Mathematical Programming 42 (1988) 421-448
North-Holland



# A PARALLEL ALGORITHM FOR CONSTRAINED CONCAVE QUADRATIC GLOBAL MINIMIZATION

A.T. PHILLIPS and J.B. ROSEN

*Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, USA*

Received 12 August 1987
Revised manuscript received 29 February 1988

The global minimization of large-scale concave quadratic problems over a bounded polyhedral set using a parallel branch and bound approach is considered. The objective function consists of both a concave part (nonlinear variables) and a strictly linear part, which are coupled by the linear constraints. These large-scale problems are characterized by having the number of linear variables much greater than the number of nonlinear variables. A linear underestimating function to the concave part of the objective is easily constructed and minimized over the feasible domain to get both upper and lower bounds on the global minimum function value. At each minor iteration of the algorithm, the feasible domain is divided into subregions and linear underestimating problems over each subregion are solved in parallel. Branch and bound techniques can then be used to eliminate parts of the feasible domain from consideration and improve the upper and lower bounds. It is shown that the algorithm guarantees that a solution is obtained to within any specified tolerance in a finite number of steps. Computational results are presented for problems with 25 and 50 nonlinear variables and up to 400 linear variables. These results were obtained on a four processor CRAY2 using both sequential and parallel implementations of the algorithm. The average parallel solution time was approximately 15 seconds for problems with 400 linear variables and a relative tolerance of 0.001. For a relative tolerance of 0.1, the average computation time appears to increase only linearly with the number of linear variables.

*Key words*: Global minimization, concave quadratic programming, parallel algorithms.

## 1. Introduction

The general problem considered is that of minimizing a concave quadratic function over a bounded polyhedral set. Specifically, the problem statement is

$$(\mathrm{GQ}) \qquad \operatorname*{global\,min}_{(x,y)\in\Omega} \psi(x, y) = \varphi(x) + d^t y$$

where $\Omega = \{(x, y): A_1x + A_2y \leq b, y \geq 0\}$ is a nonempty, bounded polyhedral set, and $x \in \mathbb{R}^n$, $y \in \mathbb{R}^k$, $A_1 \in \mathbb{R}^{m\times n}$, $A_2 \in \mathbb{R}^{m\times k}$, $b \in \mathbb{R}^m$, and $d \in \mathbb{R}^k$. The usual nonnegativity requirement on $x$ is assumed to be included in the constraints of $\Omega$. The nonlinear term $\varphi(x)$ of $\psi(x, y)$ can be expressed as $\varphi(x) = c^t x - (\frac{1}{2})x^t Qx$, where $c \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n\times n}$. The matrix $Q$ is assumed to be a positive semi-definite symmetric matrix since $\psi(x, y)$ is a concave quadratic function. Large-scale problems of this type are characterized by having many more linear variables than nonlinear variables ($k \gg n$).

The problem GQ is a constrained combinatorial optimization problem and is equivalent to other well known combinatorial optimization problems such as zero-one integer programming and the quadratic assignment problem (Pardalos and

Rosen, 1987). It follows from this equivalence that problem GQ is NP-hard. From a computational viewpoint, this means that, in the worst case, the computing time will grow exponentially with the number of nonlinear variables. An important property of this problem, which is basic to all solution methods, is that the global minimum point is always found at a vertex of the convex polytope $\Omega$. For this reason, linear programming is an essential part of any computational algorithm to solve problem GQ. For a comprehensive review of constrained global optimization, including applications and recent computational results, see Pardalos and Rosen (1986) and (1987).

In this paper a new algorithm is presented which is designed to be efficient for problems with many linear variables. It is also designed so that it can be easily implemented for, and take full advantage of, parallel processing. The algorithm is guaranteed to obtain an $\varepsilon$-approximate solution (the relative error in the objective function is bounded by a user specified tolerance $\varepsilon$) in a finite number of steps. This guaranteed bound (on the number of subproblems solved) is obtained by a worst case analysis and grows exponentially as $(4n/\varepsilon)^{n/2}$. Obviously if this represented the average computational performance of the algorithm, it would be of no practical interest. Thus, extensive computational testing of the algorithm was carried out on a range of realistic test problems. These problems ranged in size from $n = 25$ and $k = 0$ up to $n = 50$ and $k = 400$. The largest problems are an order of magnitude larger than any others of this kind previously reported in the literature. The computation times for $\varepsilon = 0.001$ using a four processor CRAY2 ranged from less than one second to a maximum of 165 seconds. The average (ten problems) parallel solution time for $n = 25$ and $k = 400$ was approximately 15 seconds. These results demonstrate that the algorithm and its parallel implementation are practical for the solution of constrained quadratic global minimization problems of the size tested, and probably for substantially larger problems as well.

The algorithm is an efficient method for two major reasons. First, at each major iteration of the method, a heuristic step (stage I) is applied in an attempt to eliminate parts of the feasible region which cannot contain the global minimum vertex. In the worst case, this step will fail to eliminate any regions. But in practice the heuristic works extremely well at every major iteration of the solution procedure, and hence the original feasible region is rapidly reduced to a much smaller polytope in which the global minimum vertex must occur. Second, the parallel implementation of the algorithm is highly efficient. At each minor iteration, $2n$ multiple-cost-row linear programs are available to be solved in parallel. Since the solution of the linear programs is the most computationally intensive part of the method, all processors can be efficiently utilized, and in fact speedups approaching and even exceeding the number processors are observed.

The approach presented here for solving problem GQ takes full advantage of the linearity of the $y$ variables. The original ideas for this approach were proposed by Rosen (1983) for the case $k = 0$ and where the explicit reduction to separable form was not required. A rectangular domain $R_x \subset \mathbb{R}^n$ is constructed to contain the

projection $\Omega_x$ of $\Omega$ on the $x$-space by solving a multiple-cost-row linear program with $n$ cost rows. The original problem is then transformed into an equivalent concave separable quadratic minimization problem with linear constraints. A linear underestimating function $\Gamma(z)+d'y$, in the new nonlinear variables $z \in \mathbb{R}^n$ and the same linear variables $y \in \mathbb{R}^k$, to the transformed separable quadratic function $\psi(z, y)$ is easily computed and the linear program

$$\min_{(z,y)\in\Omega'} \Gamma(z)+d'y$$

is solved, where $\Omega'$ is the transformed bounded convex polyhedron. The solution of this linear program gives both upper and lower bounds on the global minimum function value $\psi^*$. Branch and bound techniques are then applied to reduce the feasible region under consideration and decrease the difference between the upper and lower bounds.

The next four sections describe the construction of the linear underestimating function, and its use to obtain upper and lower bounds on the global optimum function value is justified. The branch and bound techniques used to eliminate subregions are also presented (Theorem 1), and improved bounds are obtained (Theorem 2). In Section 6 the stage I algorithm is presented. Stage I gives an initial approximate solution with a minimum of computation but with a relatively large bound. It is also used repetitively in each stage II major iteration. Stage II of the algorithm is presented and justified in Sections 7 and 8. In Section 9 the worst case behavior of the algorithm is analyzed, and it is shown that an $\varepsilon$-approximate solution is guaranteed with the solution of a large, but finite, number of subproblems. Section 10 discusses some aspects of the parallel implementation. The computational results obtained on the four processor CRAY2 are summarized in Sections 11 and 12. In Section 11 the manner in which the test problems were generated is described. The results obtained are given in Section 12 and compared (for small problems) with recent results obtained by two other methods.

The computational results are presented by means of nine figures which show both average and extreme values of the computation time as a function of $k$, the number of linear variables. Results for stage I are given in Figures 3 and 4, and for the combined stages I and II in Figures 5 thru 8. Figures 3 and 4 show that the computation time for stage I depends linearly on $k$. Both sequential and parallel results are given, and the speedup using four processors is also shown (Figures 9 and 10).

## 2. The initial linear underestimator

The concave quadratic function $\psi(x, y)$ can easily be reduced to separable form by a linear transformation based on the eigenstructure of the matrix $Q$. This reduction is described in detail in Rosen and Pardalos (1986) and Phillips and Rosen (1987a)

and will not be repeated here. Computationally it requires finding the eigenvalues and eigenvectors of the symmetric matrix $Q$ and the solution of a multiple-cost-row linear program (with $n$ cost rows). For $n \leq 100$ finding the eigenvalues and eigenvectors requires less than one second of (CRAY2) CPU time. Hence, without loss of generality, we limit consideration to the following concave separable quadratic minimization problem

$$\text{(SQ)} \qquad \operatorname*{global\,min}_{(x,y)\in\Omega} \psi(x, y) = \varphi(x) + d^t y = \sum_{i=1}^{n} q_i(x_i) + d^t y$$

where $\Omega = \{(x, y): A_1 x + A_2 y \leq b, x \geq 0, y \geq 0\}$, $q_i(x_i) = c_i x_i - \frac{1}{2}\lambda_i x_i^2$, and where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^k$, $d \in \mathbb{R}^k$, $b \in \mathbb{R}^m$, $A_1 \in \mathbb{R}^{m\times n}$, and $A_2 \in \mathbb{R}^{m\times k}$ (Note that since $Q$ is positive semi-definite $\lambda_i \geq 0$, and that in fact $\lambda_i > 0$ since we assume that each $x_i$ is a nonlinear variable).

Given a concave quadratic minimization problem that has been reduced to separable form, we can easily construct the smallest rectangular domain $R_x \subset \mathbb{R}^n$ which contains $\Omega_x$, the projection of $\Omega$ onto the $x$-space. We solve the multiple-cost-row linear program (with $n$ cost rows)

$$\text{(MCR)} \quad \max_{(x,y)\in\Omega} x_i$$

for each $i = 1, \ldots, n$ to get optimal function values $\beta_i$, $i = 1, \ldots, n$. Then the rectangular domain $R_x$ can be expressed as

$$R_x = \{x: 0 \leq x_i \leq \beta_i, i = 1, \ldots, n\}.$$

A linear underestimating function $\Gamma(x)$ of $\varphi(x)$ over $R_x$ which agrees with $\varphi(x)$ at every vertex of $R_x$ can now be constructed. This linear function $\Gamma(x)$ is given by

$$\Gamma(x) = \sum_{i=1}^{n} \gamma_i(x_i)$$

where $\gamma_i(x_i) = (c_i - \frac{1}{2}\lambda_i\beta_i)x_i$. Since $\Gamma(x)$ underestimates $\varphi(x)$ on $R_x$, it also underestimates $\varphi(x)$ on $\Omega_x \subset R_x$, and hence $\Gamma(x) + d^t y$ is a linear underestimator of $\psi(x, y) = \varphi(x) + d^t y$ over $\Omega$. The solution to the linear program

$$\text{(LU)} \qquad \min_{(x,y)\in\Omega} \Gamma(x) + d^t y$$

provides a vertex $(x', y')$ of $\Omega$ such that

$$\Gamma(x') + d^t y' \leq \psi(x^*, y^*) \leq \psi(x', y')$$

where $\psi(x^*, y^*) = \psi^*$ is the global optimum for SQ. Let $\Gamma^{(1)} = \Gamma(x') + d^t y'$ and $\psi^{(1)} = \psi(x', y')$. Thus, $\Gamma^{(1)}$ is a lower bound and $\psi^{(1)}$ is an upper bound for the global optimum function value $\psi^*$.

