Regular Paper

A Multilevel Parallelized Hybrid Branch and Bound Algorithm for Quadratic Optimization

Cong Vo,† Akiko Takeda† and Masakazu Kojima†

General QOPs (quadratic optimization problems) have a linear objective function $\boldsymbol{c}^T\boldsymbol{x}$ to be maximized over a nonconvex compact feasible region F described by a finite number of quadratic inequalities. Difficulties in solving a QOP arise from the nonconvexity in its quadratic terms. We propose a branch and bound algorithm for QOPs where branching operations have been designed to effectively reduce the nonconvexity of a given QOP so that the sub-QOPs generated during branching operations become easier to solve. The bounding procedure employed in our branch and bound algorithm is a successive convex relaxation algorithm based on semidefinite programming. A significant number of semidefinite programming problems involved in the algorithm are solved in parallel using the message passing interface library for message passing. This parallel implementation enables us to solve some highly nonconvex QOPs. Message passing and multithreading are mixed to improve the performance and parallel efficiency.

1. Introduction

In the 30 years long history of global optimization the QOP has been playing a significant role because of its importance both from mathematical and application aspects. QOPs cover a lot of important nonconvex mathematical programs including 0–1 linear and quadratic integer programs, linear complementarity problems, bilevel linear and quadratic programs, and linear fractional programs. General QOPs considered in this paper can be stated in the following way:

$$\max\{\boldsymbol{c}^T\boldsymbol{x} : \boldsymbol{x} \in F\}, \tag{1}$$

where $\boldsymbol{c} \neq \boldsymbol{0}$ is the constant and $\boldsymbol{x}$ is the variable in an Euclidean space $\mathbb{R}^n$. We call $\boldsymbol{c}^T\boldsymbol{x}$ the *objective function* and $\boldsymbol{c}$ the *objective direction*. The *feasible region* F is a nonconvex compact set defined by a finite number of quadratic constraints:

$$F = \{\boldsymbol{x} \in C_0 : \ qf(\boldsymbol{x}) \leq 0 \ \forall qf \in \mathcal{P}\}, \tag{2}$$

where C_0 defined by linear inequalities is a given compact convex set including F, and $\mathcal{P}$ is a finite *quadratic representation* of F:

$$\mathcal{P} \subset \{qf(\cdot;\gamma,\boldsymbol{q},\boldsymbol{Q}) : \boldsymbol{x} \mapsto \gamma + 2\boldsymbol{q}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}\}. \tag{3}$$

A simpler instance of the general QOP which interested many researchers is a linearly constrained QOP. It consists of a quadratic objective function and a linear inequality constraint:

$$\min\{\gamma + 2\boldsymbol{q}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} : \ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}. \tag{4}$$

One can easily transform Eq. (4) into Eq. (1) by moving the quadratic objective function to the constraint part. It is well-known that even this linearly constrained QOP is NP-hard.

This paper continues the investigation of SCR (successive convex relaxation) algorithms which are *outer-approximation procedures* for general QOPs. Those algorithms were proposed by Ref. 6) and later studied in Refs. 5), 7). They are powerful algorithms to bound optimal values of QOPs. In practice, however, there are some computational limitations of SCR algorithms as mentioned in Refs. 6), 10). Numerical results of Refs. 9), 10) exhibit that their SCR algorithms consume a lot of computational time in order to achieve tighter bounding values for some QOPs. In particular, SCR algorithms generally provide neither an accurate optimal value nor an optimal solution. To overcome these limitations we incorporate an SCR algorithm into a BB (branch and bound) algorithm for computing an optimal solution, as suggested in Refs. 6), 7), 9), 10). Our BB algorithm combines effective partition-selection techniques with an improved version of the SCR algorithm [9),10)] as a bounding procedure. Heuristics are used to improve incumbents together with upper bounds in order to reduce the number of branches that would not lead to optimal solutions and to keep the program stay

† Department of Mathematical and Computing Sciences, Tokyo Institute of Technology

within the memory capacity.

Reference 10) used a client-server based parallel computing system called Ninf [8] to implement their parallel SCR algorithm. They concluded that a powerful parallel computing facility and a parallel algorithm are necessary when solving large-scale QOPs in general and highly nonconvex ones in particular.

We modified the parallel codes [10] using the MPI (message passing interface) instead of Ninf. The MPI includes a rich set of features allowing flexible and effective parallel implementations. To fit the bounding procedure into a BB framework some other modifications were also required. As a result we developed a new SCR program for QOPs which inherits the good characteristics of Ref. 10) while enjoying less communication time, higher parallel efficiency and a complete interface with the BB algorithm.

The number of SDP (semidefinite programming) problems involved in an SCR iteration on a sub-QOP born in the BB tree is usually not a divisor nor a multiple of the number of processors used. Accordingly a sequential execution of SCR iterations will let some processors be temporarily idle. Therefore we propose 2 levels of parallelism; in the lower level we prepare a procedure to execute an SCR iteration in parallel using the MPI; in the upper level we run several instances of that procedure using Pthread (POSIX thread) to concurrently execute several SCR iterations on different sub-QOPs.

A major purpose of SCR algorithms is to convexify the feasible region F or to get rid of its nonconvexity in order to get a good approximation. Reference 5) showed that the complexity bounds of SCR algorithms depend on the diameters of C_0 and F, and on the nonconvexity of F. The diameters are automatically reduced through branching operations but the nonconvexity needs a special treatment. Section 2 will introduce the concept of the nonconvexity which is also the key of the BB algorithm. We recall previous SCR algorithms and propose an improved SCR algorithm in Section 3. We present in detail sequential BB algorithms in Section 4 and parallel BB algorithms in Section 5. Numerical experiments in Section 6 show the effectiveness of our current implementation and parallel efficiency. Some future work are presented in the last section.

