B-347 Towards the Implementation of Successive Convex Relaxation Method for Nonconvex Quadratic Optimization Problems

Akiko Takeda[†], Yang Dai[‡], Mituhiro Fukuda[⋆] Masakazu Kojima[♯] March 1999

Abstract. Recently Kojima and Tunçel proposed new successive convex relaxation methods and their localized-discretized variants for general nonconvex quadratic programs. Although an upper bound of the objective function value within *a prior* precision can be found theoretically by solving a finite number of linear programs, several important implementation problems remain unsolved. In this paper we discuss these issues, present practically implementable algorithms and report numerical results.

Key words. Nonconvex Quadratic Program, Semidefinite Programming, Global Optimization, SDP Relaxation, Semi-Infinite LP Relaxation

† e-mail: *takeda@is.titech.ac.jp*

‡ e-mail: *dai@is.titech.ac.jp*
⋆ e-mail: *mituhiro@is.titech.ac.jp*

♯ e-mail: *kojima@is.titech.ac.jp*

1 Introduction

The general quadratic program (abbreviated by QP) covers various important nonconvex mathematical programs such as 0-1 linear and quadratic integer programs [20], linear complementarity problems [4], bilinear matrix inequalities [11, 16], bilevel linear and quadratic programs [30], and sum of linear fractional programs [6, 24]. A general QP can be represented by a problem in the following form [13, 14].

$$\begin{array}{ll} \max & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} & \boldsymbol{x} \in F. \end{array} \tag{1}$$

Here

$$\begin{array}{rcl} \boldsymbol{c} & \in & R^n \equiv \text{ the } n\text{-dimensional Euclidean space,} \\ \boldsymbol{a}^T & : & \text{the transposition of a vector } \boldsymbol{a} \in R^n, \\ F & \equiv & \{\boldsymbol{x} \in R^n : p(\boldsymbol{x}) \leq 0 \ (\forall p(\cdot) \in \mathcal{P}_F)\}, \\ \mathcal{P}_F & \subseteq & \mathcal{Q} \equiv \text{ the set of quadratic functions on } R^n. \end{array}$$

Recently, Kojima and Tunçel [13] established a theoretical framework for two types of successive convex relaxation methods for solving QP (1), which are extensions of Lovász-Schrijver's lift-and-project procedures [15]. Theoretically, starting from an initially known compact convex set C_0 containing F, their methods successively construct tighter approximations C_k $(k = 1, 2, ...,)$ of the convex hull of F (abbreviated by c.hull(F)) by repeating the lift-and-project procedure with the use of semidefinite programming (abbreviated by SDP) and semi-infinite linear programming (abbreviated by LP) relaxations. Therefore, maximizing the linear objective function over the approximation C_k of c.hull(F) generated by the algorithm, we obtain successively improved upper bounds of the objective function. Although both of the algorithms enjoy the global convergence, they involve in solving an infinite number of semi-infinite SDPs (or semi-infinite LPs) to generate a new approximation C_k of c.hull(F). Hence, the implementation can not be practically conducted on a computer which is available nowadays. To resolve the difficulty, they proposed corresponding variants of the methods in a later paper [14] by bringing two new techniques, "discretization" and "localization" into the theoretical framework. The "discretization" makes it possible to approximate an infinite number of semi-infinite SDPs (or semi-infinite LPs) which need to be solved at each iteration of the algorithm by a finite number of standard SDPs (or standard LPs, respectively) with a finite number of inequalities. The "localization" allows one to generate a convex relaxation of c.hull(F) which has a good approximation around the neighborhood of the objective direction. The motivations behind are (1) the good approximation of c.hull(F) in that direction may contribute more in finding a good upper bound of the objective function, and (2) it eliminates some redundant work to make the convex relaxation accurate in unnecessary directions. They concluded that for any given small value $\epsilon > 0$, with the appropriately organized discretization and localization procedures, new variants of the methods will generate an ϵ-approximation of the objective function.

However, the details of discretization and localization have not been studied, and the effect of specifically chosen discretization and localization procedures on the efficiency of

the algorithm is not clear. In this paper, we focus on the discretized-localized version of the successive semi-infinite LP relaxation method (abbreviated by SSILP), discuss the above issues, and propose implementable algorithms. The numerical results show that the new successive convex relaxation method is quite effective in terms of quality of the approximation. It even produces better approximation comparing to algorithms of lift-and-project with one application of SDP and semi-infinite LP relaxations. See [1, 3, 8, 10, 17, 20, 27, 31] for SDP relaxation and [25, 26] for semi-infinite LP relaxations (also called reformulation-convexification).

The remaining of the paper is organized as follows. Section 2 introduces previous results of the discretized-localized SSILP method. In Section 3 we discuss implementable discretization and localization procedures and present practical versions of the algorithm. In Section 4 we report numerical results of the proposed algorithm. In Section 5 we give the conclusion.

2 Previous Results

2.1 Preliminary

Let $\mathcal{S}^n$ denote the set of $n \times n$ symmetric matrices. Given a quadratic function defined on R^n, i.e.,

$$p(\boldsymbol{x}) \equiv \gamma + 2\boldsymbol{q}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} \ (\forall \boldsymbol{x} \in R^n),$$

where $\gamma \in R$, $\boldsymbol{q} \in R^n$ and $\boldsymbol{Q} \in \mathcal{S}^n$, we designate this function by $qf(\cdot; \gamma, \boldsymbol{q}, \boldsymbol{Q})$. Then we can represent the set $\mathcal{Q}$ of quadratic functions and the set $\mathcal{L}$ of linear functions on R^n by

$$\begin{aligned} \mathcal{Q} &\equiv \{qf(\cdot; \gamma, \boldsymbol{q}, \boldsymbol{Q}) : \gamma \in R,\ \boldsymbol{q} \in R^n,\ \boldsymbol{Q} \in \mathcal{S}^n\}, \\ \mathcal{L} &\equiv \{qf(\cdot; \gamma, \boldsymbol{q}, \boldsymbol{Q}) : \gamma \in R,\ \boldsymbol{q} \in R^n,\ \boldsymbol{Q} = \boldsymbol{O}\}, \end{aligned}$$

respectively. The problem QP (1) can be written as

$$\begin{aligned} F &\equiv \{\boldsymbol{x} \in R^n : p(\boldsymbol{x}) \leq 0 \ (\forall p(\cdot) \in \mathcal{P}_F)\}, \\ \mathcal{P}_F &\subseteq \mathcal{Q}, \\ \zeta^* &\equiv \sup\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{x} \in F\}. \end{aligned}$$

We call $\mathcal{P}_F$ a *quadratic inequality representation* of F. The following two conditions [14] are necessary for the convergence of the algorithm stated in Section 2.2.

Condition 2.1

(i) F is compact.

(ii) A compact convex set $C_0 \subseteq R^n$ such that

$$\begin{aligned} F &\subseteq C_0, \\ \eta_0 &\equiv \textit{the diameter of } C_0 = \max\{\|\boldsymbol{x}' - \boldsymbol{x}\| : \boldsymbol{x}',\ \boldsymbol{x} \in C_0\} > 0 \end{aligned}$$

is known in advance.

(iii) $\|\boldsymbol{c}\| = 1$.

Note that if η_0 was zero in (ii) then F would consist of at most one point; hence QP (1) would be trivial. Also (iii) is not essential because we may assume it without loss of generality whenever $\boldsymbol{c} \neq \boldsymbol{0}$.

Condition 2.2

(iv) The quadratic inequality representation $\mathcal{P}_F$ of F is finite.