## 3. Error bounds for the initial linear underestimator

As previously stated, the lower and upper bounds for $\psi^*$ are given by $\Gamma^{(1)}$ and $\psi^{(1)}$, respectively. If we define an appropriate scale factor $\Delta\varphi$ to be the maximum

of the range of $\varphi(x)$ on $\Omega$ and the range of $d^t y$ on $\Omega$, then we shall say that $(x', y')$ is an $\varepsilon$-approximate solution if, for a given $\varepsilon > 0$,

$$\psi(x', y') - \psi^* \leq \varepsilon \Delta\varphi.$$

Defining the difference $E(x) = \varphi(x) - \Gamma(x)$ we note that

$$\psi(x', y') - \psi^* \leq E(x')$$

and hence, if $E(x') \leq \varepsilon\Delta\varphi$ then $(x', y')$ is an $\varepsilon$-approximate solution. We can easily obtain a bound on $E(x)$ for any $x \in \Omega_x \subset R_x$. Assume, without loss of generality, that

$$\lambda_1\beta_1^2 \geq \lambda_i\beta_i^2, \quad i = 1, \ldots, n,$$

and define the ratios

$$\rho_i = (\lambda_i\beta_i^2)/(\lambda_1\beta_1^2), \quad i = 1, \ldots, n.$$

Note that $\rho_i \leq 1$ for $i = 1, \ldots, n$. Now,

$$E(x) = \varphi(x) - \Gamma(x) = \sum_{i=1}^{n} [q_i(x_i) - \gamma_i(x_i)] = \tfrac{1}{2} \sum_{i=1}^{n} \lambda_i(\beta_i - x_i)x_i$$

which attains its maximum at $x_i = \beta_i/2$ for $i = 1, \ldots, n$. Thus, for any $x \in \Omega_x \subset R_x$,

$$E(x) \leq \tfrac{1}{8} \sum_{i=1}^{n} \lambda_i\beta_i^2 = \tfrac{1}{8}\lambda_1\beta_1^2 \sum_{i=1}^{n} \rho_i.$$

Furthermore, we can easily obtain a bound on the scale factor $\Delta\varphi$. We first define the following quantities:

$$\varphi_{\max} = \max_{x \in R_x} \varphi(x),$$

$$\varphi_{\min} = \min_{x \in R_x} \varphi(x),$$

$$DY_{\max} = \max_{(x,y) \in \Omega} d^t y,$$

$$DY_{\min} = \min_{(x,y) \in \Omega} d^t y.$$

Then by definition $\Delta\varphi = \max\{\varphi_{\max} - \varphi_{\min}, DY_{\max} - DY_{\min}\}$. Note that

$$\varphi_{\min} = \sum_{i=1}^{n} \min\{q_i(0), q_i(\beta_i)\}, \qquad \varphi_{\max} = \sum_{i=1}^{n} q_{i_{\max}}$$

where

$$q_{i_{\max}} = \begin{cases} q_i(0) & \text{if } \bar{\omega}_i \leq 0, \\ q_i(\beta_i) & \text{if } \bar{\omega}_i \geq \beta_i, \\ q_i(\bar{\omega}_i) & \text{if } \bar{\omega}_i \in (0, \beta_i), \end{cases}$$

and where $\bar{\omega}_i = c_i/\lambda_i$ for $1 = 1, \ldots, n$ is the unconstrained maximum of $q_i(x_i)$. Also notice that $DY_{\max}$and $DY_{\min}$ are obtained by solving two linear programs where

the feasible region is $\Omega$. Further, all linear programs solved since the reduction to separable form have been solved over the same feasible region $\Omega$ and can therefore be classified as multiple-cost-row linear programs. This property is significant for computational purposes and is a major difference between this approach and other earlier methods such as Falk and Hoffman (1976) and Horst (1976).

A lower bound for $\Delta\varphi_N \equiv \varphi_{\max} - \varphi_{\min}$ depends on the distance between $\bar{\omega}_i$ and $\beta_i/2$ for $\bar{\omega}_i \in (0, \beta_i)$ and is independent of $\bar{\omega}_i$ for $\bar{\omega}_i \notin (0, \beta_i)$. This dependence is expressed by the quantities

$$\eta_i = \min\left\{1, \left|\frac{2\bar{\omega}_i}{\beta_i} - 1\right|\right\}, \quad i = 1, \ldots, n.$$

Notice that $0 \leq \eta_i \leq 1$ and that $\eta_i = 0$ iff $\bar{\omega}_i = \beta_i/2$. Also, $\eta_i = 1$ for all $\bar{\omega}_i \leq 0$ and for all $\bar{\omega}_i \geq \beta_i$. Then, as shown by Rosen and Pardalos (1986), we get that

$$\Delta\varphi_N \equiv \varphi_{\max} - \varphi_{\min} \geq \tfrac{1}{8}\lambda_1\beta_1^2 \sum_{i=1}^{n} \rho_i(1+\eta_i)^2.$$

Hence, an a priori upper bounded on the relative error is given by

$$\frac{\psi(x', y') - \psi^*}{\Delta\varphi} \leq \frac{\sum_{i=1}^{n} \rho_i}{\sum_{i=1}^{n} \rho_i(1+\eta_i)^2} \equiv \sigma(\rho, \eta)$$

and it is easily seen that $\sigma(\rho, \eta) \in [\frac{1}{4}, 1]$ and that $\sigma(\rho, \eta) = 1$ iff $\bar{\omega}_i = \beta_i/2$ for all $i = 1, \ldots, n$. Furthermore, $\sigma(\rho, \eta) = \frac{1}{4}$ iff $\bar{\omega}_i \notin (0, \beta_i)$ for all $i = 1, \ldots, n$.

## 4. The $2n$ linear underestimators

During each minor iteration of the algorithm, the feasible region is divided into subregions by bisecting $\Omega_x$ in certain directions. Linear underestimating functions to $\psi(x, y)$ over the subregions are then constructed and minimized over the original region $\Omega$. More specifically, define $R = \{x: \beta_{i1} \leq x_i \leq \beta_{i2}, i = 1, \ldots, n\}$ where initially we would have $\beta_{i1} = 0$ and $\beta_{i2} = \beta_i$ for $i = 1, \ldots, n$ (i.e. $R = R_x$). Let $l \in \{1, \ldots, n\}$ and let $\beta_{l3} = (\beta_{l1} + \beta_{l2})/2$ be the midpoint of the interval $[\beta_{l1}, \beta_{l2}]$. Define $\gamma_{l1}(x_l)$ to be the linear function which interpolates the points $[\beta_{l1}, q_l(\beta_{l1}))$ and $(\beta_{l3}, q_l(\beta_{l3}))$. Similarly, define $\gamma_{l2}(x_l)$ to be the linear function which interpolates the points $(\beta_{l3}, q_l(\beta_{l3}))$ and $(\beta_{l2}, q_l(\beta_{l2}))$. Also, for $i = 1, \ldots, n$ let $\gamma_i(x_i)$ be the linear function which interpolates the points $(\beta_{i1}, q_i(\beta_{i1}))$ and $(\beta_{i2}, q_i(\beta_{i2}))$. It is easily shown that

$$\gamma_i(x_i) = \tfrac{1}{2}\lambda_i\beta_{i1}\beta_{i2} + (c_i - \tfrac{1}{2}\lambda_i(\beta_{i1} + \beta_{i2}))x_i, \quad i = 1, \ldots, n.$$

Clearly, $\gamma_i(x_i) \leq q_i(x_i)$ on $[\beta_{i1}, \beta_{i2}]$. Likewise,

$$\gamma_{l1}(x_l) = \tfrac{1}{2}\lambda_l\beta_{l1}\beta_{l3} + (c_l - \tfrac{1}{2}\lambda_l(\beta_{l1} + \beta_{l3}))x_l \quad \text{and}$$

$$\gamma_{l2}(x_l) = \tfrac{1}{2}\lambda_l\beta_{l3}\beta_{l2} + (c_l - \tfrac{1}{2}\lambda_l(\beta_{l3} + \beta_{l2}))x_l$$

so that $\gamma_{l1}(x_l) \leq q_l(x_l)$ on $[\beta_{l1}, \beta_{l3}]$ and $\gamma_{l2}(x_l) \leq q_l(x_l)$ on $[\beta_{l3}, \beta_{l2}]$. Now, consider the linear functions

$$\Gamma_{l1}(x) = \sum_{\substack{i=1 \\ i \neq l}}^{n} \gamma_i(x_i) + \gamma_{l1}(x_l) \quad \text{and} \quad \Gamma_{l2}(x) = \sum_{\substack{i=1 \\ i \neq l}}^{n} \gamma_i(x_i) + \gamma_{l2}(x_l)$$

and the hyperrectangles

$$R_{l1} = \{x: x \in R, \beta_{l1} \leq x_l \leq \beta_{l3}\} \text{ for } l = 1, \ldots, n, \quad \text{and}$$

$$R_{l2} = \{x: x \in R, \beta_{l3} \leq x_l \leq \beta_{l2}\} \text{ for } l = 1, \ldots, n$$

then we have that $\Gamma_{l1}(x)$ underestimates $\varphi(x)$ on $R_{l1}$ and that $\Gamma_{l2}(x)$ underestimates $\varphi(x)$ on $R_{l2}$. It is shown in Phillips and Rosen (1987a) that these linear underestimators are in fact the "best" convex underestimators of $\varphi(x)$ over their respective regions (see also Kalantari 1984).

## 5. Branch and bound techniques

As stated earlier, during each minor iteration of the algorithm the feasible region is divided into subregions by bisecting $R_x$ in certain directions (the branching step) and linear underestimating functions to $\psi(x, y)$ over the subregions are then constructed and minimized over the original region $\Omega$. Bounding techniques can then be applied to determine if any of the subregions cannot contain the global optimum vertex $(x^*, y^*)$ and may therefore be eliminated from further consideration.