2. The Nonconvexity of QOPs

The nonconvexity of a constraint in (3) results from its quadratic term $\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$. With the symmetric assumption of $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$, we separate $\boldsymbol{Q} = \boldsymbol{Q}_+ + \boldsymbol{Q}_-$ using semidefinite matrices $\boldsymbol{Q}_+ \succcurlyeq \boldsymbol{0}$ and $\boldsymbol{Q}_- \preccurlyeq \boldsymbol{0}$. Let Λ_- be the set of the negative eigenvalues λ of $\boldsymbol{Q}$. We have $\boldsymbol{Q}_- = \sum_{\lambda\in\Lambda_-} \lambda \boldsymbol{p}\boldsymbol{p}^T$ where $\boldsymbol{p}$ is a unit eigenvector corresponding to a negative eigenvalue λ in Λ_-. We call $\boldsymbol{p}$ a *nonconvex direction* of this QOP. The *nonconvexity* in the quadratic term $\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ is

$$\boldsymbol{x}^T\boldsymbol{Q}_-\boldsymbol{x} = \boldsymbol{x}^T \sum_{\lambda\in\Lambda_-} \lambda \boldsymbol{p}\boldsymbol{p}^T\boldsymbol{x} = \sum_{\lambda\in\Lambda_-} \lambda(\boldsymbol{p}^T\boldsymbol{x})^2.$$

The value range of $\boldsymbol{p}^T\boldsymbol{x}\,(\boldsymbol{x}\in F)$ is exactly the diameter of the feasible region F in the direction $\boldsymbol{p}$. We call the diameter of F on its nonconvex direction $\boldsymbol{p}$, a *nonconvex diameter* of it:

$$\mathrm{diam}(F,\lambda,\boldsymbol{p}) = \sqrt{-\lambda} \sup_{\boldsymbol{x},\boldsymbol{y}\in F} \boldsymbol{p}^T(\boldsymbol{x}-\boldsymbol{y}). \quad (5)$$

We define the *maximum nonconvex diameter* $\mathrm{diam}_{\mathrm{nc}}(\mathcal{P})$ and the *sum of nonconvex diameters* $\mathrm{diam}(\mathcal{P})$ of QOP (1):

$$\mathrm{diam}_{\mathrm{nc}}(\mathcal{P}) = \max_{(\lambda,\boldsymbol{p})\in\Pi_-(\mathcal{P})} \mathrm{diam}(F,\lambda,\boldsymbol{p}), \quad (6)$$

$$\mathrm{diam}(\mathcal{P}) = \sum_{(\lambda,\boldsymbol{p})\in\Pi_-(\mathcal{P})} \mathrm{diam}(F,\lambda,\boldsymbol{p}), \quad (7)$$

$$\Pi_-(\mathcal{P}) = \bigcup_{qf(\cdot;\gamma,\boldsymbol{q},\boldsymbol{Q})\in\mathcal{P}} \Pi_-(\boldsymbol{Q}), \quad (8)$$

where $\Pi_-(\boldsymbol{Q})$ is the set of pairs of an eigenvalue and the corresponding unit eigenvector of the negative semidefinite part of the matrix $\boldsymbol{Q}$. Both $\mathrm{diam}_{\mathrm{nc}}(\mathcal{P})$ and $\mathrm{diam}(\mathcal{P})$ represent the nonconvexity of the QOP. We use $\mathrm{diam}_{\mathrm{nc}}(\mathcal{P})$ when deciding where to split the feasible region of a QOP and use $\mathrm{diam}(\mathcal{P})$ when comparing the nonconvexity of two QOPs in the BB tree. Note that since F is not available during an SCR algorithm we use a convex relaxation C instead of F when estimating the nonconvexity [Eqs. (6) and (7)].

Our BB algorithm evaluates $\mathrm{diam}_{\mathrm{nc}}(F)$ of the sub-QOPs and tries to narrow it in order to reduce the nonconvexity of their feasible regions and to make the computation easier.

3. Successive Convex Relaxation

3.1 Notation

Successive convex relaxation is a technique to

successively transform a QOP into SDPs that provide bounds for the optimal objective value. We define:

$$\begin{aligned}\alpha(C,\boldsymbol{d}) &\equiv \sup\{\boldsymbol{d}^T\boldsymbol{x} \,:\, \boldsymbol{x}\in C\},\\ lsf(\boldsymbol{x};C,\boldsymbol{d}) &\equiv \boldsymbol{d}^T\boldsymbol{x}-\alpha(C,\boldsymbol{d}),\\ r2sf(\boldsymbol{x};C,\boldsymbol{d}_0,\boldsymbol{d}) &\equiv -lsf(\boldsymbol{x};C,\boldsymbol{d}_0)\times\\ &\qquad lsf(\boldsymbol{x};C,\boldsymbol{d}),\end{aligned}$$

here $lsf(\boldsymbol{x};C,\boldsymbol{d})$ is a *linear supporting function* for C in a direction $\boldsymbol{d}$ and $r2sf(\boldsymbol{x};C,\boldsymbol{d}_0,\boldsymbol{d})$ is a *rank-2 (quadratic) supporting function* for C in a pair of directions $\boldsymbol{d}_0$ and $\boldsymbol{d}$. Let D, D_0 and D_k be sub-sets of $\mathbb{R}^n$, we define:

$$\begin{aligned}\alpha(C,D) &= \{\alpha(C,\boldsymbol{d}) \,:\, \boldsymbol{d}\in D\},\\ lsf(C,D) &= \{lsf(\cdot;C,\boldsymbol{d}) \,:\, \boldsymbol{d}\in D\},\\ r2sf(C,D_0,D_k) &= \{r2sf(\cdot;C,\boldsymbol{d}_0,\boldsymbol{d}) \,:\\ &\qquad \boldsymbol{d}_0\in D_0,\boldsymbol{d}\in D_k\}.\end{aligned}$$