(v) The compact convex set C_0 containing F (see (ii) above) has a finite linear inequality representation $\widetilde{\mathcal{P}}_0$;

$$\left.\begin{array}{c} F \subseteq C_0 \equiv \{\boldsymbol{x} \in R^n : p(\boldsymbol{x}) \leq 0 \ (\forall p(\cdot) \in \widetilde{\mathcal{P}}_0)\} \\ \textit{for} \ \ \exists \ \ \textit{finite} \ \widetilde{\mathcal{P}}_0 \subseteq \mathcal{L}. \end{array}\right\} \tag{2}$$

Let

$$\begin{array}{rcl} \boldsymbol{Q} \bullet \boldsymbol{X} & \equiv & \text{the (trace) inner product of two symmetric matrices } \boldsymbol{Q} \text{ and } \boldsymbol{X}, \\ & & \text{i.e., } \boldsymbol{Q} \bullet \boldsymbol{X} \equiv \sum_i \sum_j Q_{ij} X_{ij}, \\ \overline{D} & \equiv & \text{the set of direction vectors with the unit length} \\ & \equiv & \{\boldsymbol{d} \in R^n : \|\boldsymbol{d}\| = 1\}, \\ D(\boldsymbol{w}, \delta) & \equiv & \{\boldsymbol{u} \in \overline{D} \ : \ \|\boldsymbol{u} - \boldsymbol{w}\| \leq \delta\} \ (\forall \delta > 0 \ \text{ and } \forall \boldsymbol{w} \in \overline{D}). \end{array}$$

Let C be a compact subset of R^n. For $\forall \boldsymbol{d}, \ \forall \boldsymbol{d}_1, \ \forall \boldsymbol{d}_2 \in \overline{D}$ and $\forall \boldsymbol{x} \in R^n$, define

$$\begin{array}{rcl} \alpha(C, \boldsymbol{d}) & \equiv & \sup\{\boldsymbol{d}^T \boldsymbol{x} \ : \ \boldsymbol{x} \in C\}, \\ \ell sf(\boldsymbol{x}; C, \boldsymbol{d}) & \equiv & \boldsymbol{d}^T \boldsymbol{x} - \alpha(C, \boldsymbol{d}), \\ r2sf(\boldsymbol{x}; C, \boldsymbol{d}_1, \boldsymbol{d}_2) & \equiv & -(\boldsymbol{d}_1^T \boldsymbol{x} - \alpha(C, \boldsymbol{d}_1))(\boldsymbol{d}_2^T \boldsymbol{x} - \alpha(C, \boldsymbol{d}_2)), \\ r1sf(\boldsymbol{x}; C, \boldsymbol{d}) & \equiv & r2sf(\boldsymbol{x}; C, \boldsymbol{d}, -\boldsymbol{d}) \\ & = & -(\boldsymbol{d}^T \boldsymbol{x} - \alpha(C, \boldsymbol{d}))(-\boldsymbol{d}^T \boldsymbol{x} - \alpha(C, -\boldsymbol{d})). \end{array}$$

We call $\ell sf(\cdot; C, \boldsymbol{d})$ *a linear supporting function for C in a direction $\boldsymbol{d} \in \overline{D}$*, $r2sf(\cdot; C, \boldsymbol{d}_1, \boldsymbol{d}_2)$ *a rank-2 (quadratic) supporting function for C in a pair of directions $\boldsymbol{d}_1, \boldsymbol{d}_2 \in \overline{D}$*, and $r1sf(\cdot; C, \boldsymbol{d})$ *a rank-1 (quadratic) supporting function for C in a direction $\boldsymbol{d} \in \overline{D}$*. Let

$$\begin{array}{rcl} \mathcal{P}^L(C, D) & \equiv & \{\ell sf(\cdot; C, \boldsymbol{d}) : \boldsymbol{d} \in D\} \ (\forall D \subseteq \overline{D}), \\ \mathcal{P}^2(C, D_1, D_2) & \equiv & \{r2sf(\cdot; C, \boldsymbol{d}_1, \boldsymbol{d}_2) : \boldsymbol{d}_1 \in D_1, \ \boldsymbol{d}_2 \in D_2\} \ (\forall D_1, \ D_2 \subseteq \overline{D}), \\ \mathcal{P}^1(C, D) & \equiv & \{r1sf(\cdot; C, \boldsymbol{d}) : \boldsymbol{d} \in D\} \ (\forall D \subseteq \overline{D}). \end{array}$$

Let $\mathcal{P} \subseteq \mathcal{Q}$ be a set of quadratic functions. The semi-infinite LP relaxation of the set

$$\{\boldsymbol{x} \in C_0 : qf(\boldsymbol{x}; \gamma, \boldsymbol{q}, \boldsymbol{Q}) \leq 0 \ (\forall qf(\cdot; \gamma, \boldsymbol{q}, \boldsymbol{Q}) \in \mathcal{P})\}$$

is defined by

$$\widehat{F}^L(C_0, \mathcal{P}) \equiv \left\{ \boldsymbol{x} \in C_0 : \begin{array}{l} \exists \boldsymbol{X} \in \mathcal{S}^n \text{ such that} \\ \gamma + 2\boldsymbol{q}^T\boldsymbol{x} + \boldsymbol{Q} \bullet \boldsymbol{X} \leq 0 \ (\forall qf(\cdot;\gamma,\boldsymbol{q},\boldsymbol{Q}) \in \mathcal{P}) \end{array} \right\}.$$

The discretized-localized SSILP algorithm needs the following additional notations. Let

$$D_1 \equiv \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n, -\boldsymbol{e}_1, -\boldsymbol{e}_2, \ldots, -\boldsymbol{e}_n\}, \tag{3}$$

where $\boldsymbol{e}_i$ denote the ith unit coordinate vector ($i = 1, 2, \ldots, n$).

Let $D \subseteq \overline{D}$. For $\forall \delta \geq 0$, a subset D' of $\overline{D}$ is *a δ-net of D* if

$$\text{for } \forall \boldsymbol{d} \in D \text{ there exists a } \boldsymbol{d}' \in D' \text{ such that } \|\boldsymbol{d}' - \boldsymbol{d}\| \leq \delta.$$

By definition, if $\delta > \delta' \geq 0$ and D' is a δ'-net of D, then D' is a δ-net of D. Particularly, D itself is a δ-net of D for any $\delta \geq 0$. It should be noted that we can take a finite δ-net D' of D whenever $\delta > 0$. Thus a δ-net D' of D with $\delta > 0$ leads us to a finite discretization of D.

2.2 Previous Results

Consider a δ-net D_2, i.e., a finite discretization of the set

$$D(\boldsymbol{c}, \kappa) \equiv \{\boldsymbol{d} \in \overline{D} \ : \ \|\boldsymbol{d} - \boldsymbol{c}\| \leq \kappa\}$$

for given $\kappa > 0$ and $\delta > 0$. Kojima and Tunçel [14] proposed a discretized-localized successive semi-infinite linear relaxation method (DLSSILP) for QP (1) as follows.

Algorithm 2.3 (Algorithm DLSSILP)

Step 0: Let D_1 be defined by (3). Choose a direction set $D_2 \subseteq \bar{D}$. Let $k = 0$.

Step 1: If $C_k = \emptyset$ then stop. Otherwise compute $\zeta_k = \max\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{x} \in C_k\}$.

Step 2: Compute $\alpha(C_k, \boldsymbol{d}) = \max\{\boldsymbol{d}^T\boldsymbol{x} : \boldsymbol{x} \in C_k\}$ $(\forall \boldsymbol{d} \in D_1 \cup D_2)$.

Step 3: Let $\mathcal{P}_k = \mathcal{P}^L(C_k, D_1) \cup \mathcal{P}^2(C_k, D_1, D_2)$.

Step 4: Let

$$\begin{aligned} C_{k+1} &= \widehat{F}^L(C_0, \mathcal{P}_F \cup \mathcal{P}_k) \\ &\equiv \left\{ \boldsymbol{x} \in C_0 : \begin{array}{l} \exists \boldsymbol{X} \in \mathcal{S}^n \text{ such that} \\ \gamma + 2\boldsymbol{q}^T\boldsymbol{x} + \boldsymbol{Q} \bullet \boldsymbol{X} \leq 0 \ (\forall qf(\cdot;\gamma,\boldsymbol{q},\boldsymbol{Q}) \in \mathcal{P}_F \cup \mathcal{P}_k) \end{array} \right\}. \end{aligned}$$

Step 5: Let $k = k + 1$, and go to Step 1.

The algorithm generates a sequence $\{C_k \ (k = 0, 1, 2, \ldots, \ )\}$ of convex subsets of C_0 and a sequence $\{\zeta_k \ (k = 0, 1, 2, \ldots, \ )\}$ of real numbers. We note here that in Step 3 $\mathcal{P}_k = \mathcal{P}^L(C_k, D_1) \cup \mathcal{P}^2(C_k, D_1, D_2)$ does not form a quadratic inequality representation of C_k in general, i.e., C_k can be a proper subset of $\{\boldsymbol{x} \in R^n : p(\boldsymbol{x}) \leq 0 \ (\forall p(\cdot) \in \mathcal{P}_k)\}$ unless $D_2 = \bar{D}$. Nevertheless, the algorithm possesses a good property in the sense that both $\{C_k \ (k = 0, 1, 2, \ldots, \ )\}$ and $\{\zeta_k \ (k = 0, 1, 2, \ldots, \ )\}$ converge globally as shown in the following theorem.