Given the hyperrectangles $R$, $R_{l1}$, and $R_{l2}$ and their corresponding linear underestmators $\Gamma(x)$, $\Gamma_{l1}(x)$, and $\Gamma_{l2}(x)$ as defined in the previous section, we solve the multiple-cost-row linear program (with $2n$ cost rows)

$$(\mathrm{LU}_{l_j}) \qquad \min_{(x,y) \in \Omega} \Gamma_{l_j}(x) + d^t y$$

for $l = 1, \ldots, n$ and $j = 1, 2$. Notice that the minimization occurs over the entire region $\Omega$. Let $(x_{l_j}, y_{l_j})$ be the solution vertex corresponding to problem $\mathrm{LU}_{l_j}$. Also, let $\Gamma_{l_j}(x_{l_j}) + d^t y_{l_j}$ be the corresponding optimal function value, and denote this by $\Gamma_{l_j}$. Define $\psi_{\mathrm{incb}}$ to be the "incumbent" function value, i.e. the lowest upper bound at any given time (note: $\psi_{\mathrm{incb}} = \psi^{(1)}$ immediately after the first linear underestimator). We now have the important

**Theorem 1.** *If* $\Gamma_{l1} > \min\{\psi_{\mathrm{incb}}, \psi(x_{l1}, y_{l1})\}$ *for some* $l \in \{1, \ldots, n\}$, *then* $x^* \notin R_{l1}$ *and hence* $R_{l1}$ *can be eliminated from further consideration. Likewise, if* $\Gamma_{l2} > \min\{\psi_{\mathrm{incb}}, \psi(x_{l2}, y_{l2})\}$ *for some* $l \in \{1, \ldots, n\}$, *then* $x^* \notin R_{l2}$ *and hence* $R_{l2}$ *can be eliminated from further consideration.*

**Proof.** $(x^*, y^*) \in \Omega$ by definition and $\min\{\psi_{\text{incb}}, \psi(x_{l1}, y_{l1})\} < \Gamma_{l1} \equiv \Gamma_{l1}(x_{l1}) + d^t y_{l1} =$ minimum of $\Gamma_{l1}(x) + d^t y$ over $\Omega$. Hence $\min\{\psi_{\text{incb}}, \psi(x_{l1}, y_{l1})\} < \Gamma_{l1} \leq \Gamma_{l1}(x) + d^t y$ for all $(x, y) \in \Omega$ and thus for all $(x, y) \in \Omega \cap (R_{l1} \times \Omega_y)$. If $x^* \in R_{l1}$ then $\min\{\psi_{\text{incb}}, \psi(x_{i1}, y_{l1})\} < \Gamma_{l1}(x^*) + d^t y^* \leq \psi(x^*, y^*)$ since $\Gamma_{l1}(x)$ underestimates $\varphi(x)$ for all $x \in R_{l1}$. But this is a contradiction since $\psi(x^*, y^*)$ is the global minimum over $\Omega$. Thus $x^* \notin R_{l1}$. A similar proof yields the second part of the theorem. □

From this theorem we can see that if $\Gamma_{lj} > \min\{\psi_{\text{incb}}, \psi(x_{lj}, y_{lj})\}$ for either $j = 1$ or 2, then $R_{lj}$ can be eliminated from further consideration. Additionally, Theorem 1 guarantees that if a subregion $R_{lj}$ of the feasible region is eliminated, then that subregion cannot contain any feasible point which has a lower function value than $\min\{\psi_{\text{incb}}, \psi(x_{lj}, y_{lj})\}$.

From these $2n$ linear programs, we would also like to obtain possibly better upper and lower bounds on the global minimum $\psi(x^*, y^*)$. We do this by first defining

$$\Gamma^{(2)} = \max_{l=1,\ldots,n} \min\{\Gamma_{l1}, \Gamma_{l2}\} \quad \text{and} \quad \psi^{(2)} = \min_{l=1,\ldots,n} \{\psi(x_{l1}, y_{l1}), \psi(x_{l2}, y_{l2})\}.$$

Using these definitions, we have the

**Theorem 2.** *If $x^* \in R$ then $\Gamma^{(2)} \leq \psi(x^*, y^*) \leq \psi^{(2)}$.*

**Proof.** The upper bound is obvious since each $(x_{lj}, y_{lj}) \in \Omega$ for $l = 1, \ldots, n$ and $j = 1, 2$. For the lower bound we have two cases to consider.

*Case 1*: Suppose $x^* \in R_{l1}$. Then $\Gamma_{l1}(x^*) + d^t y^* \leq \psi(x^*, y^*)$ since $\Gamma_{l1}(x) \leq \varphi(x)$ for all $x \in R_{l1}$. Also, $\Gamma_{l1} \equiv \Gamma_{l1}(x_{l1}) + d^t y_{l1} \leq \Gamma_{l1}(x^*) + d^t y^*$ since $\Gamma_{l1}$ is the minimum of $\Gamma_{l1}(x) + d^t y$ over $\Omega$ and $(x^*, y^*) \in \Omega$. It follows that $\Gamma_{l1} \leq \psi(x^*, y^*)$ and hence that $\min\{\Gamma_{l1}, \Gamma_{l2}\} \leq \psi(x^*, y^*)$.

*Case 2*: Suppose $x^* \notin R_{l1}$, i.e. $x^* \in R_{l2}$ (since $x^* \in R$). Then as above, $\Gamma_{l2}(x^*) + d^t y^* \leq \psi(x^*, y^*)$ and $\Gamma_{l2} \leq \Gamma_{l2}(x^*) + d^t y^*$. It follows that $\Gamma_{l2} \leq \psi(x^*, y^*)$ and hence that $\min\{\Gamma_{l1}, \Gamma_{l2}\} \leq \psi(x^*, y^*)$.

Thus, combining Cases 1 and 2, if $x^* \in R$ then $\min\{\Gamma_{l1}, \Gamma_{l2}\} \leq \psi(x^*, y^*)$. This holds for $l = 1, \ldots, n$ so that we get $\Gamma^{(2)} \leq \psi(x^*, y^*)$. □

From Theorem 2 we get upper and lower bounds on $\psi^* = \psi(x^*, y^*)$. In fact, if we continue the algorithm by bisecting the new hyperrectangle (obtained from the old hyperrectangle by eliminating subregions according to Theorem 1), we get another set of upper and lower bounds. After $k$ minor iterations of the algorithm, $\psi_{\text{incb}} = \min\{\psi^{(1)}, \psi^{(2)}, \ldots, \psi^{(k)}\}$ is the "best" upper bound and hence is also the approximation to the global optimum function value. Note that the previous results show that the process can continue as long as at least one of the $2n$ linear programs

$P_{lj}$ at each minor iteration provides a solution which satisfies the conditions of Theorem 1 (i.e. some subregion can be eliminated). An initial algorithm (stage I), based on these results, is now presented.

## 6. Stage I: The initial algorithm

Given a linearly constrained feasible region $\Omega$ (nonempty and bounded), a concave separable quadratic function $\psi(x, y)$, a hyperrectangle $R = \{x: \beta_{i1} \leq x_i \leq \beta_{i2}, i = 1, \ldots, n\}$ and an a priori lower bound $\Gamma^{(0)}$ on $\psi^*$, the general algorithm can be stated as

**Alg1($R$, $\Omega$, $\Gamma^{(0)}$)**

1. Construct the linear function $\Gamma(x) + d^t y$ which agrees with $\psi(x, y)$ at all vertices of $R$.
2. Solve the linear program

$$(\text{LU}) \quad \min_{(x,y)\in\Omega} \Gamma(x) + d^t y$$

to get the vertex $(x', y')$. Set $\Gamma^{(1)} := \max\{\Gamma(x') + d^t y', \Gamma^{(0)}\}$, $\psi^{(1)} := \psi(x', y')$, $x_0 := x'$, $y_o := y'$, and $k := 2$. If $\psi^{(1)} - \Gamma^{(1)} \leq \varepsilon\Delta\varphi$ then stop and accept $(x_0, y_0)$ as the global solution vertex with corresponding function value $\psi^{(1)}$.

3. For each $l = 1, \ldots, n$ construct the linear function $\Gamma_{l1}(x) + d^t y$ which agrees with $\psi(x, y)$ at all vertices of $R_{l1}$ and the linear function $\Gamma_{l1}(x) + d^t y$ which agrees with $\psi(x, y)$ at all vertices of $R_{l2}$, where $R_{l1} = x: x \in R, \beta_{l1} \leq x_l \leq \beta_{l3}\}$ and $R_{l2} = \{x: x \in R, \beta_{l3} \leq x_l \leq \beta_{l2}\}$, and where $\beta_{l3} = (\beta_{l1} + \beta_{l2})/2$.
4. Solve the multiple-cost-row linear program

$$(\text{LU}_{lj}) \quad \min_{(x,y)\in\Omega} \Gamma_{lj}(x) + d^t y$$

for $l = 1, \ldots, n$ and $j = 1, 2$. Let $(x_{lj}, y_{lj})$ be the solution vertex corresponding to problem $\text{LU}_{lj}$. Also, let $\Gamma_{lj}(x_{lj}) + d^t y_{lj}$ be the corresponding optimal function value, and denote this by $\Gamma_{lj}$.

5. For each $l = 1, \ldots, n$ if $\Gamma_{l1} > \min\{\psi^{(k-1)}, \psi(x_{l1}, y_{l1})\}$ then set $\beta_{l1} := \beta_{l3}$. Likewise, if $\Gamma_{l2} > \min\{\psi^{(k-1)}, \psi(x_{l2}, y_{l2})\}$ then set $\beta_{l2} := \beta_{l3}$.
6. Set

$$\Gamma^{(k)} = \max_{l=1,\ldots,n} \min\{\Gamma_{l1}, \Gamma_{l2}\} \quad \text{and} \quad \psi^{(k)} = \min_{l=1,\ldots,n} \{\psi(x_{l1}, y_{l1}), \psi(x_{l2}, y_{l2})\}.$$

7. If $\Gamma^{(k)} < \Gamma^{(k-1)}$ then set $\Gamma^{(k)} := \Gamma^{(k-1)}$. If $\psi^{(k)} > \psi^{(k-1)}$ then set $\psi^{(k)} := \psi^{(k-1)}$. Update $(x_0, y_0)$ to be vertex (of $\Omega$) with corresponding function value $\psi^{(k)}$.
8. If $\psi^{(k)} - \Gamma^{(k)} \leq \varepsilon\Delta\varphi$ then stop and accept $(x_0, y_0)$ as the global solution vertex with corresponding function value $\psi^{(k)}$.
9. If no eliminations were made in step 5, then stop. Otherwise, set $R := \{x: \beta_{i1} \leq x_i \leq \beta_{i2}, i = 1, \ldots, n\}$, $k := k + 1$, and go to 3.

The justification for the algorithm is given by the theorems of the previous section. That is, since $x^* \in R$ initially, then Theorem 1 implies that the actions taken in step 5 of the algorithm will guarantee that $x^* \in R$ for the next minor iteration (i.e. $x^*$ is in the new hyperrectangle which was obtained by eliminating subregions of the old hyperrectangle). Inductively, these arguments apply to each minor iteration $k = 2, 3, \ldots$ of the algorithm. Similarly, the upper and lower bounds of steps 6 and 7 are justified by Theorem 2 of the previous section. Note that $\psi^{(k)}$ is really just the incumbent function value $\psi_{\text{incb}}$ at minor iteration $k$ of the algorithm. We use the name $\psi^{(k)}$ instead of $\psi_{\text{incb}}$ to represent the incumbent value at this point because we will use the name $\psi_{\text{incb}}$ in a more general way in a later section.

