.086423	40.06
52	50	100	50	25	0.485909	0.627154	0.962494	29.64
53	50	100	100	50	0.647984	0.832686	1.119443	39.18
54	50	100	100	50	0.70828	0.925214	1.182871	45.71
55	50	100	100	25	0.767284	0.838957	0.95196	38.81
56	50	100	150	50	0.544611	0.755445	1.086004	38.94
57	50	100	150	50	0.71695	0.891654	0.996318	62.54
58	50	100	150	25	0.578038	0.699594	0.793066	56.53
59	50	100	150	25	0.713984	0.743964	0.760946	63.84
60	50	100	200	50	0.48787	0.771204	1.251992	37.08
61	50	100	200	50	0.64742	0.812777	0.970732	51.14
62	50	100	200	25	0.616827	0.867557	1.064487	56.01
63	50	100	200	25	0.651844	0.751472	0.793559	70.30

Table 7 Values of bilevel problems with dimension of v equal to 50.

#	Dimensions			rank(Q)	LP relaxation	Preprocess lower bound	Optimal value	% Gap shrunk	Final % gaps	
	v	b	g						CPLEX	Our code
64	75	25	50	75	0.475835	0.699877			no UB	5.19
65	75	25	50	75	0.524895	0.732925	0.98902	44.82	7.86	
66	75	25	50	50	0.70269	0.796622	1.015407	30.04	5.42	
67	75	25	50	50	0.523934	0.662829	0.925725	34.57	solved	
68	75	25	100	75	0.603074	0.783103			16.26	12.05
69	75	25	100	75	0.512566	0.680474			16.44	12.09
70	75	25	100	50	0.590782	0.727175	1.04984	29.71	17.98	
71	75	25	100	50	0.619884	0.743261	1.056703	28.24	18.16	
72	75	50	50	75	0.546256	0.697274	1.092166	27.66	25.04	
73	75	50	50	75	0.519369	0.68752			18.61	10.53
74	75	50	50	50	0.561663	0.78647	1.011143	50.01	solved	
75	75	50	50	50	0.560373	0.695674	0.980867	32.18	19.20	
76	75	50	100	75	0.508331	0.6817			22.22	5.29
77	75	50	100	75	0.619981	0.80781			16.37	4.28
78	75	50	100	50	0.801978	0.889381	1.041901	36.43	solved	
79	75	50	100	50	0.754207	0.881563	0.959749	61.96	solved	
80	75	75	50	75	0.485787	0.630083			27.69	13.95
81	75	75	50	75	0.553843	0.737812			7.31	7.04
82	75	75	50	50	0.700723	0.815533	1.026476	35.24	7.71	
83	75	75	50	50	0.601257	0.699462	0.930676	29.81	6.95	
84	75	75	100	75	0.416464	0.615691			18.33	13.76
85	75	75	100	75	0.63856	0.783523	1.02771	37.25	solved	
86	75	75	100	50	0.346691	0.519392	0.86743	33.16	29.05	
87	75	75	100	50	0.594098	0.738111	1.094845	28.76	25.26	
88	100	25	50	100	0.503283	0.649202			18.49	12.30
89	100	25	50	100	0.380048	0.489473			34.49	34.06
90	100	25	50	75	0.603231	0.738381			21.67	15.53
91	100	25	50	75	0.378112	0.542949			32.61	25.20
92	100	25	50	50	0.882787	0.94306	0.992266	55.05	solved	
93	100	25	50	50	0.567423	0.613049	0.732117	27.70	3.08	
94	100	25	75	100	0.563097	0.671054			29.43	26.60
95	100	25	75	100	0.394388	0.530366			38.94	32.75
96	100	25	75	75	0.598314	0.754421			21.18	15.56
97	100	25	75	75	0.448209	0.575138			29.41	19.13
98	100	25	75	50	0.603912	0.639672	0.842811	14.97	20.18	