We define convex relaxations of F:

$$C_{k+1}=\Bigg\{\boldsymbol{x}\in C_0 \text{ such that } \exists \boldsymbol{X}\in\mathcal{S}^n:$$

$$\begin{pmatrix}1 & \boldsymbol{x}^T\\ \boldsymbol{x} & \boldsymbol{X}\end{pmatrix}\in\mathcal{S}_+^{n+1} \text{ and}$$

$$\gamma+2\boldsymbol{q}^T\boldsymbol{x}+\boldsymbol{Q}\bullet\boldsymbol{X}\le 0\,\forall qf(\cdot;\gamma,\boldsymbol{q},\boldsymbol{Q})\in$$

$$\mathcal{P}\cup lsf(C_k,D_0)\cup r2sf(C_k,D_0,D_k)\Bigg\}, \quad (9)$$

for $k=0,1,\ldots$ where $\mathcal{S}^n$ is the set of $n\times n$ symmetric matrices, $\mathcal{S}_+^n$ is the set of $n\times n$ positive semidefinite symmetric matrices and

$\boldsymbol{Q}\bullet\boldsymbol{X}=\sum_{i=1}^n\sum_{j=1}^n Q_{ij}X_{ij}$.

3.2 Previous SCR Algorithms

Reference 6) established a theoretical framework of SCR algorithms that lead to the convex hull of the feasible region F of a given QOP. They proposed *discretization*s of these algorithms which require solutions of a finite number of SDPs in each major iteration in Ref. 7). In the same paper the *localization* technique was developed for generating convex sets that approximate F accurately only for directions that lie in a neighborhood of the objective direction $\boldsymbol{c}$. Conceptual SCR algorithms [6] and localization SCR algorithms [7] adopt $D_0=\{\pm\boldsymbol{e}_1,\pm\boldsymbol{e}_2,\ldots\pm\boldsymbol{e}_n\}$ and various kinds of unit direction sets D_k to guarantee that solving convex optimization problems $\alpha(C_k,\boldsymbol{c})$ gives a non-increasing sequence of real numbers ζ_k converging to the optimum value of QOP (1).

Nevertheless, these SCR algorithms [6),7)] are impractical as they require solutions of infinitely many or potentially a very large number of SDPs. Reference 9) presented implementable SCR algorithms which dramatically reduces the number of SDPs involved in the relaxation procedures by employing finite direction sets:

$$D_0=\{\pm\boldsymbol{p}:\exists\lambda\in\mathbb{R},(\lambda,\boldsymbol{p})\in\Pi_-(\mathcal{P})\},$$

$$D_k(\theta)=\left\{\frac{\boldsymbol{c}\cos\theta+\boldsymbol{p}\sin\theta}{\|\boldsymbol{c}\cos\theta+\boldsymbol{p}\sin\theta\|}:\boldsymbol{p}\in D_0\right\} \quad (10)$$

$$E_k=D_0\cup D_k\cup\{\boldsymbol{c}\}. \quad (11)$$

Some numerical results of these algorithms were reported in Refs. 9), 10). Note that accurate bounds are no longer guaranteed in these algorithms.

We show parallel SCR algorithms proposed in 10) in the following. First for the master (called client in Ref. 10)):

Algorithm 1 On the master processor:

(1) Compute D_0. Set $k=1$ and $C_k=C_0$.
(2) Compute D_k with some value $\theta=\theta_k$. Assign each $\boldsymbol{d}\in E_k$ (11) to an idle worker among n_cpu worker processors. Send $\boldsymbol{d}$ and $\alpha(C_{k-1},E_{k-1})$ to the worker.
(3) Receive results $\alpha(C_k,\boldsymbol{d})\in\alpha(C_k,E_k)$ from workers.
(4) Find an upper bound ζ_k for the maximum objective function value of QOP (1): $\zeta_k=\alpha(C_k,\boldsymbol{c})$ among $\alpha(C_k,E_k)$ received. If ζ_k satisfies some termination criteria then stop.
(5) Let $k\leftarrow k+1$ and go to (2).

and for workers (called server in 10)):

Algorithm 2 On a worker processor among n_cpu worker processors:

(1) Compute D_0. Let $k=1$ and $C_k=C_0$.
(2) Receive data of a vector $\boldsymbol{d}\in E_k$ and $\alpha(C_{k-1},E_{k-1})$.
(3) Generate C_k using the data received.
(4) Compute $\alpha(C_k,\boldsymbol{d})$ by *solving an SDP*. Return the value to the master.
(5) Let $k\leftarrow k+1$ and go to (2).

The value of θ_k is modified when the decreasing speed of ζ_k is smaller than a specific parameter in the algorithms above. Reference 10) use

$$\theta_k=\frac{4\pi}{9\times 2^{k'}}\quad (k>k') \quad (12)$$

and increase $k'=0,1,2,\ldots,K$ to decrease θ_k through the algorithm. Some parameters are used to decide how to increase k'. They set

$K = 3$ because larger K does not induce clearly better bounds ζ_k.

Definition 1 We call the data set θ_k, E_{k-1} and $\alpha(C_{k-1}, E_{k-1})$ the *boundary data* of the QOP in iteration k of Algorithm 1.

3.3 An improved SCR algorithm

Here we describe modifications for an improved parallel SCR algorithm which is used in our BB framework. We set

$$\theta_k = \frac{4\pi}{9 \times 2^{k'\%7}} \quad (k > k') \tag{13}$$

to turn around the value of θ_k ($k'\%7$ means $k' \bmod 7$). It has been verified in practice that Eq. (13) is better than Eq. (12): the program is more stable and gradually reduces ζ_k even when k is about some hundreds.

We introduce a rule to automatically assign directions $\boldsymbol{d}$ in E_k to worker processors. For details see Section 5. This rule eliminates sending data of directions $\boldsymbol{d}$ between the master and workers as done in Ref. 10). The boundary data is sent from the master to all workers using a tree-based algorithm MPI_Bcast with a communication cost $O(\log n_w)$ where n_w is the number of workers. The previous algorithm [10] takes $O(n_w)$ for this operation.