## 7. Stage II: Restarting when no further eliminations are possible

An important drawback of the algorithm as it stands (stage I only) is that termination at an $\varepsilon$-approximate solution (for a specified small $\varepsilon$) is not guaranteed. That is, at some minor iteration $k$ the algorithm may be forced to stop if no subregion can be eliminated and $\psi^{(k)} - \Gamma^{(k)} > \varepsilon \Delta\varphi$. At such a time we have available an incumbent vertex $v$ with associated function value $\psi_{\text{incb}}$, a lower bound $\Gamma$, and a hyperrectangle $R$ such that $x^* \in R$ (and $R \subset R_x$). To resume the algorithm we now pick some direction $e_i$ and bisect $R$ along that direction to get two new hyperrectangles $R_1$ and $R_2$. This is done by adding one additional bound (upper or lower) to the constraints. That is, we add a single upper bound to the feasible region $\Omega$ (to get $\Omega_1$) to ensure that the region $R_2$ is excluded from feasibility when $R_1$ is the hyperrectangle under consideration. Likewise, a corresponding lower bound is added to $\Omega$ (to get $\Omega_2$) to ensure that the region $R_1$ is excluded from feasibility when $R_2$ is the hyperrectangle under consideration. Thus, two independent subproblems are generated with $\Omega_1$ and $\Omega_2$ as the feasible domains and $R_1$ and $R_2$ as the respective hyperrectangles. It is important to note at this point that the addition of this bounding constraint to the feasible region $\Omega$ is not required to ensure convergence of the algorithm. In fact, all of the convergence properties to be presented later are obtained independent of the feasible region under consideration after all eliminations and bisections. The bounding constraint is added to $\Omega$ in order to facilitate pruning of the feasible region under consideration at each major iteration of the algorithm.

Let the hyperrectangle $R = \{x: \beta_{j1} \leq x_j \leq \beta_{j2}, j = 1, \ldots, n\}$. Pick $i \in \{1, \ldots, n\}$ and let $\beta_{i3} = (\beta_{i1} + \beta_{i2})/2$. Then for $R_1 = \{x: x \in R \text{ and } x_i \leq \beta_{i3}\}$ and $R_2 = \{x: x \in R \text{ and } x_i \geq \beta_{i3}\}$ we let $\Omega_1 = \Omega \cap \{(x, y): x_i \leq \beta_{i3}\}$ and $\Omega_2 = \Omega \cap \{(x, y): x_i \geq \beta_{i3}\}$. We can then apply the algorithm $\text{Alg1}(R_1, \Omega_1, \Gamma)$ to the feasible region $\Omega_1$ with hyperrectangle $R_1$, incumbent vertex $v$ with associated function value (upper bound) $\psi_{\text{incb}}$, and lower bound $\Gamma$. Similarly, and in parallel, we can apply the algorithm $\text{Algl}(R_2, \Omega_2, \Gamma)$ to the feasible region $\Omega_2$ with hyperrectangle $R_2$. Clearly, the procedure can again be applied to the hyperrectangles (with associated feasible

regions $\Omega_1$ and $\Omega_2$) obtained from this second application of the algorithm. In addition, if at any time the lower bound for a subproblem is not less than the current incumbent function value then that subproblem may be "pruned", i.e. eliminated from further consideration. In this way the new procedure is a parallel branch and bound algorithm.

The choice of the bisected direction $e_i$ can be made in many ways. One such choice is to pick $i$ such that

$$\lambda_i(\beta_{i2}-\beta_{i1})^2 = \max_{j=1,\ldots,n} \lambda_j(\beta_{j2}-\beta_{j1})^2$$

where the hyperrectangle $R=\{x: \beta_{j1} \leq x_j \leq \beta_{j2}, j=1,\ldots,n\}$. It is easily shown that the error $E(x)$ over the hyperrectangle $R$ is bounded above by

$$E(x) \leq \tfrac{1}{8} \sum_{j=1}^{n} \lambda_j(\beta_{j2}-\beta_{j1})^2.$$

The $i$th term of this error over $R$ is $\frac{1}{8}\lambda_i(\beta_{i2}-\beta_{i1})^2$. Since direction $e_i$ was the bisected direction, the $i$th term of the error over the two sub-hyperrectangles $R_1$ and $R_2$ is $\frac{1}{32}\lambda_i(\beta_{i2}-\beta_{i1})^2$. Hence, each subproblem has an upper bound on the error which strictly decreases as the algorithm proceeds. Since the $i$th term of the error is the largest of the error terms and it strictly decreases by a factor of four at each major iteration (the point at which a new constraint is added), then at some major iteration with hyperrectangle $R=\{x: \beta_{j1} \leq x_j \leq \beta_{j2}, j=1,\ldots,n\}$

$$\tfrac{1}{8}\lambda_j(\beta_{j2}-\beta_{j1})^2 \leq \varepsilon\Delta\varphi/n \quad \text{for } j=1,\ldots,n$$

so that $E(x) \leq \varepsilon\Delta\varphi$ for all $x \in R$, and hence finite convergence to an $\varepsilon$-approximate solution is guaranteed.

## 8. The parallel branch and bound algorithm

Given a linearly constrained feasible region $\Omega'$ (nonempty and bounded) and a concave quadratic function $\psi'(x, y)$, the parallel branch and bound algorithm can be stated as

**Alg2($\psi'$, $\Omega'$)**

1. Reduce the problem to separable form to get the separable quadratic function $\psi(x, y)$ and the transformed feasible domain $\Omega$ (nonempty and bounded).

2. Compute the enclosing hyperrectangle $R_x$ and the range $\Delta\varphi$ by solving (in parallel) the multiple-cost-row linear program

$$\text{(MCR)} \quad \max_{(x,y)\in\Omega} a_{i1}^t x + a_{i2}^t y$$

where

$$a_{i1}=\begin{cases} e_i, & i=1,\ldots,n,\\ 0, & i=n+1,n+2.\end{cases} \quad \text{and} \quad a_{i2}=\begin{cases} 0 & i=1,\ldots,n,\\ d & i=n+1,\\ -d & i=n+2,\end{cases}$$

for each $i=1,\ldots,n+2$ to get the vertices $v_1,\ldots,v_{n+2}$ with corresponding optimal function values $\beta_1,\ldots,\beta_n, DY_{\max}, -DY_{\min}$. Set the initial hyperrectangle $R_1^{(1)} := \{x: 0 \leq x_i \leq \beta_i,\ i=1,\ldots,n\}$ and the initial feasible domain $\Omega_1^{(1)} := \Omega$. Compute $\Delta\varphi$ (as described in Section 3.0).

3. Set $S := \emptyset$, $I^{(1)} := \{1\}$, $\psi_{\text{incb}} := \min\{\psi(v_1),\ldots,\psi(v_{n+2})\}$, $L_1^{(1)} := -\infty$, and $k := 1$.
4. Set $I^{(k+1)} := \emptyset$.
5. For each $j \in I^{(k)}$ do 5.1–5.5 (all $j \in I^{(k)}$ in parallel)
5.1. Apply Alg1$(R_j^{(k)}, \Omega_j^{(k)}, L_j^{(k)})$ to get the upper bound $\psi_j^{(k)}$, the lower bound $\Gamma_j^{(k)}$, and the candidate vertex $v_j^{(k)}$.
5.2. If $\psi_j^{(k)} - \Gamma_j^{(k)} \leq \varepsilon\Delta\varphi$ then set $S := S \cup \{(k,j)\}$ and go 5.5.
5.3. Denote the current hyperrectangle by $R_j^{(k)} = \{x: \beta_{i1}^{k,j} \leq x_i \leq \beta_{i2}^{k,j},\ i=1,\ldots,n\}$ and pick $i \in \{1,\ldots,n\}$ such that

$$\lambda_i(\beta_{i2}^{k,j} - \beta_{i1}^{k,j})^2 = \max_{s=1,\ldots,n} \lambda_s(\beta_{s2}^{k,j} - \beta_{s1}^{k,j})^2.$$

5.4. Set $\beta_3^{k,j} := (\beta_{i2}^{k,j} + \beta_{i1}^{k,j})/2$. Set $R_{2j-1}^{(k+1)} := \{x: x \in R_j^{(k)} \text{ and } x_i \leq \beta_3^{k,j}\}$ and $R_{2j}^{(k+1)} := \{x: x \in R_j^{(k)} \text{ and } x_i \geq \beta_3^{k,j}\}$. Set $\Omega_{2j-1}^{(k+1)} := \Omega_j^{(k)} \cap \{(x,y): x_i \leq \beta_3^{k,j}\}$ and $\Omega_{2j}^{(k+1)} := \Omega_j^{(k)} \cap \{(x,y): x_i \geq \beta_3^{k,j}\}$. Set $L_{2j-1}^{(k+1)} := \Gamma_j^{(k)}$, and $L_{2j}^{(k+1)} := \Gamma_j^{(k)}$. Set $I^{(k+1)} := I^{(k)} \cup \{2j-1, 2j\}$.
5.5. Continue step 5.0.
6. If $I^{(k+1)} = \emptyset$ then set $k := k+1$ and go to 4.
7. Set $\psi := \psi_s^{(r)} = \min\ \psi_j^{(k)}$ for all $(k,j) \in S$, $\Gamma := \min \Gamma_j^{(k)}$ for all $(k,j) \in S$, and $v := v_s^{(r)}$.

Notationally, $I^{(k)}$ represents the set of subscripts (names) of the subproblems to be solved at major iteration $k$ of the algorithm, i.e. it is merely a way to keep track of which subproblems are to be solved at a given major iteration $k$. Clearly, when $I^{(k)} = \emptyset$ the problem is solved. The set $S$ represents the set of index pairs $(k,j)$ such that the subproblem characterized by the hyperrectangle $R_j^{(k)}$ and the feasible region $\Omega_j^{(k)}$ was solved within the tolerance $\varepsilon\Delta\varphi$. Upon termination of the algorithm, the solution vertex $v$ with associated function value $\psi$ and lower bound $\Gamma$ satisfies the obvious

**Theorem 3.** *$\psi - \Gamma \leq \varepsilon\Delta\varphi$ and $\Gamma \leq \psi^* \leq \psi$.*

**Proof.** $\psi := \min \psi_j^{(k)}$ for all $(k,j) \in S$ and $\Gamma := \min \Gamma_j^{(k)}$ for all $(k,j) \in S$, where $S = \{(k,j): \psi_j^{(k)} - \Gamma_j^{(k)} \leq \varepsilon\Delta\varphi\}$. Suppose $\Gamma = \Gamma_p^{(q)}$ where $(q,p) \in S$. Then, $\psi_p^{(q)} - \Gamma_p^{(q)} \leq \varepsilon\Delta\varphi$ and hence, $\psi - \Gamma = \psi - \Gamma_p^{(q)} \leq \psi_p^{(q)} - \Gamma_p^{(q)} \leq \varepsilon\Delta\varphi$. Also, $x^* \in R_j^{(k)}$ for some $(k,j) \in S$ so that $\Gamma = \Gamma_p^{(q)} \leq \Gamma_j^{(k)} \leq \psi^* \leq \psi$ since $\Gamma_j^{(k)}$ is a lower bound on $\psi^*$, and $\psi$ is always an upper bound on $\psi^*$. □

Four important additions need to be made to the initial algorithm Alg1 in order to properly update the incumbent function value $\psi_{\text{incb}}$, allow pruning, and keep the upper bound for each subproblem as tight as possible. We add to Alg1 the following four steps:

2.1. If $\psi^{(1)} < \psi_{\text{incb}}$ then $\psi_{\text{incb}} := \psi^{(1)}$
else $\psi^{(1)} := \psi_{\text{incb}}$.