Theorem 2.4 *([14]) Suppose that Condition 2.1 holds. Let $\kappa > 0$, $\epsilon > 0$, and D_1 be given by (3). There exists a $\delta > 0$ such that if we take a δ-net D_2 of $D(\boldsymbol{c}, \kappa)$, then $\{C_k \ (k = 0, 1, 2, \ldots \ )\}$ and $\{\zeta_k \ (k = 0, 1, 2, \ldots \ )\}$ generated by Algorithm 2.3 satisfy the following properties.*

(a) c.hull$(F) \subseteq C_{k+1} \subseteq C_k$ *and* $\zeta^* \leq \zeta_{k+1} \leq \zeta_k$ $(\forall k = 0, 1, 2, \ldots, \ )$ *(monotonicity),*

(b) if $F = \emptyset$ then $C_k = \emptyset$ for $\exists$ finite number k (detecting-infeasibility),

(c) if $F \neq \emptyset$ then $\zeta^* \equiv \max\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{x} \in F\} \leq \zeta_k \equiv \max\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{x} \in C_k\} < \zeta^* + \epsilon$ for $\exists$ finite number k.

We call ζ_k satisfying $\zeta_k < \zeta^* + \epsilon$ an *ϵ-approximation* of QP (1). It should be noted that Condition 2.2 and the δ-net enable C_k to be defined by finitely many constraints. Consequently, the maximization of linear function $\boldsymbol{c}^T\boldsymbol{x}$ or $\boldsymbol{d}^T\boldsymbol{x}$ $(\forall \boldsymbol{d} \in D_1 \cup D_2)$ over C_k in Algorithm 2.3 is a LP with finitely many inequality constraints. The number of LPs needed to be solved is finite in Step 2. Moreover, each LP in one iteration has the same feasible region C_k.

3 Practically Implementable Algorithms

For arbitrarily given $\epsilon > 0$, $\kappa > 0$, there exist $\delta > 0$ and $k > 0$ such that if we take a δ-net D_2 of $D(\boldsymbol{c}, \kappa)$, then after iterating Steps 1-5 of Algorithm 2.3 for k times, an ϵ-approximation ζ_k of QP (1) will be found. See Theorem 2.4. However, no specific result on the relations between ϵ, κ and δ has been clarified. What we know is that for a fixed κ, if we ask the algorithm to generate an ϵ-approximation with small ϵ, we have to choose a δ-net with sufficiently small δ, and repeat the algorithm for sufficiently many times. A δ-net with smaller δ contains more vectors when κ is fixed. Consequently, the number of LP constraints of each C_k and the number of LPs to be solved will increase as ϵ becomes small.

The primary goal of this research is to study the efficient implementation of the algorithm. We hope that insights gaining from the experiments may lead to possible theoretical results on the convergent rate of the algorithm.

Concerning the implementation of Algorithm DLSSILP, several issues have to be addressed. (a) How large should we take the neighborhood in the direction of the objective function so that a good approximation of QP (1) could be achieved? (b) How many vectors should be included in a δ-net? (c) How do we distribute them?

In this section we discuss these issues and present practically implementable algorithms.

3.1 Choices of δ-net

In their papers, Kojima and Tunçel presented a family of rank-2 quadratic supporting functions for compact sets from the following set of vectors, which is used for the proof of the global convergence of the algorithm (see [13, 14] for details). Although the set is mainly constructed for the purpose of proof and may not be the best choice of δ-net from viewpoint of efficiency of the algorithm, we use it as our primary choice of δ-net.

Let $\mathcal{N} = \{1, 2, \ldots, n\}$. For a fixed $\theta \in [0, \pi/2]$ and $\forall i \in \mathcal{N}$, define

$$\begin{aligned} \boldsymbol{b}_i(\theta, \boldsymbol{c}) &\equiv (\boldsymbol{c}\cos\theta + \boldsymbol{e}_i \sin\theta)/\|\boldsymbol{c}\cos\theta + \boldsymbol{e}_i \sin\theta\|, \\ \bar{\boldsymbol{b}}_i(\theta, \boldsymbol{c}) &\equiv (\boldsymbol{c}\cos\theta - \boldsymbol{e}_i \sin\theta)/\|\boldsymbol{c}\cos\theta - \boldsymbol{e}_i \sin\theta\|, \end{aligned} \tag{4}$$

$$D' \equiv \{\boldsymbol{b}_i(\theta, \boldsymbol{c}), \bar{\boldsymbol{b}}_i(\theta, \boldsymbol{c}) \ (i \in \mathcal{N})\}. \tag{5}$$

Then $\boldsymbol{b}_i(\theta, \boldsymbol{c}), \bar{\boldsymbol{b}}_i(\theta, \boldsymbol{c}) \in \bar{D}$ and $D' \subseteq \bar{D}$. The distance of any pair of vectors from D' is determined completely by the value of θ. Define

$$\kappa \equiv \max\{\|\boldsymbol{d} - \boldsymbol{c}\| \mid \boldsymbol{d} \in D'\}$$

and

$$\delta \equiv \max_{\boldsymbol{d} \in D(\boldsymbol{c}, \kappa)} \min_{\boldsymbol{d}' \in D' \cup \{\boldsymbol{c}\}} \|\boldsymbol{d} - \boldsymbol{d}'\|.$$

Then the set $D' \cup \{\boldsymbol{c}\}$ is a δ-net of $D(\boldsymbol{c}, \kappa)$. By changing the value of θ, we obtain different values of κ and δ. Here we include the vector $\boldsymbol{c}$ in the δ-net, since when value $\zeta_k = \alpha(C_k, \boldsymbol{c})$ is updated, the corresponding rank-2 quadratic constraints

$$-(\boldsymbol{c}^T\boldsymbol{x} - \alpha(C_k, \boldsymbol{c}))(\boldsymbol{d}_1^T\boldsymbol{x} - \alpha(C_k, \boldsymbol{d}_1)) \leq 0 \ (\forall \boldsymbol{d}_1 \in D_1)$$

are expected to cut off unnecessary area so that a good approximation of neighborhood of c.hull(F) in the direction of the objective function might be obtained. These constraints also force $\{\zeta_k \ (k = 0, 1, 2, ...,)\}$ to be a nonincreasing sequence.

It should be noted that with κ and θ defined above we are not guaranteed to obtain an ϵ-approximation of the objective function after a reasonable number of iterations. One needs to find a δ-net which is sufficiently fine so that the ϵ-approximation can be obtained in a finite number iterations. However, no previous information is available on how to make the refinement. Since Algorithm DLSSILP solves LPs for many times, increasing the number of vectors in a δ-net will result in a substantial increase in the amount of work spent on solving LPs due to the increase of the size of constraints as well as to the increase of the number of LPs to be solved. More precisely, if we take $D' \cup \{\boldsymbol{c}\}$ as a δ-net, the number of quadratic functions in $\mathcal{P}_k$ will be as many as $O(n^2)$. Although the refinement may lead to the decrease in iterations of the algorithm for obtaining a solution with *a prior* precision, it still seems reasonable to use a δ-net with cardinality of $O(n)$.

A possible refinement of a δ-net $D' \cup \{\boldsymbol{c}\}$ with cardinality $O(n)$ could be constructed as follows. For the given θ, if we halve the value of θ, i.e., take $\theta' = \theta/2$, and generate vectors $\boldsymbol{b}_i(\theta', \boldsymbol{c})$ and $\bar{\boldsymbol{b}}_i(\theta', \boldsymbol{c})$ $(i \in \mathcal{N})$ as above. Add those vectors to the set $D' \cup \{\boldsymbol{c}\}$, then the new set is also a δ-net of $D(\boldsymbol{c}, \kappa)$. Actually it is a δ'-net of $D(\boldsymbol{c}, \kappa)$ for some $\delta' < \delta$. By our preliminary testing, the refinement did make improvement on the quality of the approximation. However, the effect is relatively small and the computation time increases significantly. Therefore we shall use D_2

$$D_2 \equiv D' \cup \{\boldsymbol{c}\}$$

as the δ-net of $D(\boldsymbol{c}, \kappa)$ hereafter for the implementation of the algorithm. Note that the normalization of the vectors $\boldsymbol{c}$, $\boldsymbol{b}_i(\theta, \boldsymbol{c})$, $\bar{\boldsymbol{b}}_i(\theta, \boldsymbol{c})$ is only for the convenience of the proof of Theorem 2.4. It is not necessary to do this when implement the algorithm.