We do not change θ_k when executing SCR iterations on a node but change θ_k when moving from a node in the BB tree to its sub-QOPs. SCR iterations on a sub-QOP start with an $\alpha(C_k, E_k)$ inherited from its father. In each SCR iteration the feasibility of an sub-QOP is checked and the upper bound value ζ_k obtained is compared with the incumbent optimum of the BB tree to avoid the computation of sub-QOPs that would not induce good bounds.

4. A Branch-and-bound Algorithm

Definition 2 Data of a node in the BB tree consist of

- its *id* and the id of its father *mid* in the tree; this is to analyze the structure of the BB tree;
- *diam*: an estimation of its nonconvexity;
- *feas*: the feasibility of its father;
- *up*: an *local upper bound* value (an upper bound value of its father);
- its boundary data.

The common data for all nodes include the input file name and parameters for the SCR algorithm. A node's private data is computed from the data of its father through the splitting operation. When a node gets its turn to be computed, the data will be used to prepare a QOP to be passed to the SCR algorithm.

Note 1 From now on we use just "QOP" to indicate a sub-QOP in a BB tree.

4.1 Splitting a QOP

If we can not discard a QOP, we find its largest nonconvex diameter (6) and then insert a cutting plane through the middle point of the diameter. The two new QOPs inherit the feasibility *feas* and the upper bound *up* from their father. Their nonconvexity *diam* are estimated by subtracting a half of the father's largest nonconvex diameter (6) from the sum of the nonconvex diameters (7) on the father.

As a classical BB algorithm what we want to do is:

(1) Find feasible solutions as they help to improve the incumbent and to delete nodes that would not provide optimal solutions.
(2) Reduce the (maximum) upper-bound of nodes in order to reduce the upper-bound of the BB tree.
(3) Reduce the (maximum) nonconvex diameter of nodes as it makes the QOPs easier.
(4) Delete unnecessary nodes as soon as possible to save memory storage.

Items (1) and (2) have to be done together in order to reduce the number of branches that would not lead to optimal solutions. Item (4) is critical due to memory limitation.

4.2 Selecting the Next QOP

The QOPs are kept in two priority queues DiamQ and BoundQ for the purpose of improving the incumbent and upper bounds. In both DiamQ and BoundQ, a QOP with a larger upper bound value is considered better. However from DiamQ we want to induce feasible solutions so a possibly easier QOP with a smaller nonconvex diameter or better feasibility is solved first even if its upper bound value is small. In contrast, from BoundQ we want to find out and solve harder QOPs with larger upper bound values so that ones with larger nonconvex diameter or worse feasibility (24) will be solved first.

We have two computation phases where the focus of computational efforts is different. In the following algorithms in this section *bb_up* indicates the upper bound value of the BB tree and *bb_opt* indicates the incumbent value. Whenever bb_opt or bb_up is updated, we assign the changes to *bb_opt_speed* or *bb_up_speed*, respectively. They are used in a heuristic algorithm to decide a limit on the number of iter-

ations in Algorithm 4 which reduces the upper bound and Algorithm 5 which improves the incumbent. If bb_up decreases faster (respectively slower) than bb_opt increases we will spend more processor time to "decrease bb_up" (respectively "increase bb_opt"). The next heuristic algorithm decide limits S and T used in Algorithm 4 and Algorithm 5 respectively.

Algorithm 3 The number of iterations S (T)

(1) If bb_opt_speed and bb_up_speed are both very small, set $S \leftarrow V$ ($T \leftarrow V$). We initialize $V \leftarrow 10$ and increase it little by little.

(2) Else if bb_opt_speed $<$ ($>$) bb_up_speed, increase S (T) to the total number of children. This number gradually gets larger.

(3) Otherwise reduce $S \leftarrow S/2$ ($T \leftarrow T/2$).

4.3 Reducing the Upper Bound Value

Here we focus on refining the upper bound of all nodes in the BB tree. When solving concurrently several QOPs, we do not update bb_up but use the upper bound of the top node to guess it.

We define *bb_gap* and *r.bb_gap*:

$$\text{bb\_ gap} = \text{bb\_ up} - \text{bb\_ opt}, \tag{14}$$

$$\text{r.bb\_ gap} = \frac{\text{bb\_ gap}}{\max\{1, \|\text{bb\_ up}\|\}}. \tag{15}$$

The algorithms will make r.bb_gap $\leq$ *bb_acc* – a required accuracy.

Algorithm 4 To reduce the upper bound: First we pick from DiamQ all nodes with local upper bounds larger than the (global) incumbent of the BB tree and move them to BoundQ. Set $s \leftarrow 0$, $G \leftarrow$ (bb_up - bb_opt) $\times$ (1 − bb_acc) then *repeat* the following:

(1) Set $s \leftarrow s + 1$. Get a top node p out of BoundQ.

(2) Terminate the algorithm if:
- (a) either BoundQ is empty,
- (b) or $G >$ p.up − bb_opt,
- (c) or $s > S$.

(3) Execute SCR iterations on p and consider the solutions of p to update bb_opt or branch p if necessary.

Actually this algorithm is a kind of BeFS (*best first search strategy*). In theory we can run it to process only *critical* QOPs until an optimal solution has been found and then no superfluous bound calculations take place. However memory problems arise if the number of critical QOPs becomes too large. That is why we need heuristic Algorithm 3 to get a limit for the number of iterations.

4.4 Improving the Incumbent

Here we concentrate on nodes which have small diameters and the feasibility of solutions of whose fathers is small.

Algorithm 5 To improve the incumbent: At first DiamQ is empty.

(1) Let p be the top node of BoundQ, we *update* bb_up here:

$$\text{bb\_ up} \leftarrow \text{p.up}. \tag{16}$$

If bb_up $>$ bb_opt, push p into DiamQ, otherwise we stop the program. Set $G \leftarrow$ (bb_up − bb_opt) $\times$ (1 − bb_acc) and initialize $t \leftarrow 0$ then *repeat* the following steps:

(2) Set $t \leftarrow t + 1$. Get the top node p of DiamQ. If p.up $>$ bb_opt, execute SCR iterations on p then consider the solutions of p to update bb_opt or branch p if necessary.