2.2. If $\Gamma^{(1)} \geq \psi_{\text{incb}}$ then terminate this problem (i.e. prune).

7.1. If $\psi^{(k)} < \psi_{\text{incb}}$ then $\psi_{\text{incb}} := \psi^{(k)}$
else $\psi^{(k)} := \psi_{\text{incb}}$.

7.2. If $\Gamma^{(k)} \geq \psi_{\text{incb}}$ then terminate this problem (i.e. prune).

Note that in Alg2 the final function value (the approximate global solution) $\psi$ is really just the incumbent function value $\psi_{\text{incb}}$ and that $\psi_{\text{incb}}$ is intended to be a value available to all subproblems (i.e. a globally shared value).

## 9. Convergence analysis

To analyze the worst case performance of the algorithm, we assume that for each subproblem generated no eliminations can be performed (as given by Theorem 1). Since at each minor iteration we cannot discard any of the feasible region $\Omega$, we are forced to generate two new subproblems for each problem at the current major iteration. Assume also that we cannot prune any of the subproblems. Then, in order to terminate, we must satisfy $E(x) \leq \varepsilon \Delta\varphi$ over each hyperrectangle $R_s^{(k)}$ for $s = 1, \ldots, 2^{k-1}$ and at some major iteration $k$. That is, we must have

$$\tfrac{1}{8} \sum_{i=1}^{n} \lambda_i (\beta_{i2} - \beta_{i1})^2 \leq \varepsilon \Delta\varphi$$

over each $R_s^{(k)} = \{x: \beta_{i1} \leq x_i \leq \beta_{i2},\ i = 1, \ldots, n\}$ for $s = 1, \ldots, 2^{k-1}$. It suffices to consider only one such hyperrectangle $R_j^{(k)}$ since $\Delta\beta_i \equiv \beta_{i2} - \beta_{i1} i = 1, \ldots, n$, is the same for all $R_s^{(k)}$ for $s = 1, \ldots, 2^{k-1}$ because no regions were ever eliminated (see Figure 1). Hence, if the condition stated above holds over some $R_j^{(k)}$ then it holds over all $R_s^{(k)}$, $s = 1, \ldots, 2^{k-1}$.

Let $k_i$ be the number of bisections along coordinate direction $e_i$ required to reduce the original hyperrectangle $R_1^{(1)}$ to the current hyperrectangle $R_j^{(k)}$. As an example, $k_i$ is the number of bisections of $e_i$ along the path $R_1^{(1)} \to R_1^{(2)} \to R_2^{(3)} \to R_3^{(4)} \to \cdots \to R_j^{(k)}$ in Figure 2. Note that $k$ is the number of levels in the complete binary tree (complete since no pruning or eliminations occur). Also, $k-1$ represents the sum of the number of bisections in each direction along the path from $R_1^{(1)}$ to $R_j^{(k)}$. Consider now the $i$th term of the error

$$\tfrac{1}{8}\lambda_i (\beta_{i2} - \beta_{i1})^2 = \tfrac{1}{8}\lambda_i (\Delta\beta_i)^2 = \tfrac{1}{8}\lambda_i \left(\frac{\beta_i}{2^{k_i}}\right)^2$$

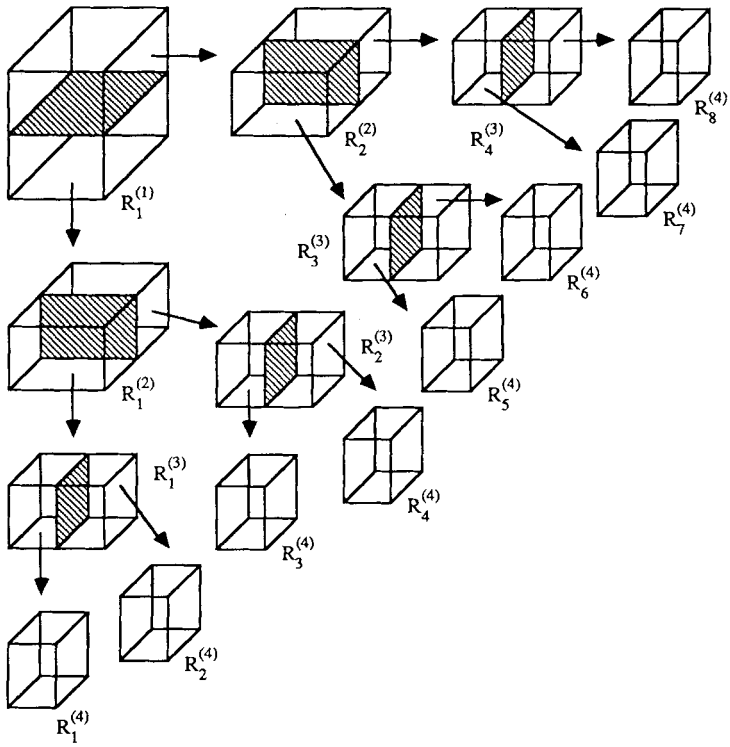


Fig. 1. A tree of bisected subproblems.

where $\beta_i$ is the optimal solution to the problem MCR in step 2 of A1g2. Hence, if

$$\tfrac{1}{8}\lambda_i\beta_i^2/2^{2k_i} \leqslant \varepsilon\Delta\varphi/n \quad \text{for } i=1,\ldots,n$$

then $E(x) \leqslant \varepsilon\Delta\varphi$ over $R_j^{(k)}$ as desired. Thus, we require that $k_i$ be the least integer such that

$$k_i \geqslant \log_2\left(\frac{n\lambda_i\beta_i^2}{8\varepsilon\Delta\varphi}\right)^{1/2} \quad \text{for } i=1,\ldots,n.$$

It follows that the total number of bisections along the path from $R_1^{(1)}$ to $R_j^{(k)}$ is bounded above by

$$k-1=\sum_{i=1}^{n} k_i \leqslant \sum_{i=1}^{n}\left(\log_2\left(\frac{n\lambda_i\beta_i^2}{8\varepsilon\Delta\varphi}\right)^{1/2}+1\right).$$

Since $k$ represents the number of levels in the complete binary tree, then the total number of bisections required for the algorithm to terminate is $2^{k-1}-1$. It can then

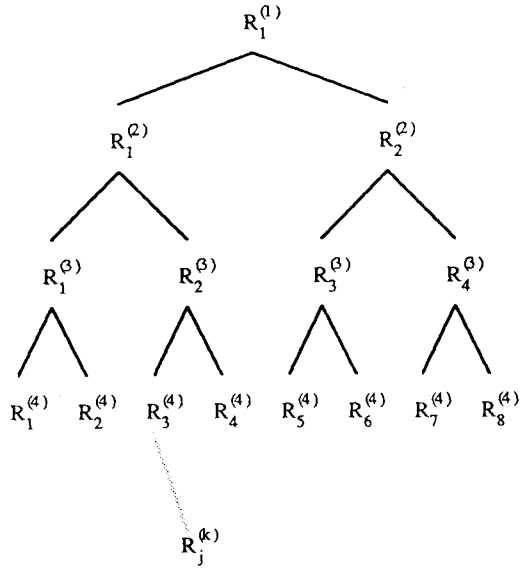


Fig. 2. The worst case expansion of the tree.

be shown that the total number of bisections required for termination is bounded above by

$$\#\text{bisections} \leq \prod_{i=1}^{n} \left( \frac{n\lambda_i \beta_i^2}{2\varepsilon \Delta \varphi} \right)^{1/2} - 1.$$

Using the error bounds of Section 3, the worst case number of bisections is also bounded above by

$$\#\text{bisections} < \left( \frac{4n\sigma(\rho, \eta)}{\varepsilon} \right)^{n/2}.$$

In addition, the total number of subproblems in the tree (i.e. calls to A1g1) is $2^k - 1$ and, therefore, is bounded above by

$$\#\text{subprobs} \leq 2 \prod_{i=1}^{n} \left( \frac{n\lambda_i \beta_i^2}{2\varepsilon \Delta \varphi} \right)^{1/2} - 1 < 2 \left( \frac{4n\sigma(\rho, \eta)}{\varepsilon} \right)^{n/2}.$$

Hence, the total number of linear programs solved is bounded above by

$$\#\text{LPs} \leq 2(n+1) \prod_{i=1}^{n} \left( \frac{n\lambda_i \beta_i^2}{2\varepsilon \Delta \varphi} \right)^{1/2} - n + 1 < 2(n+1) \left( \frac{4n\sigma(\rho, \eta)}{\varepsilon} \right)^{n/2}.$$

Since $x^* \in R_x$, the initial bounding hyperrectangle, then $x^* \in R_j^{(k)}$ for some $j \in \{1, 2, \ldots, 2^{k-1}\}$. By the analysis above, $\psi_s^{(k)} - \Gamma_s^{(k)} \leq \varepsilon \Delta \varphi$ for all $s = 1, 2, \ldots, 2^{k-1}$ so that $\psi_j^{(k)} - \Gamma_j^{(k)} \leq \varepsilon \Delta \varphi$. Note that since $x^* \in R_j^{(k)}$ we get $\Gamma_j^{(k)} \leq \psi^* \leq \psi_j^{(k)}$. By step 5.2

of A1g2, the pair $(k, j)$ is put into the solution set $S$, and since $\psi \equiv \min \psi_r^{(s)}$ for all $(s, r) \in S$ we get that $\psi - \psi^* \leq \psi_j^{(k)} - \psi^* \leq \psi_j^{(k)} - \Gamma_j^{(k)} \leq \varepsilon \Delta \varphi$ which proves the

**Theorem 4.** *The solution vertex $v$ with associated function value $\psi$ provided* Alg2 *satisfies $\psi - \psi^* \leq \varepsilon \Delta \varphi$ and is obtained in a finite number of steps.*

Hence, $\psi$ is an $\varepsilon$-approximate solution to the original concave separable quadratic minimization problem.