Recall that to construct $\mathcal{P}_k = \mathcal{P}^L(C_k, D_1) \cup \mathcal{P}^2(C_k, D_1, D_2)$ in Step 3 of Algorithm 2.3 with the current sets D_1 and D_2, we have to solve $(4n+1)$ LPs. The number of supporting functions in the set $\mathcal{P}_k$ is $2n + (2n)(2n + 1)$, where the first term is the number of linear support supporting functions of $\mathcal{P}^L(C_k, D_1)$. From our preliminary test, it turns out that the values

$$\alpha(C_k, \boldsymbol{d}_1) = \max\{\boldsymbol{d}_1^T \boldsymbol{x} : \boldsymbol{x} \in C_k\} \ (\forall \boldsymbol{d}_1 \in D_1) \tag{6}$$

change very little when C_k is updated. Hence we adopt $\mathcal{P}^L(C_0, D_1)$ in stead of $\mathcal{P}^L(C_k, D_1)$, $\mathcal{P}^1(C_0, D_1)$ in stead of $\mathcal{P}^1(C_k, D_1)$, and use $\alpha(C_0, \boldsymbol{d}_1)$ $(\forall \boldsymbol{d}_1 \in D_1)$ and $\alpha(C_k, \boldsymbol{d}_2)$ $(\forall \boldsymbol{d}_2 \in D_2)$ to generate rank-2 quadratic supporting functions (see definition in Section 2) at each iteration. This will save considerable amount of work on solving LPs of (6). We denote the rank-2 quadratic supporting functions generated in this way by $\mathcal{P}^2(C_0, D_1; C_k, D_2)$ to avoid confusion.

We also change the distribution of the vectors in the δ-net while running the algorithm. The vectors in the δ-net are replaced when the function value reaches a plateau. This is carried out by decreasing the value of θ and replacing old $\boldsymbol{b}_i(\theta, \boldsymbol{c}), \bar{\boldsymbol{b}}_i(\theta, \boldsymbol{c})$ $(i \in \mathcal{N})$ in the δ-net with those corresponding to the new θ. But this procedure will be only repeated for K times till θ becomes a small value, since getting better approximation by the further

decrease of θ seems to be unlikely then. The rate of decreasing of θ will be determined by a sequence $\{\sigma_\ell\}_{\ell=0}^K$ $(1 = \sigma_0 > \sigma_1 > \cdots > \sigma_K)$, i.e., if the initial value of θ is θ_0, then at the ℓ-th $(\ell = 1, ..., K)$ replacement, θ is set to be equal to $\sigma_\ell \theta_0$.

3.2 Algorithms

With the δ-nets described in the previous subsection, we are ready to give a practical version of Algorithm DLSSILP.

Algorithm 3.1 (Algorithm DLSSILP1)

Input : an integer $K > 0$, real values $\theta_0 > 0,\ \epsilon_1 > 0,\ \epsilon_2 > 0$, and a sequence $\{\sigma_\ell\}_{\ell=0}^K$.
Output : a value $\bar{\zeta}$.

Step 0: Let $D_1 = \{\boldsymbol{e}_i, -\boldsymbol{e}_i\ (i \in \mathcal{N})\}$, $D_2 = \{\boldsymbol{c},\ \boldsymbol{b}_i(\theta_0, \boldsymbol{c}),\ \bar{\boldsymbol{b}}_i(\theta_0, \boldsymbol{c})\ (i \in \mathcal{N})\}$. Let $\zeta_{-1} = +\infty$.

Step 1: Compute $\alpha(C_0, \boldsymbol{d}) = \max\{\boldsymbol{d}^T \boldsymbol{x} : \boldsymbol{x} \in C_0\}$ $(\forall \boldsymbol{d} \in D_1 \cup D_2)$, let $k = 0$, $\ell = 0$.

Step 2: If $C_k = \emptyset$, let $\bar{\zeta} = -\infty$, stop. Otherwise compute $\zeta_k = \max\{\boldsymbol{c}^T \boldsymbol{x} : \boldsymbol{x} \in C_k\}$. If $\ell = K$ and $\frac{\zeta_{k-1} - \zeta_k}{\max\{|\zeta_k|,\ \epsilon_2\}} \leq \sigma_K \epsilon_1$, let $\bar{\zeta} = \zeta_k$, stop.

Step 3: If $\frac{\zeta_{k-1} - \zeta_k}{\max\{|\zeta_k|,\ \epsilon_2\}} \leq \sigma_\ell \epsilon_1$, then let $\ell = \ell + 1$, $\theta_\ell = \sigma_\ell \theta_0$, and replace the δ-net D_2 by the new vectors $\boldsymbol{b}_i(\theta_\ell, \boldsymbol{c}), \bar{\boldsymbol{b}}_i(\theta_\ell, \boldsymbol{c})$ and $\boldsymbol{c}$.

Step 4: Compute $\alpha(C_k, \boldsymbol{d}_2) = \max\{\boldsymbol{d}_2^T \boldsymbol{x} : \boldsymbol{x} \in C_k\}$ $(\forall \boldsymbol{d}_2 \in D_2)$.

Step 5: Let $\mathcal{P}_k = \mathcal{P}^L(C_0, D_1) \cup \mathcal{P}^1(C_0, D_1) \cup \mathcal{P}^2(C_0, D_1; C_k, D_2)$.

Step 6: Let

$$\begin{aligned} C_{k+1} &= \widehat{F}^L(C_0, \mathcal{P}_F \cup \mathcal{P}_k) \\ &\equiv \left\{ \boldsymbol{x} \in C_0 : \begin{array}{l} \exists \boldsymbol{X} \in \mathcal{S}^n \text{ such that} \\ \gamma + 2\boldsymbol{q}^T \boldsymbol{x} + \boldsymbol{Q} \bullet \boldsymbol{X} \leq 0\ (\forall qf(\cdot; \gamma, \boldsymbol{q}, \boldsymbol{Q}) \in \mathcal{P}_F \cup \mathcal{P}_k) \end{array} \right\}. \end{aligned}$$

Step 7: Let $k = k + 1$, and go to Step 2.

Remark 1. The set $\mathcal{P}_k$ in Step 5 is different from that in the original Algorithm DLSSILP. Here the set of rank-1 quadratic supporting function $\mathcal{P}^1(C_0, D_1)$ is included.

Remark 2. In Step 4 we solve $(2n + 1)$ LPs with the same feasible region C_k. It is not necessary to solve each of them from scratch. After the first one is solved, solutions of subsequent LPs could be found by the standard reoptimization procedure developed for sensitivity study of LP. This contributes a great deal to the efficiency of the implementation.

3.3 Other Variants of the Algorithm

As we mentioned before, $\mathcal{P}_k$ is not a quadratic representation of C_k in general. To save the amount of work in each iteration, we could even use less quadratic functions; for example, we could take

$$\bar{\mathcal{P}}^2(C_0, D_1; C_k, D_2) = \left\{ \begin{array}{l} -(\pm \boldsymbol{e}_i^T \boldsymbol{x} - \alpha(C_0, \pm \boldsymbol{e}_i))(\boldsymbol{b}_j(\theta, \boldsymbol{c})^T \boldsymbol{x} - \alpha(C_k, \boldsymbol{b}_j(\theta, \boldsymbol{c}))), \\ -(\pm \boldsymbol{e}_i^T \boldsymbol{x} - \alpha(C_0, \pm \boldsymbol{e}_i))(\bar{\boldsymbol{b}}_j(\theta, \boldsymbol{c})^T \boldsymbol{x} - \alpha(C_k, \bar{\boldsymbol{b}}_j(\theta, \boldsymbol{c}))), \\ -(\pm \boldsymbol{e}_i^T \boldsymbol{x} - \alpha(C_0, \pm \boldsymbol{e}_i))(\boldsymbol{c}^T \boldsymbol{x} - \alpha(C_k, \boldsymbol{c})), \\ (1 \leq \forall i \leq \forall j \leq n) \end{array} \right\}.$$

The number of quadratic supporting functions in $\bar{\mathcal{P}}^2(C_0, D_1; C_k, D_2)$ is $2n(n+2)$, which is almost half of that in $\mathcal{P}^2(C_0, D_1; C_k, D_2)$. Replacing $\mathcal{P}^2(C_0, D_1; C_k, D_2)$ with $\bar{\mathcal{P}}^2(C_0, D_1; C_k, D_2)$ in $\mathcal{P}_k$ we obtain another variant of the algorithm, which we call Algorithm DLSSILP2.

4 Computational Experiments

In this section we report computational results of the new algorithms. We use CPLEX (Ver 6.0) as LP solver and SDPA (Ver. 4.30) [9] as SDP solver. The program was coded in C++ and was run on a Digital Alpha Workstation (CPU Alpha 2116-400 MHz with 512 MB of memory).

4.1 Test Problems

Our set of test problems for the computational study consists of six types of problems from literature.

(a) Minimization of a linear function over linear and quadratic constraints (LQCP);

(b) Minimization of a concave quadratic function with linear constraints (CQP);

(c) Bilinear programs (BLP);

(d) Mixed quadratic 0-1 integer programs (MQI);

(e) Sum of linear fractional programs (SLFP);

(f) Bilevel quadratic programs (BLevel).

Since the transformation from problems (a)-(c) to our QP form (1) is straitforward, we only discuss about problems (d)-(f).