(3) Terminate the algorithm if:
- (a) either $G >$ bb_up − bb_opt,
- (b) or $t > T$.

(4) If $G <$ bb_up − bb_opt but DiamQ is empty and BoundQ is not empty then go to (1) to take a new node p from BoundQ.

This algorithm is a kind of DFS (*depth first search strategy*). Here an active QOP with smallest nonconvex diameter and most probably standing at one of deepest levels in the search tree is chosen for exploration. Hence the number of QOPs grows slowly and the memory requirement is usually a manageable number. As the smallest nonconvex diameter gets small enough, a feasible solution is found and the incumbent bb_opt may be updated. The new incumbent is used to delete unnecessary QOPs in the BB tree. So our BB algorithm is a hybrid one combining BeFS and DFS strategies.

5. Parallel Algorithms

Figure 1 illustrates the 2 levels of parallelization. We choose Pthread for concurrent computing on a (possibly multi-processors) master computer and MPI for parallel computing on a cluster which consists of up to hundreds of processors. We are going to solve some QOPs concurrently on an upper parallel level and a lot of SDPs involved from them on a lower parallel level to improve the speedup and the parallel efficiency.

We assign the same number of SDPs to each

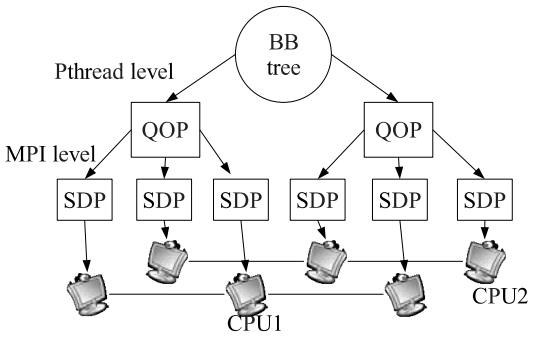

Fig. 1 Two parallel levels.

processor in the cluster in order to improve load balance among the processors and reduce their idle time, assuming that the cluster is homogeneous and that:

Claim 1 Computational time for solving every SDP is almost the same.

Let the number of available *worker* processors be $n_{cpu} \geq 1$ and the number of SDPs involved in each SCR iteration to a QOP be $n_{sdp} \gg 1$. Let the number of QOPs being solved concurrently on the master be $n_{qop} \geq 1$, and the number of SDPs solved by the workers $n_{solved} \geq 0$. We solve $n_{qop} \times n_{sdp} - n_{solved}$ SDPs in parallel, hence processors get idling one by one when

$$n_{qop} \times n_{sdp} - n_{solved} < n_{cpu}, \tag{17}$$

so that we start solving the next QOP in the BB tree.

5.1 Identifying the SDPs

To make the parallel algorithm work correctly, the master and the workers have to keep common information about which SDPs each worker solves; the SDPs were born in which SCR iteration of which QOPs. Also, the total number of SDPs is very large so that an *automatic* way of keeping the above information without transferring identities of the SDPs between the master and the workers is desired.

We identify the SDPs involved in the BB algorithm in the following way: Let each SCR iteration on the master get a unique id $q = 0, 1, 2, \ldots$ and each SDP involved in some SCR iteration get an id $s = 0, 1, 2, \ldots, n_{sdp}-1$. Then the unique id for an SDP in the BB algorithm is:

$$t = s + q \times n_{sdp}. \tag{18}$$

5.2 Assigning SDPs to processors

Now on a processor $c = 0, 1, 2, \ldots, n_{cpu} - 1$, we compute a pair q, s to identify an SDP to be solved using a simple rule:

$$\begin{cases} t &= k_c \times n_{cpu} + c, \\ q &= t/n_{sdp}, \\ s &= t\%n_{sdp}, \end{cases} \tag{19}$$

where $k_c = 0, 1, 2, \ldots$ indicates the number of SDPs solved on c. It is clear that:

$$k_c = t/n_{cpu}, \tag{20}$$

and that the workers who take part in the computing of the SCR iteration q are:

$$c = t\%n_{cpu}, \quad (t \in [t_0, t_0 + n_{sdp} - 1]), \tag{21}$$

where $t_0 = q \times n_{sdp}$. The master will *broadcast* the boundary data of the SCR iteration q to a communication group of these workers [Eq. (21)] using a tree-based algorithm MPI_Bcast with a communication cost $O(\log n_{\_cpu})$ where $n_{\_cpu}$ is the number of the group workers.

The master keeps the information indicating corresponding SCR iterations q, SDPs s and workers c, so it is able to correctly handle the SDP results returned from the workers. On the other hand every worker c updates its private t, q, s in order to know when it should receive the boundary data of a new SCR iteration q and to know which SDPs it is assigned to solve.

5.3 Algorithms

This subsession shows some concrete algorithms based on the concepts above.

Algorithm 6 A *thread* on the master making new *threads* of execution of solving QOPs:
First initialize
$k_0, k_1, k_2, \ldots, k_{n_{cpu}-1}, n_{qop}, n_{solved}$ all by 0.
While the BB tree is not empty *repeat* the following:

(1) Select a QOP, *wait* until the condition [Eq. (17)] becomes true then let $n_{qop} \leftarrow n_{qop} + 1$.
(2) *Start* a new thread of execution
  (a) Perform SCR iterations (Algorithm[7]) on the QOP.
  (b) Evaluate the solution of the QOP, delete or branch it to make new QOPs.

The "while" loop does not wait for (2) but immediately goes to (1).

Algorithm 7 Execution of an SCR iteration on the master:

(1) Get a new unique SCR id $q \leftarrow q + 1$.
(2) *Broadcast* the current *boundary data* of the corresponding QOP to the corresponding group of workers [Eq. (21)].
(3) *Wait* until all needed SDPs' results have

Table 1 Test problems.