## 10. Parallel implementation

The *speedup* achieved by a parallel algorithm running on $N$ processors is often defined as the ratio between the time taken by a given parallel computer executing the fastest serial algorithm for a problem and the time taken by that same parallel computer executing the parallel algorithm using $N$ processors (Quinn, 1987). Note that the fastest serial algorithm may be quite different from the parallel algorithm being examined. Unfortunately, there is often no consensus as to which serial algorithm is the fastest for a given class of problems (for example, mathematical programming problems such as linear programming). Hence, we use a more practical definition of speedup; that is, speedup is the ratio between the time taken by a given parallel computer executing the parallel algorithm using only one processor and the time taken by that same parallel computer executing the parallel algorithm using $N$ processors. In fact, we also discount any overhead incurred by the parallel algorithm when it is executing on only one processor (so that it more closely imitates a serial algorithm).

The algorithm is highly parallel in nature and is therefore a good candidate for parallel computation. The most obvious parallelism occurs in the solution of the multiple-cost-row linear programs (MCRLP), and in the branching and solution of the independent subproblems.

The parallelism at the MCRLP level has finer granularity than the parallelism at the subproblem level. For each subproblem there are $2kn+1$ linear programs, for some $k \geq 1$, to be minimized over the same feasible region. At each minor iteration of the algorithm performed on each subproblem, $2n$ of the linear programs (i.e. one MCRLP) can be done in parallel. In addition, an initial MCRLP which begins the solution procedure consists of $n+2$ linear programs (to be minimized over the same feasible region), and therefore can also be performed in parallel.

At the subproblem level, each subproblem at major iteration $k$ is independent of the other subproblems at that major iteration (except for the sharing of the incumbent function value $\psi_{\text{incb}}$) and is defined by its hyperrectangle $R_j^{(k)}$ and corresponding feasible region $\Omega_j^{(k)}$. Hence, the algorithm can be applied to each subproblem in parallel, but unlike the MCRLP level, we cannot guarantee that a particular number of parallel subproblems will need to be solved at each major iteration (i.e. the

number of subproblems may vary anywhere from 0 to $2^{k-1}$ at each major iteration $k$ of the algorithm). Because considering the parallelism at the subproblem level does not interfere with the parallelism at the MCRLP level, both the subproblems and the MCRLPs within each subproblem could be done in parallel.

For parallelism at the subproblem level, a speedup greater than one can be observed only for problems in which more than one level of the solution tree is expanded. Furthermore, all $N$ processors are used only when $N$ or more subproblems at the same level (major iteration) $k$ remain to be solved. On the other hand, it is also possible in this case to observe speedups greater than the number of processors (Phillips and Rosen 1987b).

For parallelism at the MCRLP level, all $N$ processors are active at each level of the solution tree ($N < n$ is assumed), to that the percent of processor utilization can be very high. For this reason, and since speedups greater than the number of processors are also possible in this case (Phillips and Rosen, 1988), the implementation of the algorithm that is presented considers *only* the parallelism at the MCRLP level.

## 11. Test problems

The algorithm was tested on three kinds of problems: small examples constructed by hand, medium size examples obtained from the literature, and large sparse randomly generated problems with no known solutions. Eleven example problems (Phillips and Rosen 1987b) were constructed by hand and tested. In addition, the five medium size problems from the literature (Rosen and van Vliet 1987) were tested to allow the direct comparison of the proposed method with two other available methods for solving problem (SQ). The other two methods compared were the stochastic approach of Rosen and van Vliet (1987) and the zero-one integer programming approach of Glinsman and Rosen (1986). Finally, the randomly generated problems were constructed so that the feasible region was nonempty and bounded, and $A_1$ and $A_2$ were sparse constraint matrices with eight nonzero elements per column (Chvátal, 1983). More precisely, the random problems generated had the following form:

$$\psi(x, y) = \theta_1 \varphi(x) + \theta_2 d^t y,$$

$$\varphi(x) = (\tfrac{1}{2}) \sum_{i=1}^{n} \lambda_i (x_i - \bar{\omega}_i)^2,$$

$$\Omega = \{(x, y): A_1 x + A_2 y \leqslant b, x \geqslant 0, y \geqslant 0\},$$

where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^k$, $\lambda \in \mathbb{R}^n$, $\bar{\omega} \in \mathbb{R}^n$, $d \in \mathbb{R}^k$, $A_1 \in \mathbb{R}^{m \times n}$, $A_2 \in \mathbb{R}^{m \times k}$, and $\theta_1$, $\theta_2 \in \mathbb{R}$. Note that this form of the quadratic function differs from that of problem SQ only by a constant. The parameters $\theta_1$ and $\theta_2$ are designed to allow scaling of the nonlinear and linear terms so that neither one totally dominates the other. For the test problems generated randomly, they were fixed at $\theta_1 = -0.001$ and $\theta_2 = 0.1$. Note

that when the components of $\theta_2 d$ are large (as compared to the components of $(\frac{1}{2})\theta_1 \lambda$) the problem is primarily a linear program, and hence much easier. The particular choices of $\theta_1$ and $\theta_2$ were selected so as to generate difficult problems. The constant $\bar{\omega} \in \mathbb{R}^n$ represents the unconstrained maximum of the quadratic function. Again, the choice of $\bar{\omega}$ was made so as to generate difficult problems. More precisely for $\bar{\omega}$ near the interior of the polytope, the problem is generally much harder since many local minima may exist. Choosing $\bar{\omega}$ exterior to the polytope tends to generate much easier problems.

## 12. Results

### *12.0. Computational results on the CRAY2*

This section describes the computational results obtained on the CRAY2 supercomputer for the example problems, medium size problems from the literature, and the randomly generated problems described in the previous section. In all of the test problems a solution vertex was recognized as the global solution if it was $\varepsilon$-approximate for $\varepsilon = 0.001$.

The CRAY2 supercomputer, located in the Minnesota Supercomputer Center, is a four processor vector MIMD supercomputer with a 4.1 ns (nanosecond) clock cycle, 256 million words of central memory, and the UNICOS operating system. The compiler used was a developmental version of the CFT77 Fortran compiler. The parallel results cited below were obtained by using all four processors of the CRAY2 in a dedicated environment.

### *12.1. Results for small and medium size problems*

The tables in this section summarize the results obtained for the example problems and the medium size problems from the literature. Table 1 presents the results obtained from testing the program sequentially on the set of hand constructed examples. In all cases, the parameters $\theta_1 = -1.0$ and $\theta_2 = 1.0$ were used, and a solution vertex was recognized as the global solution if it was $\varepsilon$-approximate for $\varepsilon = 0.001$. Problems "example" and "prob15" required the expansion of three levels of the solution tree and the solution of five subproblems in order to obtain an $\varepsilon$-approximate solution. Problem "prob10" required the expansion of two levels of the tree and the solution of three subproblems. All other problems required only one level of the tree and the solution of one subproblem.

Table 2 compares three different algorithms on five test problems from the literature (Rosen and van Vliet, 1987). Each algorithm was tested sequentially on either the CRAY2 or the Cyber 845 (Glinsman and Rosen's algorithm only). This table clearly shows that for these concave quadratic problems, the algorithm of Phillips and Rosen with $\varepsilon = 0.001$ is the most efficient. Note that for smaller values of $\varepsilon$, the algorithm of Phillips and Rosen would perform less favorably. The algorithm

Table 1

Sizes and sequential solution times (secs) for the eleven hand-constructed test problems on the CRAY2

| name | $m, n, k$ | Time |
|---|---|---|
| example | 5, 2, 0 | 0.026 |
| prob1 | 5, 6, 0 | 0.022 |
| prob2 | 5, 6, 0 | 0.020 |
| prob3 | 5, 6, 0 | 0.026 |
| prob10 | 4, 2, 0 | 0.017 |
| prob11 | 4, 3, 0 | 0.015 |
| prob12 | 4, 3, 0 | 0.014 |
| prob13 | 10, 3, 0 | 0.022 |
| prob14 | 10, 3, 0 | 0.020 |
| prob15 | 4, 4, 0 | 0.029 |
| prob20 | 9, 2, 1 | 0.023 |

Table 2

Comparison of solution times (secs) of three different algorithms on the CRAY2 for the five test problems from the literature

| name\algorithm | $m, n, k$ | P&R87 | R&vV87 | G&R86[a] |
|---|---|---|---|---|
| R&vV87.1 | 5, 10, 0 | 0.11 | 1.50 | 10.34 |
| R&vV87.2 | 10, 20, 0 | 1.43 | 18.69 | 20.47 |
| R&vV87.3 | 20, 20, 0 | 3.21 | 73.84 | 211.87 |
| R&vV87.7 | 20, 30, 0 | 9.16 | 118.76 | 417.26 |
| R&vV87.8 | 20, 40, 0 | 16.52 | 195.53 | 328.55 |

[a] Time on the Cyber 845.

of Rosen and van Vliet, although not as efficient as the algorithm of Phillips and Rosen for the cases tested, applies to the broader class of general differentiable concave problems, and hence may be more suited to non-quadratic problems. In addition, its performance could be enhanced if run in parallel. The detailed results concerning the stochastic algorithm of Rosen and van Vliet and the 0-1 integer approach of Glinsman and Rosen are available in Rosen and van Vliet (1987) and Glinsman and Rosen (1986), respectively.

### *12.2. Results for large-scale problems*

An important property of this algorithm is that an $\varepsilon$-approximate solution is guaranteed in a finite number of steps. Considering only the first level of the solution tree (stage I), the current incumbent vertex $v_{\text{incb}}$ with corresponding function value $\psi_{\text{incb}}$ is clearly not guaranteed to be $\varepsilon$-approximate. But computational experience has shown that $v_{\text{incb}}$ is in many cases either the global solution vertex $v^*$, or is such that the relative error $\psi_{\text{incb}} - \psi^*)/\Delta\varphi$ is small. In addition, it has been computationally observed that the time required to complete stage I as a function of $k$ is

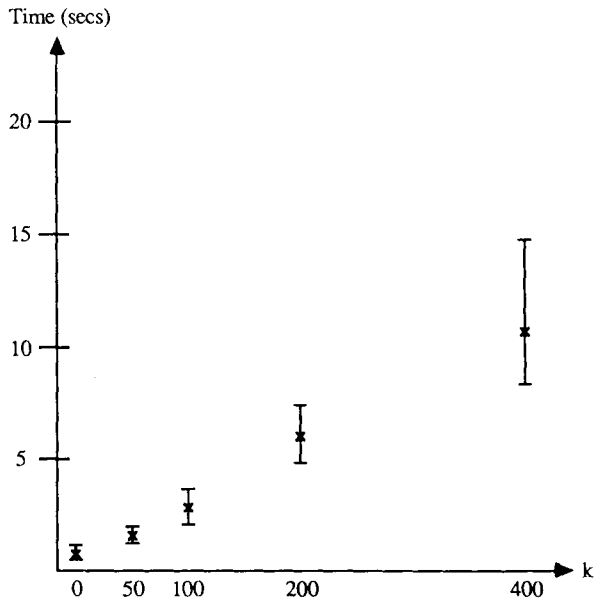


Fig. 3. Time (sequential) vs $k$ with $m = 20$ and $n = 25$ on the CRAY2. Statistics for stage I of the solution only. (10 problems of each size and $\varepsilon \approx 0.1$).