The general form of the mixed quadratic integer program [12] is

$$\begin{array}{ll} \max & \boldsymbol{q}^T \boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \gamma \\ \text{s.t.} & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \leq \boldsymbol{0}, \\ & x_i \in \{0, 1\} \text{ for } i \in \mathcal{N}, \end{array}$$

where $\boldsymbol{q}$, $\boldsymbol{Q}$, γ and $\boldsymbol{x}$ are defined as before, $\boldsymbol{A} \in R^{m\times n}$ and $\boldsymbol{b} \in R^n$. We rewrite the problem as

$$\begin{array}{ll} \max & x_{n+1} \\ \text{s.t.} & \boldsymbol{q}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \gamma - x_{n+1} \geq 0, \\ & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \leq \boldsymbol{0}, \\ & x_i(x_i - 1) = 0 \text{ for } i \in \mathcal{N}, \end{array}$$

which can be easily transformed to QP (1).

The sum of linear fractional programming problem [6, 24] considers

$$\begin{array}{ll} \max & \displaystyle\sum_{\ell\in\mathcal{K}} \frac{\boldsymbol{a}_\ell{}^T\boldsymbol{x} + a_{0\ell}}{\boldsymbol{b}_\ell{}^T\boldsymbol{x} + b_{0\ell}} \\ \text{s.t.} & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \leq \boldsymbol{0}, \end{array}$$

where $\mathcal{K} = \{1, 2, \ldots, k\}$, $\boldsymbol{a}_\ell, \boldsymbol{b}_\ell$ $(\ell \in \mathcal{K})$ are n-dimensional vectors, and $a_{0\ell}, b_{0\ell} (\ell \in \mathcal{K})$ are real values. We assume that $\boldsymbol{b}_\ell{}^T\boldsymbol{x} + b_{0\ell} > 0$ $(\ell \in \mathcal{K})$ for any feasible $\boldsymbol{x}$. By introducing a new variable $x_{n+\ell}$ $(\ell \in \mathcal{N})$ for each fractional term, we can replace the problem with the following equivalent one.

$$\begin{array}{ll} \max & \displaystyle\sum_{\ell\in\mathcal{K}} x_{n+\ell} \\ \text{s.t.} & \boldsymbol{a}_\ell{}^T\boldsymbol{x} + a_{0\ell} - (\boldsymbol{b}_\ell^T\boldsymbol{x})x_{n+\ell} - b_{0\ell}x_{n+\ell} \geq 0, \quad \forall \ell \in \mathcal{K} \\ & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \leq \boldsymbol{0}. \end{array} \tag{7}$$

The bilevel quadratic programming [12] solves the problem

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}} & F(\boldsymbol{x}, \boldsymbol{y}) \\ \text{s.t.} & \boldsymbol{x} \in X \\ & \boldsymbol{y} \in \left\{ \begin{array}{ll} \min\limits_{\boldsymbol{y}} & f(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{c}_y^T\boldsymbol{x} + \boldsymbol{d}_y^T\boldsymbol{y} + \boldsymbol{x}^TQ_x\boldsymbol{y} + \boldsymbol{y}^TQ_y\boldsymbol{y} \\ \text{s.t.} & A\boldsymbol{x} + B\boldsymbol{y} \leq \boldsymbol{b} \end{array} \right\}, \end{array}$$

where $F(\boldsymbol{x}, \boldsymbol{y})$ is a quadratic function of $\boldsymbol{x}$ and $\boldsymbol{y}$, and $f(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{c}_y^T\boldsymbol{x} + \boldsymbol{d}_y^T\boldsymbol{y} + \boldsymbol{x}^TQ_x\boldsymbol{y} + \boldsymbol{y}^TQ_y\boldsymbol{y}$ is a convex function of $\boldsymbol{y}$ when $\boldsymbol{x}$ is fixed. All the coefficient vectors and matrices have appropriate sizes and X is a polytope. Then, the KKT condition for the inner problem is both necessary and sufficient for the inner optimality, which preserves the equivalence of the above problem with the following one.

$$\begin{array}{ll} \max\limits_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{u}} & -F(\boldsymbol{x}, \boldsymbol{y}) \\ \text{s.t.} & A\boldsymbol{x} + B\boldsymbol{y} \leq \boldsymbol{b} \\ & 2\boldsymbol{y}^TQ_y + \boldsymbol{x}^TQ_x + \boldsymbol{u}^TB + \boldsymbol{d}_y^T = \boldsymbol{0}^T \\ & u_i(A\boldsymbol{x} + B\boldsymbol{y} - \boldsymbol{b})_i = 0, \\ & \boldsymbol{x} \in X, \boldsymbol{u} \geq \boldsymbol{0}. \end{array} \tag{8}$$

The constraints consist of linear and bilinear ones and can be considered as a QP (1) after an appropriate transformation.

4.2 Implementation

For the comparison, we implemented Algorithm DLSSILP1 with two different $\mathcal{P}_F$ and $\mathcal{P}_k$, and two other typical algorithms related with the lifting-and-project procedure for QP. The latter two algorithms are the SDP relaxation [1, 3, 8, 10, 17, 20, 21, 27, 31] and the LP relaxation proposed by Sherali and Tuncbilek [25, 26]. We call them Algorithms DLSSILP1, DLSSILP1+Sherali(LP), Sherali(SDP) and Sherali(LP), respectively. In the transformed QP form, there are usually linear constraints in addition to quadratic ones. The input data for the algorithms except the first one are identical and are generated as follows. Following the idea of the reformulation-convexification in [26], we generate quadratic constraints through the construction of pairwise linear constraint products. Those new and original quadratic constraints are considered as input data for the latter three algorithms.

An interesting observation is that if Algorithm DLSSILP1 is started with $\theta = 90^\circ$ it actually generates quadratic constraints which correspond to those from the pairwise products of lower and upper bound constraints. More precisely, when $\theta = 90^\circ$, vectors $\boldsymbol{b}_i, \bar{\boldsymbol{b}}_i$ $(i \in \mathcal{N})$ in the δ-net (see (4) in Section 3) are corresponding to unit vectors $\boldsymbol{e}_i, -\boldsymbol{e}_i$ $(i \in \mathcal{N})$, and the values $\alpha(\boldsymbol{e}_i, C_0), \alpha(-\boldsymbol{e}_i, C_0)$ $(i \in \mathcal{N})$ are upper and lower bounds of the variable x_i $(i \in \mathcal{N})$, respectively. The set $\mathcal{P}^2(C_0, D_1; C_0, D_2)$ constructed by Algorithm DLSSILP1 contains all quadratic functions from the pairwise products of lower and upper bound constraints. Therefore if the feasible region of the original problem is given only by those box constraints, then Sherali and Tuncbilek's method is identical to the first step of our algorithm starting with $\theta = 90^\circ$.

Box Constraints of Variables: Table 1 gives test problems that we used in numerical study. The feasible regions of those problems are defined by quadratic and/or linear constraints including lower and upper bounds for each variable. Note that we have preprocessed the test problems to obtain box constraints in case they are not given by the original data.

The transformation of the original problem into our QP uses an artificial variable, say t, to replace quadratic objective function or each linear fractional term, say $g(\boldsymbol{x})$. Without imposing box constraints on the new variable t, both Algorithm DLSSILP1 and the LP relaxation algorithm of Sherali and Tuncbilek [26] may fail to generate a solution due to the unboundness of the new variable. Therefore it is necessary to derive lower and upper bounds for the new variable before we start the algorithms.

The bounds for the new variable is calculated as follows. Suppose that l_i and u_i such that $l_i \leq x_i \leq u_i$ $(i \in \mathcal{N})$ are given by the original problem. If $g(\boldsymbol{x})$ is a quadratic function, then it contains nonlinear terms $x_i x_j$. Values of $\min\{l_i u_i, l_i u_j, l_j u_i, l_j u_j\}$ and $\max\{l_i u_i, l_i u_j, l_j u_i, l_j u_j\}$ can be used as lower and upper bounds of those nonlinear terms, respectively. Then lower and upper bounds of t can be generated by using bounds of the nonlinear terms and those of the original variables. When $g(\boldsymbol{x})$ is a fractional linear term, it is sufficient to take the ratio of the maximum value of the numerator to the minimum value of the denominator as an upper bound of t. Similarly, the ratio of the minimum value of the numerator to the maximum value of the denominator can serve as a lower bound of t. Since both the denominator and the numerator are linear functions of the original variables,

the maximum and the minimum values of the functions are not difficult to be calculated if box constraints of the original variables are given.

The only exception is problems of type (f). For those problems, since the computation of upper bounds for nonnegative variables u_i in (8) is not trivial, we only use a sufficiently large value ($u_i = 1000$) as an upper bound for each u_i.