No	Problem	n	m	QC	Local	SDP
1	BLevel3_6Lb	19	25	10	4	129
2	BLevel8_3Lb	21	22	10	4	137
3	LC30_162	31	46	1	162	143
4	LC30_36	31	46	1	36	143
5	LC40_6	41	61	1	6	199
6	LC40_72	41	61	1	72	199
7	LC60_72	61	91	1	72	299

been collected via Algorithm 8.

(4) After finishing an SCR iteration and deciding to execute the next SCR iteration we modify the number of QOPs solved:

$$n_{solved} \leftarrow n_{solved} - n_{sdp}. \tag{22}$$

Algorithm 8 A *thread* on the master to receive SDP results from the workers:
Repeat:

(1) *Receive* an SDP's result (a floating number) α from a worker c. Set

$$n_{solved} \leftarrow n_{solved} + 1. \tag{23}$$

(2) Derive t, q, s from (19) to know which SDP is corresponding to this α. After saving α let $k_c \leftarrow k_c + 1$.

(3) *Notify* (3) in the thread executing an SCR iteration q in Algorithm 7 when the number of SDPs' results for q is sufficient.

(4) *Notify* (1) in Algorithm 6 about checking condition [Eq. (17)] if necessary.

Algorithm 9 A loop on the worker c solving SDPs:
Set $k_c \leftarrow 0$, $\_q \leftarrow -1$ then *repeat*:

(1) Derive t, q, s from Eq. (19).

(2) If $\_q \neq q$:
  (a) If $\_q \geq 0$ *send* SDPs' results to the master.
  (b) *Receive* the *boundary data* of the SCR iteration q from the master and let $\_q \leftarrow q$.

(3) After solving SDP s let $k_c \leftarrow k_c + 1$.

6. Computational Experiments

The notations $1L$ and $2L$ in this section indicate 1 level parallelization and 2 levels parallelization, respectively.

6.1 Test Problems

We use some linearly constrained QOPs and bilevel QOPs [2),3)] (LC and BLevel problems in **Table 1**, respectively) to test the algorithm. The columns "n" and "m" respectively denote the number of variables and the number of constraints (not including box constraints) of the transformed problem. The column "QC" denotes the number of quadratic constraints among m constraints. The column "Local" gives the number of local optima. The column "SDP" gives the number of SDPs involved in an SCR iteration on a sub-QOP. The optimal values of all test problems are known in advance.

We use a cluster consisting of 256 dual Athlon nodes (Cluster A):

CPU Athlon MP 1900+ (1,600 MHz) × 2
Mem 768 MB
NIC Myricom Myrinet 2000
Software Linux 2.4.19, glibc-2.2.5-11.5, gcc-3.3.1, LAM/MPI-7.0.3, boost-1.30.2, SDPA-6.0

Only 214 processors are available in our numerical experiments. From now on, nC indicates that the number of processors is n.

6.2 Algorithmic Options

We describe some major parameters of the BB algorithm.

Feasibility

The feasibility of a solution $\boldsymbol{x}$ is:

$$\max_{qf \in \mathcal{P}} \frac{\max\{0, qf(\boldsymbol{x}; \gamma, \boldsymbol{q}, \boldsymbol{Q})\}}{\max\{1, \|\boldsymbol{q}\|, \|\boldsymbol{Q}\|\}}; \tag{24}$$

$\boldsymbol{x}$ is considered feasible if its feasibility is smaller than a given positive parameter.

Optimality

The BB algorithm will be terminated when r.bb_gap [Eq. (15)] is smaller than a given tolerance bb_acc.

Changing θ_k

There is a parameter to control how many time we change θ_k on each node. However, experiments show that fixing the value of this parameter to 1 usually gives good results.

6.3 Numerical results

We analyse numerical results of $1L$ in which multiple SDPs induced from a QOP are simultaneously solved in the lower level but only one QOP is processed at a time in the upper level, to examine characteristics of BB algorithms. Numerical results of $2L$ in which several QOPs are processed concurrently are given in Section 6.4 to show its advantages over $1L$. The program was implemented using C++ with gcc, SDPA 4), LAM/MPI 1) and Boost.Threads.

We set the parameters: feasibility [Eq. (24)] ≤ 0.01 and gap [Eq. (15)] ≤ 0.01. The execution time (real time) is measured by a wall clock on the master processor side.

Table 2 shows the successful results when we give the same priority to both DFS and BeFS.

Table 2 Hybrid, Cluster A/1L.

No	born	solved	SCR
1	647	352	354
2	18417	9688	9690
3	18953	13708	13723
4	287	229	234
5	269	156	161
6	1893	1361	1366
7	871	708	713

Table 3 Execution time (seconds), Cluster A/1L.

No	Hybrid	DFS	BeFS
1	85	91	86
2	2699	5382	2592
3	9256	12558	9960
4	147	432	243
5	300	266	638
6	2584	7612	3764
7	8736	12062	17049

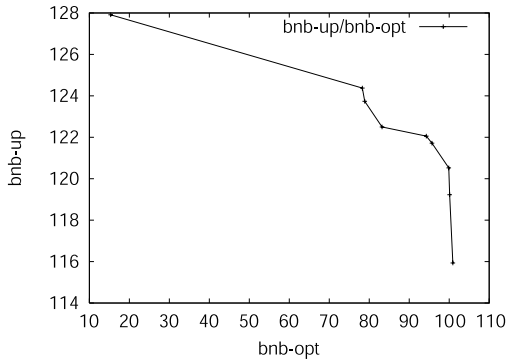

Fig. 2 bb_up / bb_opt (LC30_162).

The column "No" shows the id of the problems as in Table 1. The columns "born" and "solved" indicate the number of QOPs born and solved in the BB tree respectively. The column "SCR" gives the number of SCR iterations completed. The last column gives the wall time that passed on the master processor.