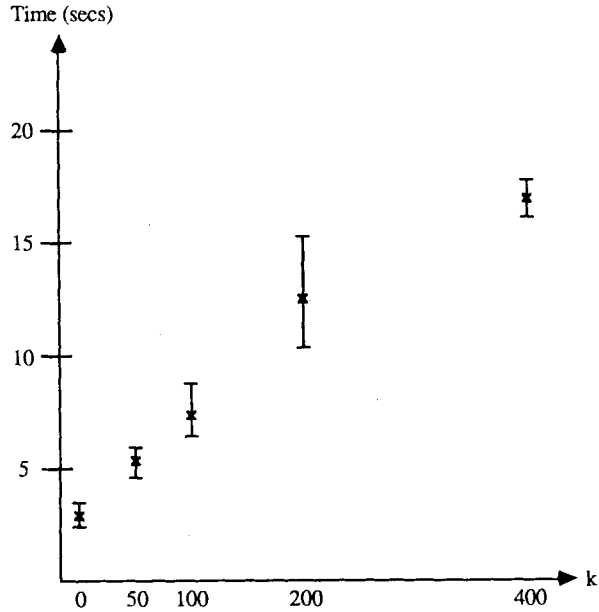


Fig. 4. Time (sequential) vs $k$ with $m = 20$ and $n = 50$ on the CRAY2. Statistics for stage I of the solution only. (10 problems of each size and $\varepsilon \approx 0.1$)

approximately linear. Figure 3 demonstrates this by showing the time (minimum, maximum, and average) required to complete stage I for $m = 20$, $n = 25$, and various values of $k$. Figure 4 is similar for $m = 20$ and $n = 50$. Further results concerning solutions obtained by stage I alone can be found in Stuart, Rosen, and Phillips (1988) for problems with $m$ as large as 160, $n$ as large as 100, and $k$ as large as 400.

As stated earlier, in all of the randomly generated test problems a solution vertex was recognized as the global solution if it was $\varepsilon$-approximate for $\varepsilon = 0.001$. Figure 5 displays the number of CPU seconds required on the CRAY2 to obtain

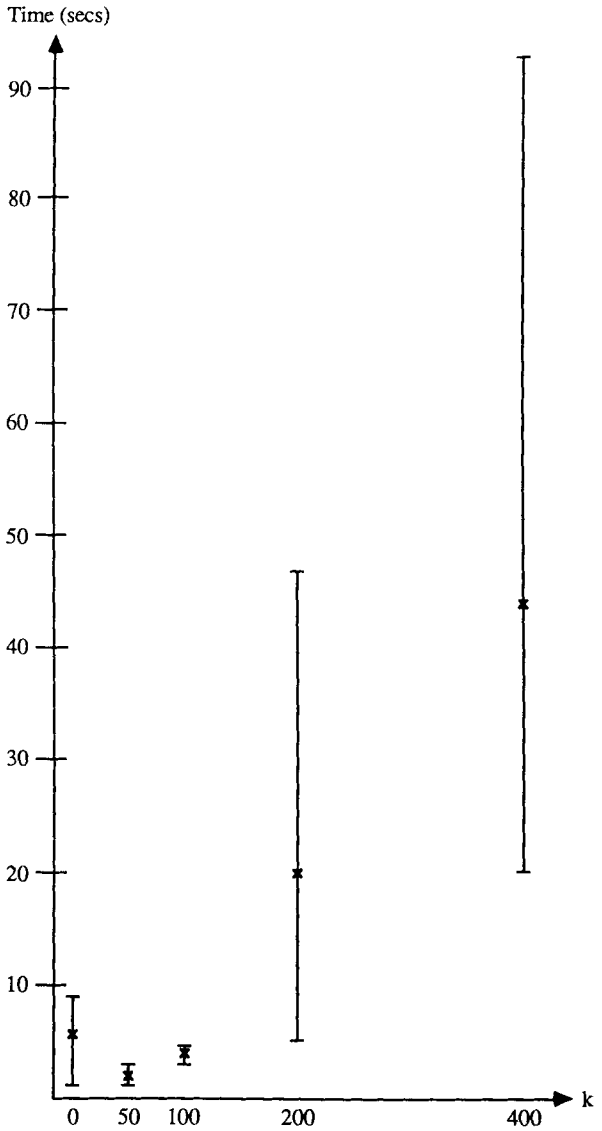


Fig. 5. Time (sequential) vs $k$ with $m = 20$ and $n = 25$ on the CRAY2. Statistics for stages I and II of the solution. (10 problems of each size and $\varepsilon = 0.001$).

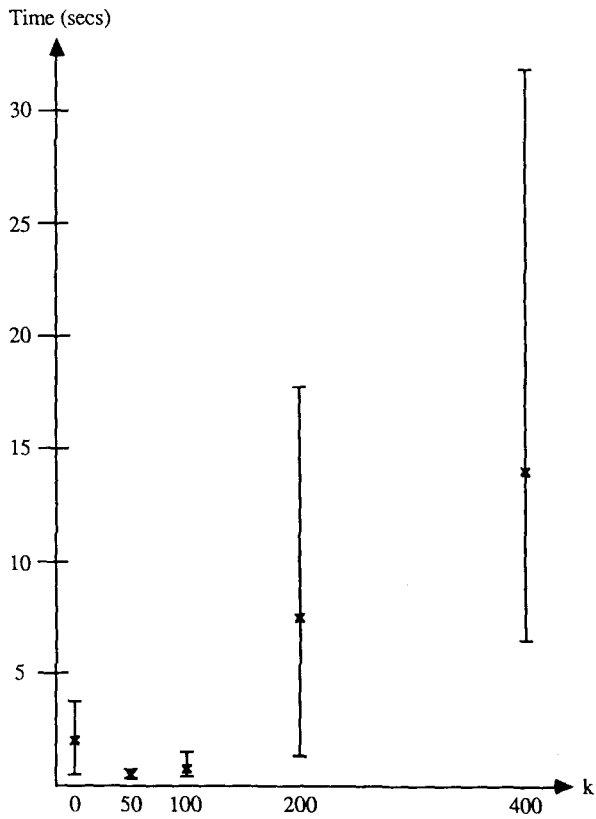


Fig. 6. Time (parallel) vs $k$ with $m = 20$ and $n = 25$ on the CRAY2. Statistics for stages I and II of the solution. (10 problems of each size and $\varepsilon = 0.001$).

$\varepsilon$-approximate solutions for large sparse randomly generated problems solved sequentially (i.e. on one processor) with $m = 20$ constraints, $n = 25$ nonlinear variables, and the number of linear variables $k$ varying from 0 to 400. For each value of $k$, the range of solution times are indicated by the bar, and the average solution time is denoted by $X$. Notice that the case $k = 0$ is significantly more difficult than the cases $k = 50$ and 100. This is not surprising since for $k = 0$, the problem to be solved is a purely nonlinear one. As $k$ increases moderately (to 50 and 100) the presence of the linear terms make the problem more linear and hence somewhat easier. But as $k$ gets large (such as $k = 200$ and 400) the total number of problem variables increases and again the problem becomes more difficult. Figure 6 displays the corresponding parallel solution times (minimum, maximum, and average) for the same set of test problems.

Figure 7 displays the number of CPU seconds required on the CRAY2 to solve randomly generated problems sequentially with 20 constraints, 50 nonlinear vari-

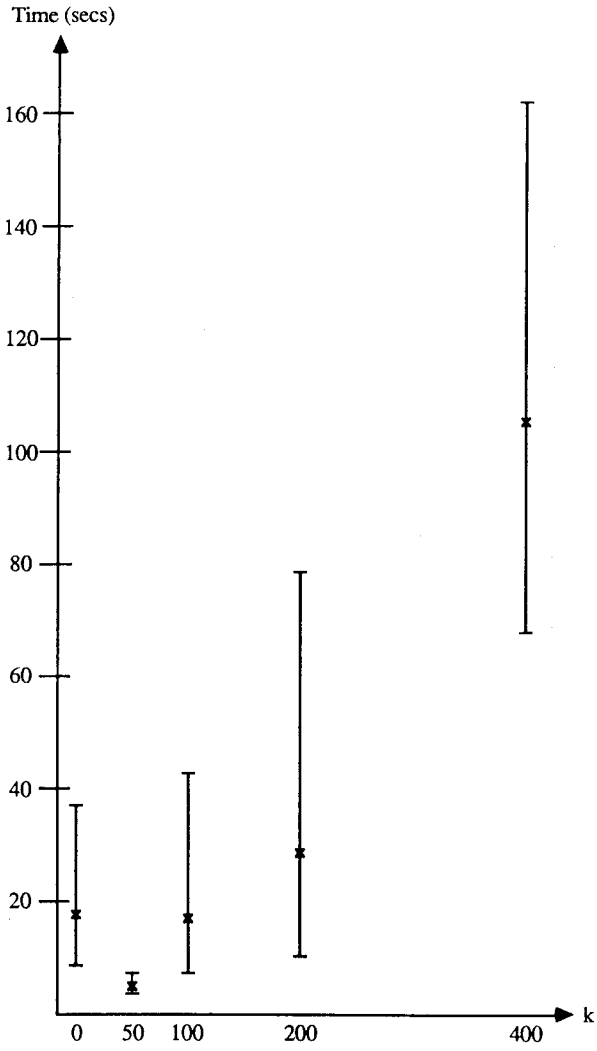


Fig. 7. Time (sequential) vs $k$ with $m = 20$ and $n = 50$ on the CRAY2. Statistics for stages I and II of the solution. (10 problems of each size and $\varepsilon = 0.001$)

ables, and the number of linear variables varying from 0 to 400. Figure 8 shows the corresponding parallel solution times.

Figures 9 and 10 display the speedup obtained by the parallel implementation of the algorithm over the sequential version. It is clear from these figures that in certain cases speedups greater than four (the number of processors used) can actually be observed. Furthermore, in some cases an average speedup of four can be observed.