In below, we designate the input data for Algorithm DLSSILP1 by DataDLSSILP. The other three algorithms use the same input data : the quadratic constraints in DataDLSSILP and pairwise products of linear constraints in DataDLPSSILP. We denote them as DataRelax.

Parameters: The parameters used by the algorithms are given in Table 2.

Remark: We start the algorithm with $\theta = 90^\circ$, since we observed from our experiments, it somehow generates better upper bounds for the most of the test problems. When we replace a δ-net, we decrease θ by a factor of $\frac{8}{9}$ at the first time, and then by a factor of $\frac{1}{2}$ for the other replacements ($\ell \geq 2$). This is also decided empirically.

Computational Results : We report numerical results for four algorithms in Table 4. The legends used are listed in Table 3. Statistical results of relative errors of the four algorithms are summarized in Table 5. Each number in columns 2-5 of Table 5 indicates the number of cases where the relative errors of the solutions are within the corresponding range. The last column gives the number of cases solved successfully by each algorithm. Since the SDP relaxation could only solve $\frac{2}{3}$ of the cases, we only compare the results of the first three algorithms. The failure of the SDP relaxation may be caused by the numerical instability due to the pairwise products of the linear constraints, since they could result in very big or very small coefficients at the same time.

Table 5 shows that on average, Algorithm DLSSILP1+Sherali(LP) performed best in generating good upper bounds for the test problems, and Algorithm Sherali(LP) generated less good ones. We also observed that Algorithm DLSSILP1+Sherali(LP) reported 24 cases with relative errors less than 0.01, and Algorithm DLSSILP1 reported 21 cases. We could conclude that Algorithm DLSSILP1 is also competent in obtaining good upper bounds for objective functions. For the aspect of computing time, Algorithm Sherali(LP) needs much less time, while Algorithm DLSSILP1+ Sherali(LP) is more time-consuming on average. However, there are few exceptions. For example, comparing to Algorithm DLSSILP1, Algorithm DLSSILP1+Sherali(LP) produced better solution for problem LQCP3 in less time. Another observation is that both of the successive LP relaxation algorithms seem to be more capable for solving difficult types of problems such as types (e) and (f) than Algorithm Sherali(LP) does.

Table 1: The test problems

Problem	Type	Source	n	m	#QP
LQCP1	(a)	[7]	3	3	1
LQCP2	(a)	[7]	7	7	3
LQCP3	(a)	[7]	8	6	3
LQCP4	(a)	[7]	6	7	7
LQCP5	(a)	[7]	9	10	4
LQCP6	(a)	[29]	7	7	3
LQCP7	(a)	[29]	2	4	2
LQCP8	(a)	[5, 28]	3	3	2
LQCP9	(a)	[5, 28]	10	11	4
CQP1	(b)	[7]	7	3	1
CQP2	(b)	[7]	11	6	1
CQP3	(b)	[7]	6	2	1
CQP4	(b)	[7]	21	11	1
CQP5	(b)	[7]	14	10	1
CQP6	(b)	[7]	7	6	1
CQP7	(b)	[7]	11	12	1
CQP8	(b)	[29]	5	7	1
BLP1	(c)	[22]	11	11	1
BLP2	(c)	[2]	11	14	1
BLP3	(c)	[2]	11	11	1
MQI1	(d)	[23]	5	11	7
MQI2	(d)	[12]	6	14	8
SLFP1	(e)	[6]	4	4	2
SLFP2	(e)	[6]	5	7	2
SLEP3	(e)	[6]	4	4	2
BLevel1	(f)	[30]	7	16	10
BLevel2	(f)	[30]	7	14	9
BLevel3	(f)	[30]	9	15	9
BLevel4	(f)	[30]	6	11	7
BLevel5	(f)	[30]	10	19	12

Legend : n and m denote the number of variables and the number of constraints (not include box constraints) in the transformed QP form, respectively. The notation #QP denotes the number of quadratic constraints. The second column gives the type of problems.

Table 2: Parameters for implementation

parameter	value
ϵ_1	0.001
ϵ_2	1.0
θ_0	90°
σ_0	1
σ_1	$\frac{8}{9}$
$\sigma_\ell(\ell \geq 2)$	$\sigma_1 * (\text{ratio}\,\theta)^{\ell-1}$
ratio θ	0.5
K	3

Table 3: Legends

Problem	:	type of problem;
DLSSILP1	:	algorithm using $\mathcal{P}^L(C_0, D_1) \cup \mathcal{P}^1(C_0, D_1) \cup \mathcal{P}^2(C_0, D_1; C_k, D_2)$ as $\mathcal{P}_k$ and DataDLSSILP as $\mathcal{P}_F$;
DLSSILP1+Sherali(LP)	:	algorithm using $\mathcal{P}^L(C_0, D_1) \cup \mathcal{P}^1(C_0, D_1) \cup \mathcal{P}^2(C_0, D_1; C_k, D_2)$ as $\mathcal{P}_k$ and DataRelax as $\mathcal{P}_F$;
Sherali(LP)	:	the LP relaxation algorithm with input DataRelax;
Sherali(SDP)	:	the SDP relaxation algorithm with input DataRelax;
R.error	:	the relative error of a solution, i.e., $\lvert\bar{\zeta} - \zeta^*\rvert/\lvert\zeta^*\rvert$, where $\bar{\zeta}$ is the solution value found by each algorithm, and ζ^* is the value of the optimal solution value or the best known solution value;
cpu	:	cpu time in second;
iter.	:	number of iterations (Steps 1- 5) the algorithm repeated.

Table 4: Computational results of four algorithms

Problem	DLSSILP1			DLSSILP1+Sherali(LP)			Sherali(LP)		Sherali(SDP)	
	R.error	cpu	iter.	R.error	cpu	iter.	R.error	cpu	R.error	cpu
LQCP1	4.68e-4	0.53	46	3.70e-4	0.43	30	4.23e-1	0.00	4.23e-1	0.48
LQCP2	1.02e-4	0.38	7	0.00e+0	0.52	5	1.83e-16	0.02	7.93e-6	16.93
LQCP3	2.94e-1	165.78	289	2.79e-2	35.18	51	5.20e-1	0.03	———	——
LQCP4	1.07e-3	0.25	6	9.70e-4	0.38	6	4.42e-3	0.02	———	——
LQCP5	1.56e-15	3.87	6	1.28e-15	7.13	6	2.50e-1	0.03	———	——
LQCP6	1.02e-4	0.47	8	4.77e-15	0.35	5	3.67e-16	0.02	9.39e-5	17.08
LQCP7	2.08e-2	0.13	27	1.35e-2	0.17	25	6.07e-2	0.00	7.56e-5	0.20
LQCP8	3.64e-3	0.25	35	1.93e-3	0.30	28	5.79e-1	0.00	5.79e-1	0.35
LQCP9	1.28e-15	3.85	7	1.85e-15	31.83	6	2.50e-1	0.05	———	——
CQP1	9.34e-16	0.20	5	0.00e+0	0.33	5	0.00e+0	0.03	1.66e-6	7.02
CQP2	1.18e-3	9.22	20	5.47e-15	5.58	6	2.12e-2	0.20	2.18e-2	80.07
CQP3	4.05e-2	2.63	37	2.06e-2	2.45	28	6.82e-2	0.02	6.82e-2	8.85
CQP4	4.26e-2	352.73	63	———	——	–	2.08e-4	6.40	2.80e-3	1931.52
CQP5	0.00e+0	3.57	5	5.92e-16	2.97	5	2.37e-16	0.15	3.26e-6	243.65
CQP6	3.23e-16	0.33	5	1.61e-16	1.18	5	0.00e+0	0.05	2.00e-5	15.43
CQP7	4.03e-4	1.42	7	1.18e-7	9.52	5	1.18e-7	0.53	1.57e-5	199.42
CQP8	2.73e-16	0.15	6	0.00e+0	0.27	5	0.00e+0	0.02	2.96e-6	7.30
BLP1	8.25e-7	1.05	6	4.18e-6	4.38	5	4.18e-6	0.15	1.72e-4	178.22
BLP2	1.09e-3	9.08	15	5.14e-6	74.02	5	5.14e-6	1.03	1.24e-1	240.27
BLP3	6.38e-6	1.07	6	1.37e-6	4.17	5	1.37e-6	0.15	1.25e-4	142.53
MQI1	2.02e-16	0.11	5	0.00e+0	0.18	5	2.02-16	0.03	1.66e-6	4.15
MQI2	6.92e-2	10.17	116	6.34e-16	1.92	8	3.33e-1	0.02	6.74e-6	13.85
SLFP1	1.78e-16	0.18	9	3.04e-8	0.23	9	2.92e-1	0.00	———	——
SLFP2	9.61e-4	0.57	16	9.76e-5	1.30	14	3.54e-1	0.02	3.54e-1	4.68
SLFP3	3.16e-2	1.23	44	2.96e-2	1.48	40	2.58e-1	0.00	2.58e-1	1.08
BLevel1	5.74e-2	73.43	509	3.57e-8	0.73	5	3.57e-8	0.03	———	——
BLevel2	2.95e-1	58.52	382	3.37e-3	23.95	72	7.06e-1	0.03	———	——
BLevel3	2.23e-4	1.78	15	1.65e-12	2.72	7	2.22e-1	0.05	———	——
BLevel4	1.65e-2	0.90	27	1.38e-2	1.27	22	1.00e+0	0.02	———	——
BLevel5	6.76e-3	52.10	80	1.59e-4	32.65	23	1.16e-1	0.05	———	——

The empty entry shows the algorithm could not obtain a solution.