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

	Dimensions				Preprocess	Our code		unmodified CPLEX	
#	v	b	g	rank(Q)	time	time	nodes	time	nodes
1	50	25	50	50	2.92	5.96	1556	64.01	36311
2	50	25	50	50	2.70	7.72	2560	70.79	55449
3	50	25	50	25	4.03	0.53	152	3.52	1395
4	50	25	50	25	2.51	3.66	834	28.79	8785
5	50	25	100	50	7.66	46.89	12238	112.51	56827
6	50	25	100	50	8.15	29.99	5240	840.10	686855
7	50	25	100	25	5.50	1.13	154	36.29	21962
8	50	25	100	25	7.06	11.90	2706	364.99	512865
9	50	25	150	50	14.20	34.38	7064	$\geq$ 3600	2042811
10	50	25	150	50	9.58	20.46	5170	255.14	74572
11	50	25	150	25	10.10	4.31	974	39.37	20765
12	50	25	150	25	14.56	16.56	4622	1072.49	966602
13	50	25	200	50	17.34	161.83	34378	$\geq$ 3600	2001153
14	50	25	200	50	18.56	168.98	39694	$\geq$ 3600	3371704
15	50	25	200	25	19.37	9.09	2284	706.38	463141
16	50	25	200	25	25.27	116.34	16582	444.74	245964
17	50	50	50	50	3.41	51.01	20124	283.70	277842
18	50	50	50	50	3.61	9.07	2420	93.89	79268
19	50	50	50	25	2.86	1.75	308	5.17	1897
20	50	50	50	25	3.89	2.68	872	43.18	22949
21	50	50	100	50	7.53	29.02	7926	$\geq$ 3600	4014451
22	50	50	100	50	7.43	13.78	4422	208.67	71494
23	50	50	100	25	5.86	5.53	1118	22.96	12752
24	50	50	100	25	7.83	1.61	222	39.19	26333
25	50	50	150	50	13.48	96.14	14774	361.15	223345
26	50	50	150	50	15.29	153.99	21876	581.26	360898
27	50	50	150	25	11.11	6.04	1356	84.34	38741
28	50	50	150	25	13.93	16.21	3194	216.39	136792
29	50	50	200	50	17.46	248.31	48162	1008.91	465121
30	50	50	200	50	25.74	143.38	26886	1165.71	648962
31	50	50	200	25	7.85	0.01	2	0.46	0
32	50	50	200	25	19.89	38.52	8810	1966.71	1297588
33	50	75	50	50	3.48	9.82	3334	43.88	23923
34	50	75	50	50	3.67	22.58	6612	131.80	87870
35	50	75	50	25	3.68	3.07	652	8.96	7615
36	50	75	50	25	3.11	3.07	672	78.15	64029
37	50	75	100	50	9.70	14.07	2166	28.00	8436
38	50	75	100	50	7.12	4.84	662	101.97	46016
39	50	75	100	25	8.48	9.26	2748	217.66	192307
40	50	75	100	25	8.81	12.40	3996	1035.67	833881
41	50	75	150	50	13.43	2.39	232	3.83	926
42	50	75	150	50	12.51	32.03	5386	178.99	118337
43	50	75	150	25	5.71	0.04	14	0.43	0
44	50	75	150	25	16.55	21.92	3628	3143.11	2896080
45	50	75	200	50	21.96	56.51	5864	1881.67	904828
46	50	75	200	50	21.35	26.14	4890	194.52	64333
47	50	75	200	25	21.39	17.50	2552	2861.97	1606015
48	50	75	200	25	21.42	6.18	692	26.36	6211
49	50	100	50	50	4.87	13.45	3284	272.87	190537
50	50	100	50	50	5.85	51.07	19014	$\geq$ 3600	30944839
51	50	100	50	25	4.52	8.85	1636	59.19	46175
52	50	100	50	25	5.36	9.94	2614	234.31	224258
53	50	100	100	50	8.08	120.59	33178	2334.10	1597429
54	50	100	100	50	9.86	147.33	28058	2008.30	1283510
55	50	100	100	25	5.81	5.98	716	49.56	31727
56	50	100	150	50	16.61	218.98	48542	$\geq$ 3600	3310389
57	50	100	150	50	11.40	18.82	2662	46.03	13536
58	50	100	150	25	13.10	27.25	4164	148.25	45614
59	50	100	150	25	8.00	0.97	42	9.44	2528
60	50	100	200	50	21.63	3122.24	323810	$\geq$ 3600	13525363
61	50	100	200	50	22.05	91.29	9660	353.94	122675
62	50	100	200	25	23.57	48.67	10332	3046.81	1566417
63	50	100	200	25	13.62	8.13	824	299.12	135438