Table 3 shows good performance of the hybrid algorithm in comparing with DFS and BeFS. Superiority of the hybrid algorithm over DFS suggests that good incumbents are relatively easier to be attained while upper bounds demand more time to be reduced. This fact is also illustrated in **Fig. 2** where the upper bound makes larger changes than the incumbent. Note that the figures in this section plot points where the upper bound and the incumbent of the BB tree are changed. Superiority of the hybrid algorithm over BeFS lies in better incumbents which can discard QOPs with worse upper bounds.

We observe from the details of numerical results that as a consequence of DFS, the number of active QOPs increases in an almost linear way depending on the number of QOPs solved. This fact appears in the relation between the number of nodes born and the number of nodes solved in Table 2. Finally, **Fig. 3** shows that bb_gap decreases steadily when the number QOPs increases.

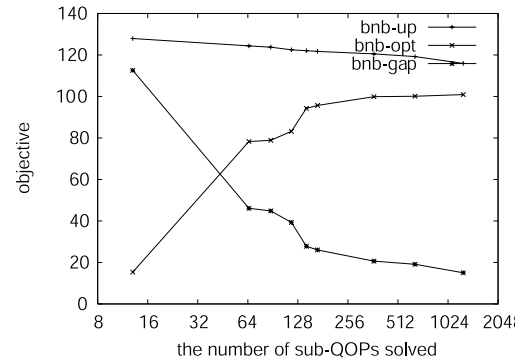

Fig. 3 Objective value / #solved (LC30_162).

Table 4 Variations in CPU time (%), Cluster A/1L.

No	naive	Hybrid	DFS	BeFS
1	10.37	1.05	0.94	1.12
2	22.48	0.39	0.45	0.43
3	22.70	0.38	0.37	0.37
4	27.18	0.59	0.53	0.45
5	29.53	3.10	2.82	2.43
6	29.20	2.79	1.81	1.81
7	17.27	0.87	0.91	0.88

6.4 Parallel efficiency

This subsection shows that load balance is very good even when using just 1L. However, in terms of speedup and idle time, we see that 2L is apparently better than 1L.

Load balance

Table 4 shows the *coefficient of variation* of the CPU time used for solving SDPs on the worker processors, using 1L. The column "naive" shows results of a naive parallel algorithm (ran on 130C) in which the master communicates directly to every worker without using algorithms described in Section 5; therefore the coefficient of variation of the CPU time get considerably large. The columns "Hybrid", "DFS" and "BeFS" show results of our latest implementation of the algorithms proposed (ran on 214C), in particular the rule discribed in Section 5.2 to assign SDPs to processors and to broadcast the boundary data; not surprisingly this implementation largely improves the load balance. *These results support Claim 1.* We have verified that 2L shows slightly better

Table 5 Relative speedup, Cluster A/1L.

CPU	ser	2C	4C	8C	16C
seconds	5445	5480	1855	831	410
speedup	0.99	1	2.9	6.5	13.3
CPU	32C	64C	128C	130C	214C
seconds	232	146	120	84	85
speedup	23.6	37.5	45.6	65.2	64.4

load balance .

Speedup

Relative speedup when solving BLevel3_6Lb on Cluster A is shown in **Table 5**. The column "ser" shows the result of a serial implementation. Note that 1 processor is used for the master. The speedup depends linearly on the number of processors when it increases from 2 to 128. However, with 130 processors available, 129 processors are used as workers and 129 is exactly the number of SDPs in each SCR iteration (see BLevel3_6Lb in Table 1), explaining the *big gap* between the speedup of 128 and 130 processors. The algorithms in Section 5 were proposed to reduce this "big gap" as well as to make more improvement in the speedup when the number of available processors gets larger than the number of SDPs involved in each SCR iteration (compare the speedup of 130 and 214 processors in Table 5).

Our current implementation of 2L does not run on Cluster A properly, due to some compatiblity problems of glibc, gcc, LAM/MPI and Boost.Threads. However, it runs perfectly on another cluster consisting of 40 dual Athlon nodes (Cluster B):

CPU Athlon MP 2400+ (2 GHz) × 2
Mem 1 GB–2 GB
NIC 1000BASE-T × 2
Software Linux 2.4.22, glibc-2.2.5-34, gcc-3.3.2, LAM/MPI-7.0.0, boost-1.30.2, SDPA-6.0

Table 6 shows that 2L is better than 1L in terms of the execution (real) time, the speedup and the idle time of the master. Let the execution time be M and the mean of workers CPU time W, we compute the idle time as $(M - W)/M$. We observe that the execution time and the speedup of 1L in the 2nd and 4th columns do not improve smoothly, while those of 2L in the 3rd and 5th columns do. The simple reason is, considering the cases of using 42, 43, 44, 45 processors, we see that only when using 44 processors, the number of SDPs in an SCR iteration (= 129, see BLevel3_6Lb in Table 1) is a multiple of the number of workers

Table 6 Cluster B, 2L vs 1L.

	Real		Speedup		Idle (%)	
CPU	1L	2L	1L	2L	1L	2L
ser	4740		0.98	0.98		
2C	4850	4825	1	1	0.0	0.0
4C	1641	1636	3.0	2.9	1.2	1.2
8C	733	666	6.6	7.2	5.2	5.1
16C	357	352	13.6	13.7	9.4	9.2
32C	202	178	24.0	27.1	21.7	12.7
42C	165	137	29.4	35.2	27.7	13.5
43C	163	130	29.8	37.1	28.5	11.1
44C	133	124	36.5	38.9	14.1	8.7
45C	133	121	36.5	39.9	16.4	9.1
64C	124	87	39.1	55.5	37.1	11.2
80C	93	78	52.2	61.9	32.9	21.1

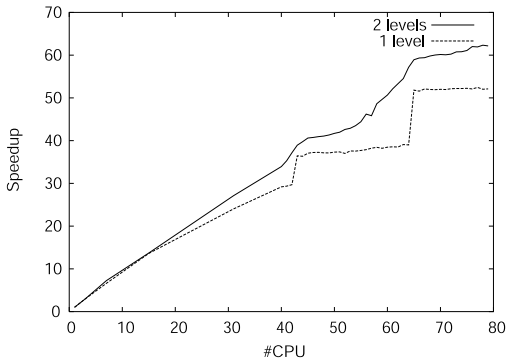

Fig. 4 Scability of 2L vs 1L.