Behavior of the algorithms : Experiments were conducted in order to see how the other factors effect the behavior of the algorithms. The following three factors were considered : (i) the different quadratic representation $\mathcal{P}_k$ of C_k; (ii) δ-nets without replacement; and (iii) the number of replacements of the δ-net. We picked up one specific problem with relatively large numbers of variables and constraints from each group of the problems in Table 1 for

Table 5: Statistical results of relative error

Algorithm	range of relative error				#. cases solved
	$[0, 0.01)$	$[0.01, 0.1)$	$[0.1, 1)$	$[1, +\infty)$	
DLSSILP1	21	7	2	0	30
DLSSILP1+Sherali(LP)	24	5	0	0	29
Sherali(LP)	14	3	12	1	30
Sherali(SDP)	13	2	5	0	20

the test.

(i) As we discussed in Section 3.3, set $\bar{\mathcal{P}}^2(C_0, D_1; C_k, D_2)$ could be chosen as an alternative set of $\mathcal{P}^2(C_0, D_1; C_k, D_2)$ to represent $\mathcal{P}_k$ in the algorithm. Algorithm DLSSILP2 uses $\bar{\mathcal{P}}^2(C_0, D_1; C_k, D_2)$ as $\mathcal{P}^2(C_0, D_1; C_k, D_2)$ in the representation of $\mathcal{P}_k$, and all parameters are the same as in Algorithm DLSSILP1. The numerical results of these two algorithms are presented in Tables 6-7. Algorithm DLSSILP1 are slightly better than Algorithm DLSSILP2 in terms of generating good upper bounds. But the former takes more time. There exists almost no difference between the algorithms DLSSILP1+Sherali(LP) and DLSSILP2+Sherali(LP) as far as the quality of the solution is concerned. But the former consumes more time.

(ii) To see the behavior of variants of Algorithm DLSSILP1 where a δ-net was fixed throughout the execution of the algorithm, we chose Problem CQP3 and SLFP3 as examples. The initial values of θ_0 for the fixed δ-net was set as 10°, 40° and 80°, respectively. In Figures 1-2, CQP3-10 (SLFP3-10), CQP3-40 (SLFP3-40) and CQP3-80 (SLFP3-80) are corresponding to each such case, respectively. CQP3-replace (SLFP3-replace) indicates the result where the δ-net was replaced. Here K was set as 5, and all other parameters were as the same as in Table 3. For the sake of representation, we only show the results up to 120 and 30 iterations for Problem CQP3 and Problem SLFP3, respectively. It is obvious from the figures that the replacement of the δ-net is usually effective to reduce the relative error quickly.

(iii) Table 8 shows the results of Algorithm DLSSILP1 when different numbers of replacements, e.g. $K = 3,\ 5$, are selected. It seems that we could obtain relatively good upper bounds when $K = 3$. This is also confirmed from Figures 1-2, as the drastic decrease of the relative error usually occur at the early stage of the execution of the algorithm.

Table 6: Results of Algorithms DLSSILP1 and DLSSILP2

Problem	DLSSILP1			DLSSILP2		
	R.error	cpu	iter.	R.error	cpu	iter.
LQCP3	2.94e-1	165.78	289	3.37e-1	115.65	436
CQP4	4.26e-2	352.73	63	1.14e-1	61.45	33
BLP2	1.09e-3	9.08	15	4.92e-3	4.70	19
MQI2	6.92e-2	10.17	116	1.36e-1	4.20	99
SLFP3	3.16e-2	1.23	44	3.92e-2	0.60	34
BLevel2	2.95e-1	58.52	382	8.82e-1	1.43	25

Table 7: Results of Algorithms DLSSILP1+Sherali(LP) and DLSSILP2+Sherali(LP)

Problem	DLSSILP1+Sherali(LP)			DLSSILP2+Sherali(LP)		
	R.error	cpu	iter.	R.error	cpu	iter.
LQCP3	2.78e-2	35.18	51	3.59e-2	22.20	51
CQP4	———	——	–	———	——	–
BLP2	5.14e-6	74.02	5	5.14e-6	39.83	5
MQI2	6.34e-16	1.92	8	2.54e-16	1.32	9
SLFP3	2.96e-2	1.48	40	4.36e-2	0.73	29
BLevel2	3.37e-3	23.95	72	6.76e-2	28.75	77

The empty entry shows the algorithm could not obtain a solution.

Table 8: Results for different numbers of replacements of δ-net

Problem	DLSSILP1 (K= 3)			DLSSILP1 (K= 5)		
	R.error	cpu	iter.	R.error	cpu	iter.
LQCP3	2.94e-1	166.72	289	2.67e-1	236.65	363
CQP4	4.26e-2	343.35	63	3.99e-2	551.02	91
BLP2	1.09e-3	8.97	15	1.02e-3	11.10	17
MQI2	6.92e-2	10.28	116	5.02e-2	15.18	181
SLFP3	3.30e-2	1.37	45	1.88e-2	2.42	83
BLevel2	2.95e-1	57.00	382	2.95e-1	57.20	384

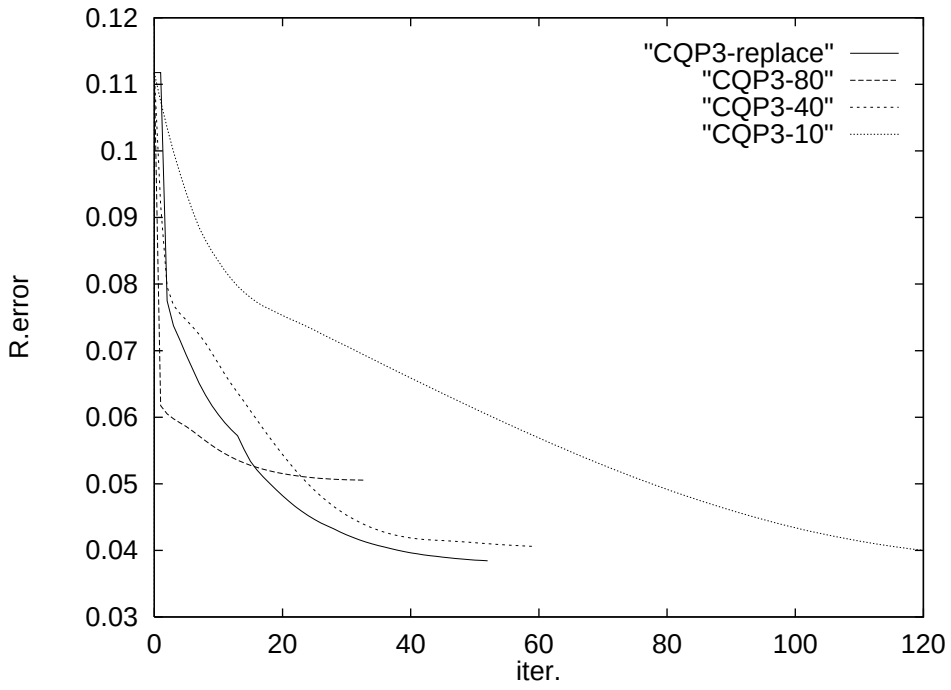

Figure 1: Behavior of Algorithm DLSSILP1 with fixed and varied δ-net for problem CQP3