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

	Dimensions				Preprocess	Our code		unmodified CPLEX	
#	v	b	g	rank(Q)	time	time	nodes	time	nodes
64	75	25	50	75	8.50	$\geq$ 3600	542572	$\geq$ 3600	3248691
65	75	25	50	75	11.67	1197.30	212828	$\geq$ 3600	2730181
66	75	25	50	50	11.47	290.70	75616	$\geq$ 3600	12455384
67	75	25	50	50	7.73	299.37	68770	$\geq$ 3600	17447435
68	75	25	100	75	16.93	$\geq$ 3600	449218	$\geq$ 3600	13374465
69	75	25	100	75	22.19	$\geq$ 3600	268486	$\geq$ 3600	13553753
70	75	25	100	50	20.20	396.67	86638	$\geq$ 3600	14159416
71	75	25	100	50	16.82	934.03	159358	$\geq$ 3600	7263261
72	75	50	50	75	10.94	3587.62	445828	$\geq$ 3600	2126574
73	75	50	50	75	12.60	$\geq$ 3600	488384	$\geq$ 3600	1758366
74	75	50	50	50	12.75	89.09	19502	$\geq$ 3600	31475242
75	75	50	50	50	10.46	139.84	28866	$\geq$ 3600	1747600
76	75	50	100	75	22.05	$\geq$ 3600	408186	$\geq$ 3600	1269948
77	75	50	100	75	19.24	$\geq$ 3600	429360	$\geq$ 3600	1389433
78	75	50	100	50	13.46	51.33	7478	$\geq$ 3600	4579441
79	75	50	100	50	17.91	5.75	362	3585.51	1415201
80	75	75	50	75	12.25	$\geq$ 3600	442694	$\geq$ 3600	1984975
81	75	75	50	75	11.15	$\geq$ 3600	463366	$\geq$ 3600	2227245
82	75	75	50	50	9.32	94.15	16830	$\geq$ 3600	4006474
83	75	75	50	50	11.50	86.90	15076	$\geq$ 3600	3636937
84	75	75	100	75	24.20	$\geq$ 3600	276876	$\geq$ 3600	1390365
85	75	75	100	75	20.69	1221.35	107572	3537.29	1604647
86	75	75	100	50	21.87	2519.64	267744	$\geq$ 3600	1185965
87	75	75	100	50	24.30	1569.69	197996	$\geq$ 3600	1144374
88	100	25	50	100	19.05	$\geq$ 3600	289686	$\geq$ 3600	11369879
89	100	25	50	100	25.84	$\geq$ 3600	276956	$\geq$ 3600	1133754
90	100	25	50	75	16.77	$\geq$ 3600	375530	$\geq$ 3600	11536111
91	100	25	50	75	27.42	$\geq$ 3600	298962	$\geq$ 3600	1101594
92	100	25	50	50	8.99	6.78	408	155.29	27742
93	100	25	50	50	8.18	79.48	5450	$\geq$ 3600	2647874
94	100	25	75	100	22.87	$\geq$ 3600	329070	$\geq$ 3600	9800375
95	100	25	75	100	30.57	$\geq$ 3600	327440	$\geq$ 3600	868634
96	100	25	75	75	25.19	$\geq$ 3600	360946	$\geq$ 3600	9755635
97	100	25	75	75	25.16	$\geq$ 3600	220680	$\geq$ 3600	983853
98	100	25	75	50	26.23	1030.75	100044	$\geq$ 3600	1049577

Table 10 Performance on bilevel problems with larger dimensions of v.