(= 43), and the same number of SDPs are assigned to each worker, hence a worker is not forced to wait other workers so long after finishing its own computing.

We conclude that in general cases, 2L has exhausted every single processor of the cluster while 1L has not. **Figure 4** illustrates the speedup of 2L more linearly and nearer to the ideal speedup, compared to 1L.

The idle time of the master tends to become larger as we use more worker processors, because there is some delay before a worker can make peer-to-peer connections to the master in order to send SDPs' results, and because a worker can begin processing new SDPs only after completing of sending the results to the master. This bottleneck may improve by gathering SDPs' results before sending; and/or preparing two threads on each worker, one for receiving boundary data and computing SDPs, and the other for sending SDPs' results.

7. Concluding Remarks

Our parallel BB algorithm successfully solves some highly nonconvex QOPs taking account of memory management and load balancing prob-

lems. We define a measure for the nonconvexity of the QOP, show how to estimate it on the sub-QOPs and apply it fruitfully to the splitting operation. Our algorithm combining BeFS and DFS improves lower and upper bound simultaneously, and a heuristic algorithm is used to avoid superfluous computation in efforts to refine one bound.

We propose and implement 2 levels parallelization to handle several sub-QOPs concurrently on the master processor and solve a significant number of corresponding SDPs on the worker processors in order to achieve high speedup and parallel efficiency.

Solving more complicated highly nonconvex QOPs requires more processor power. Techniques to connect and utilize powerful computing facilities like a mixture of grid computing and cluster computing are required.

Acknowledgments The authors would like to thank Professor Satoshi Matsuoka of Tokyo Institute of Technology and Professor Katsuki Fujisawa of Tokyo Denki University for letting us use their laboratories advanced PC clusters. We are grateful to anonymous referees for many suggestions to improve the presentation of the paper.

References

1) Burns, G., Daoud, R. and Vaigl, J.: LAM: An Open Cluster Environment for MPI, *Proc. Supercomputing Symposium*, pp.379–386 (1994).
2) Calamai, P.H. and Vicente, L.N.: Generating Quadratic Bilevel Programming Problems, *ACM Trans. Math. Softw.*, Vol.20, pp.102–122 (1994).
3) Calamai, P.H., Vicente, L.N. and Judice, J.J.: A New Technique for Generating Quadratic Programming Test Problems, *Math. Program.*, Vol.61, pp.215–231 (1993).
4) Fujisawa, K., Kojima, M., Nakata, K. and Yamashita, M.: SDPA (SemiDefinite Programming Algorithm) User's Manual – Version 6.00 (2002).
5) Kojima, M. and Takeda, A.: Complexity Analysis of Conceptual Successive Convex Relaxation Methods for Nonconvex Sets, *Mathematics of Operations Research*, Vol.26, No.3, pp.519–542 (2001).
6) Kojima, M. and Tunçel, L.: Cones of matrices and successive convex relaxations of nonconvex sets, *SIAM Journal on Optimization*, Vol.10, No.3, pp.750–778 (2000).
7) Kojima, M. and Tunçel, L.: Discretization and Localization in Successive Convex Relaxation Methods for Nonconvex Quadratic Optimization Problems, *Math. Program.*, Vol.89, No.1, pp.79–111 (2000).
8) Sato, M., Nakada, H., Sekiguchi, S., Matsuoka, S., Nagashima, U. and Takagi, H.: Ninf: A Network Based Information Library for Global World-Wide Computing Infrastructure, *HPCN Europe*, pp.491–502 (1997).
9) Takeda, A., Dai, Y., Fukuda, M. and Kojima, M.: Towards the Implemenation of Successive Convex Relaxation Methods for Nonconvex Quadratic Optimization Problems, *Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems*, Pardalos, P.M.(ed.), pp.489–510, Kluwer Academic Publisher (2000).
10) Takeda, A., Fujisawa, K., Fukaya, Y. and Kojima, M.: Parallel Implementation of Successive Convex Relaxation Methods for Quadratic Optimization Problems, *Journal of Global Optimization*, Vol.24, No.2, pp.237–260 (2002).

(Received October 10, 2003)
(Accepted January 29, 2004)

Cong Vo was born in 1974. He received his M. Sci. from Tokyo Institute of Technology in 2003. He has been a Ph.D. student of the same institute since 2003. His major research subject is mathematical programming including global and combinatorial optimization. He is a member of ORSJ.

Akiko Takeda was born in 1973. She received her M. Eng. from Keio University in 1998 and Dr. Sci. from Tokyo Institute of Technology in 2001. She had worked in Toshiba Corporation since 2001 and had engaged in mathematical modeling. Since 2003 she has been in Tokyo Institute of Technology as a research assistant. Her current research interest is parallel computing on optimization problems and systems of polynomial equations. She is a member of ORSJ.

Masakazu Kojima was born in 1947. He received his M. Eng. and Dr. Eng. from Keio University in 1971 and 1974, respectively. He has been working in Tokyo Institute of Technology since 1975 as an assistant professor (1975–1979), an associate professor (1979–1989) and a professor (1989–). His major research subject is mathematical programming including linear, nonlinear and combinatorial optimization. He received INFORMS Computing Society Prize and Frederick W. Lanchester Prize in 1992. He was selected as a highly cited researcher in mathematics by Institute for Scientific Information in 2003. He is a member of MPS, SIAM, ORSJ and JSIAM.