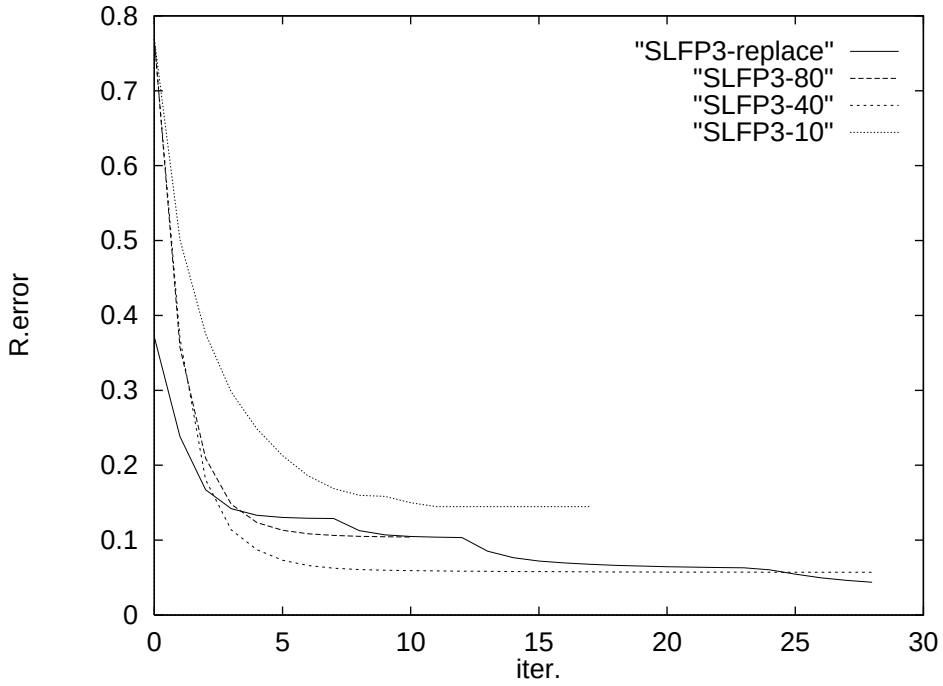

Figure 2: Behavior of Algorithm DLSSILP1 with fixed and varied δ-net for problem SLFP3

5 Conclusion

In this paper we presented practically implementable versions of the discretized-localized successive semi-infinite linear programming relaxation method proposed by Kojima and Tunçel. We studied the behavior of the algorithms through computational experiments. The numerical results demonstrated that the algorithms could generate relatively good upper bounds of the objective functions for most of the test problems. As a primary relaxation method they are quite satisfactory.

Several issues are left for further study. The most challenging problem is to give theoretical analysis for the convergence rate of the algorithm. For the practical implementation, attempting different types of δ-net by exploring special structures of problems is important not only in making the algorithm more efficient but also in obtaining information for the theoretical study. Moreover, suitable data structures which enable to solve problems of large size should be developed. Further extension of the research could be to embed this algorithm within a branch-and-bound algorithm for solving difficult classes of nonconvex quadratic programs.

Acknowledgment

The authors would like to thank Professor Yinyu Ye of Iowa University and Dr. Thomas

G. W. Epperly of Aspen Technology for providing test data. This research is partially supported by the Grant-in-Aid of Ministry of Science, Culture and Education of Japan.

References

[1] F. Alizadeh, (1995), "Interior point methods in semidefinite programming with applications to combinatorial optimization," *SIAM Journal on Optimization* **5**, 13–51.

[2] F. A. AL-Khayyal and J. E. Falk, (1983), "Jointly constrained biconvex programming", *Math. of Oper. Res.* **8**, 273-286.

[3] E. Balas, S. Ceria and G. Cornuéjols, (1993), "A lift-and-project cutting plane algorithm for mixed 0-1 programs," *Mathematical Programming* **58**, 295–323.

[4] R. W. Cottle, J. S. Pang and R. E. Stone, (1992), *The Linear Complementarity Problem*, Academic Press, New York.

[5] T. G .W. Epperly and R .E. Swaney, (1996), "Branch and bound for global NLP : Iterative LP algorithm and results", in I. E. Grossmann ed., *Global Optimization in Engineering Design*, Kluwer, Dordrecht.

[6] J. E. Falk and S. W. Palocsay, (1992), "Optimizing the sum of linear fractional functions", in C. A. Floudas eds., *Collection : Recent Advances in Global Optimization*, Kluwer, Dordrecht, 221-258.

[7] C. A. Floudas and P. M. Pardalos, (1990) *A Collection of Test Problems for Constrained Global Optimization Algorithms,* Lecture Notes in Computing Science, Springer, New York.

[8] T. Fujie and M. Kojima, (1997), "Semidefinite relaxation for nonconvex programs," *Journal of Global Optimization* **10**, 367–380.

[9] K. Fujisawa, M. Kojima and K. Nakata, (1995), "SDPA (SemiDefinite Programming Algorithm) - User's manual - version 4.10," Technical Report B-308, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro, Tokyo, Japan, Revised May 1998, Available at http://www.is.titech.ac.jp/pub/OpRes/software/SPDA.

[10] M. X. Goemans, (1997), "Semidefinite programming in combinatorial optimization," *Mathematical Programming* **79**, 143–161.

[11] K. C. Goh, M. G. Safonov and G. P. Papavassilopoulos, (1995), "Global optimization for the biaffine matrix inequality problem," *Journal of Global Optimization* **7**, 365-380.

[12] I. E. Grossmann and A. Kravanja, (1997), "Mixed-integer nonlinear programming : A survey of algorithms and applications," in L. T. Biegler eds., *Large-Scale Optimization with Applications, Part II : Design and Control*, Springer-Verlag, 73-100.

[13] M. Kojima and L. Tunçel, (1998), "Cones of matrices and successive convex relaxations of nonconvex sets," Technical Report B-338, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro, Tokyo, Japan. Also issued as CORR 98-6, Dept. of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.

[14] M. Kojima and L. Tunçel, (1998), "Discretization and Localization in Successive Convex Relaxation Methods for Nonconvex Quadratic Optimization Problems," Technical Report B-341, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro, Tokyo, Japan.

[15] L. Lovász and A. Schrijver, (1991), "Cones of matrices and set functions and 0-1 optimization," *SIAM J. on Optimization* **1**, 166–190.

[16] M. Mesbahi and G. P. Papavassilopoulos, (1997), "A cone programming approach to the bilinear matrix inequality problem and its geometry," *Mathematical Programming* **77**, 247–272.

[17] Yu. E. Nesterov, (1997), "Semidefinite relaxation and nonconvex quadratic optimization," CORE Discussion Paper, #9744.

[18] Yu. E. Nesterov and A. S. Nemirovskii, (1994), *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM, Philadelphia.

[19] G. Pataki and L. Tunçel, (1997), "On the generic properties of convex optimization problems in conic form," *Research Report* 97–16, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada.

[20] S. Poljak, F. Rendl and H. Wolkowicz, (1995), "A recipe for semidefinite relaxation for (0,1)-quadratic programming," *Journal of Global Optimization* **7**, 51–73.

[21] M. V. Ramana, (1993) *An algorithmic analysis of multiquadratic and semidefinite programming problems,* PhD thesis, Johns Hopkins University, Baltimore, MD.

[22] N. V. Sahinidis, (1998), "BARON : Branch and reduce optimization navigator, User's manual ver. 3.0", Department of Chemical Engineering, University of Illinois at Urbana-Champaign.

[23] N. V Sahinidis and I.E. Grossmann, (1991), "Reformation of multiperior MILP models for planning and scheduling of chemical processes", *Comp. & Chem. Eng.* **15**, 255.

[24] S. Schaible, (1995), "Fractional programming" in R. Horst and P. Pardalos eds., *Handbook of Global Optimization*, Kluwer, Dordrecht, 495-608.

[25] H. D. Sherali and A. Alameddine, (1992), "A new reformulation-linearization technique for bilinear programming problems," *Journal of Global Optimization* **2**, 379–410.

[26] H. D. Sherali and C. H. Tuncbilek, (1995), "A reformulation-convexification approach for solving nonconvex quadratic programming problems," *Journal of Global Optimization* **7**, 1–31.

[27] N. Z. Shor, (1990), "Dual quadratic estimates in polynomial and boolean programming," *Annals of Operations Research* **25**, 163-168.

[28] R. E. Swaney, (1990), "Global solution of algebraic nonlinear programs," *AIChE Annual Meeting*, Chicago.

[29] V. Visweswaran and C. A. Floudas, (1990), "A global optimization (GOP) for certain classes of nonconvex NLPs – II. Application of theory and test problems," *Computers and Chemical Engineering* **14**, 1419–1434.

[30] V. Visweswaran, C. A. Floudas, M. G. Ierapetritou and E. N. Pistikopoulos, (1996), "A decomposition-based global optimization approach for solving bilevel linear and quadratic programs," in C. A. Floudas eds., *State of the Art in Global Optimization*, Kluwer, Dordrecht, 139-162.

[31] Y. Ye, (1997), "Approximating quadratic programming with quadratic constraints," Working paper, Dept. of Management Sciences, The University of Iowa.