Although the algorithm is guaranteed to obtain an $\varepsilon$-approximate solution in a finite number of steps, this guaranteed bound (worst case), on the linear programs

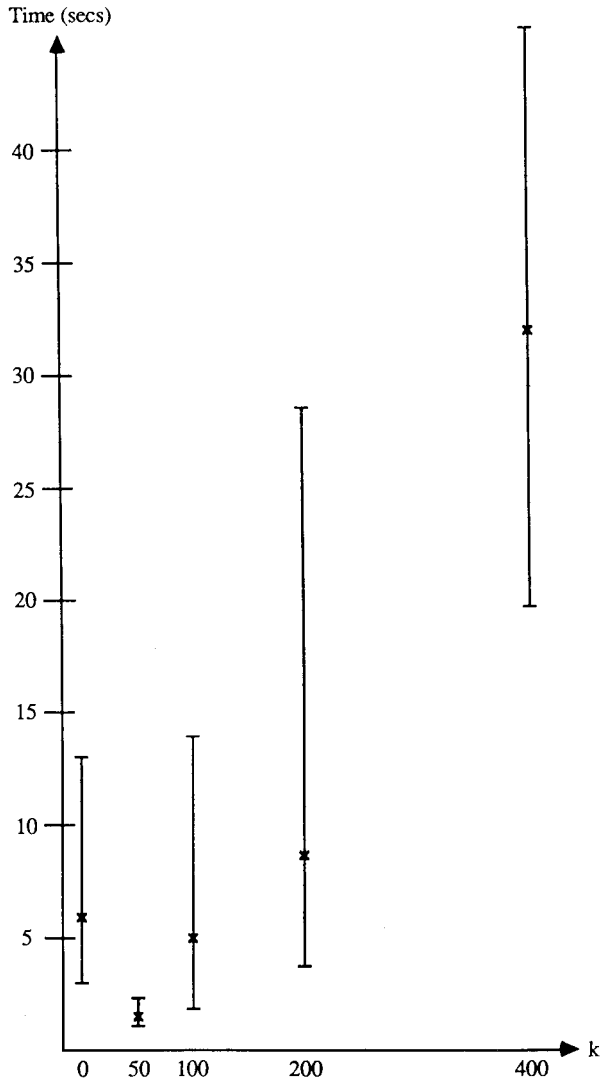


Fig. 8. Time (parallel) vs $k$ with $m = 20$ and $n = 50$ on the CRAY2. Statistics for stages I and II of the solution. (10 problems of each size and $\varepsilon = 0.001$).

solved, grows exponentially as $n(4n/\varepsilon)^{n/2}$ (see Section 9). Obviously if this represented the average computational performance of the algorithm, it would be of no practical interest. Fortunately this is not the case. Figure 11 displays the number of linear programs (minimum, maximum, and average) solved in order to obtain an $\varepsilon$-approximate solution for the randomly generated test problems. Whereas the theoretical upper bound predicts that at most $1.645 \times 10^{64}$ linear programs would need to be solved (in the worst case) for a problem of size $n = 25$ (independent of

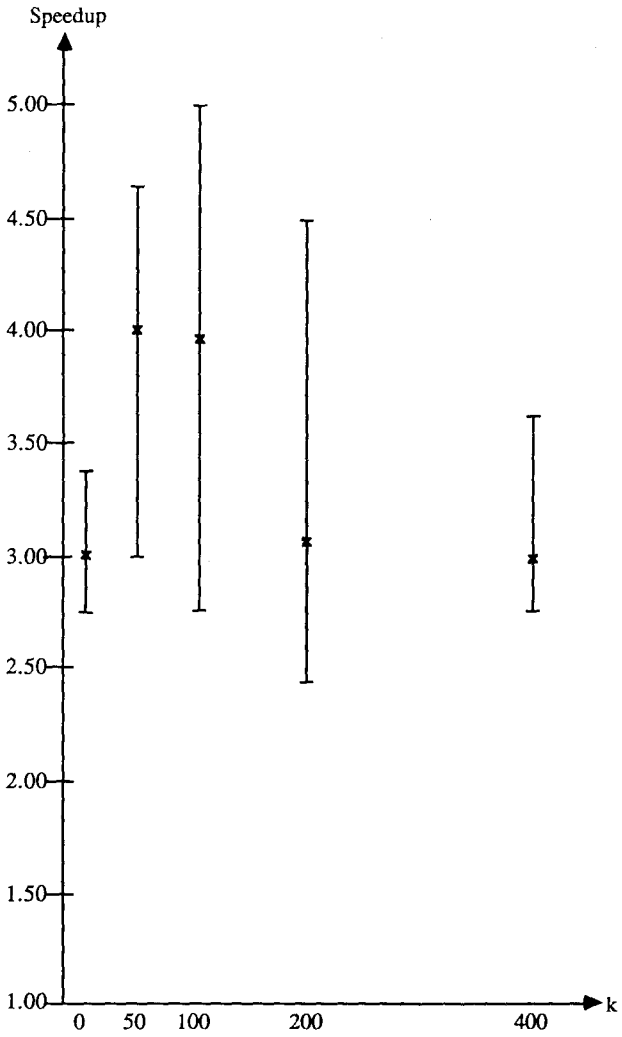


Fig. 9. Speedup vs $k$ with $m = 20$ and $n = 25$ on the CRAY2. (10 problems and $\varepsilon = 0.001$)

$k$), computational experience indicates that no more than 6300 linear programs were required for any particular problem with $n = 25$. Results for the case $n = 50$ are similar and therefore not reported.

Finally, because the number of linear programs may be large if an $\varepsilon$-approximate solution is desired, then the number of pivots required to obtain the solution may become prohibitive. Fortunately, because the multiple-cost-row approach to solving the linear programs is efficient, the average number of pivots per linear program remains small. In fact, the computational results indicate that the average number

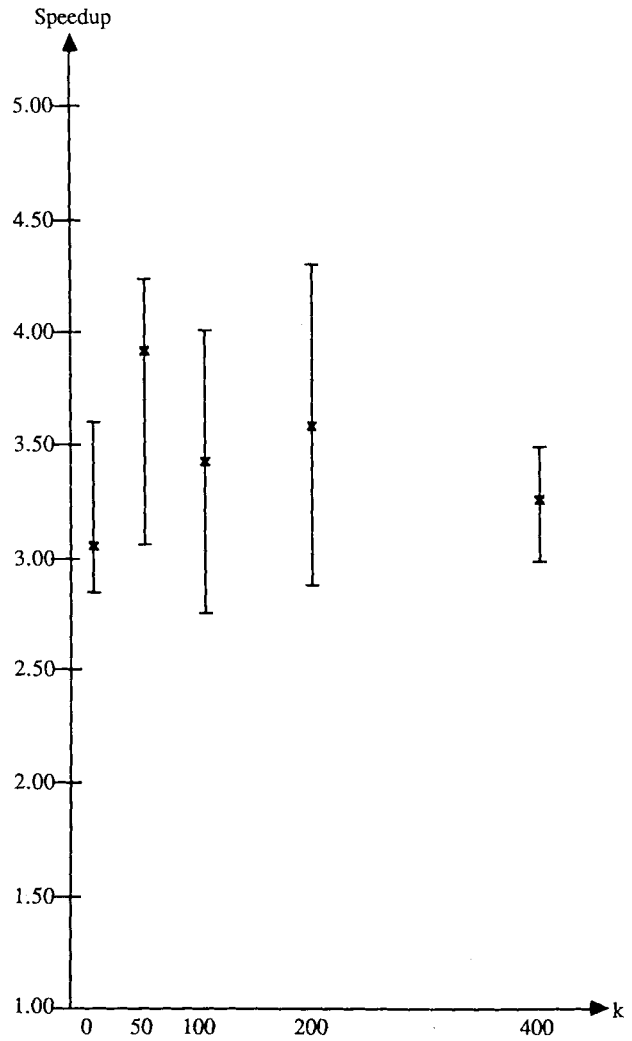


Fig. 10. Speedup vs $k$ with $m = 20$ and $n = 50$ on the CRAY2. (10 problems and $\varepsilon = 0.001$)

of pivots per linear program was never more than 36. More precisely, in cases for which at least two levels of the solution tree were required to obtain an $\varepsilon$-approximate solution (i.e. stage I alone did not provide an $\varepsilon$-approximate solution), the average number of pivots per linear program never exceeded ten. Only in the cases for which stage I alone was sufficient to obtain the $\varepsilon$-approximate solution did this average exceed ten (but it never exceeded 36). Hence, although a potentially large number of linear programs may need to be solved to obtain an $\varepsilon$-approximate solution for a given problem, the number of pivots per linear program is small, and therefore the overall solution time remains reasonable.

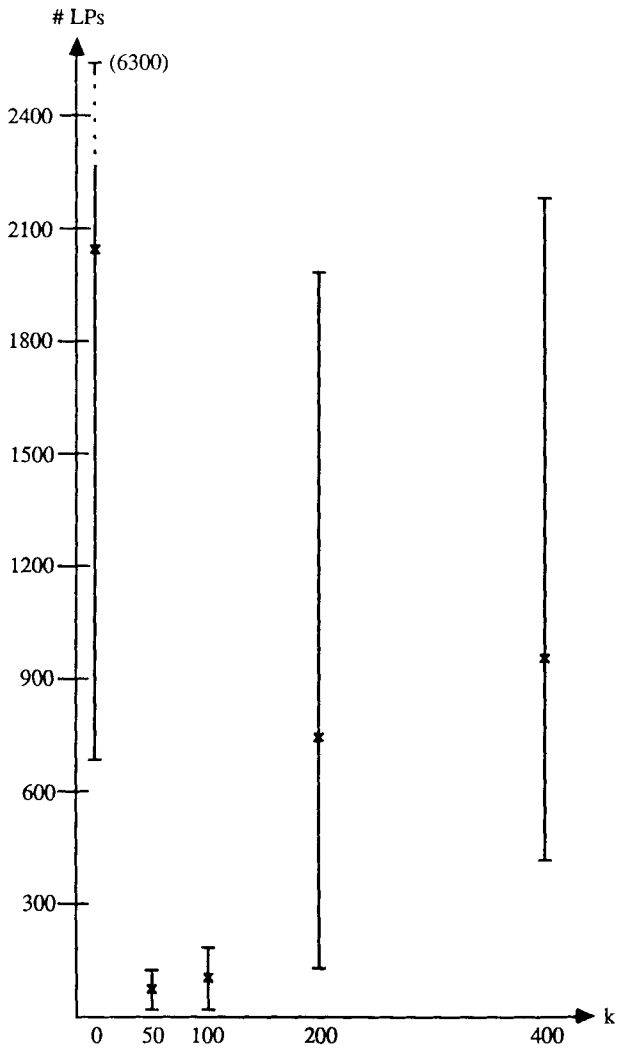


Fig. 11. Number of linear programs per problem vs $k$ with $m = 20$ and $n = 25$ on the CRAY2. (10 problems of each size and $\varepsilon = 0.001$).

## 13. Conclusions

The computational results included in this paper demonstrate that the algorithm is an efficient method for finding an $\varepsilon$-approximate solution for large-scale concave quadratic global minimization problems. Furthermore, for shared memory parallel machines, the method is very effective in obtaining average speedups approaching the number of processors.

## Acknowledgments

This research was supported in part by the National Science Foundation grants DCR-8405489 and DCR-8420935, the Air Force Office of Scientific Research grant AFOSR 87-017, by the Minnesota Supercomputer Institute, and Cray Research, Inc. In addition, A.T. Phillips' research was supported by an Office of Naval Research (ONR) Graduate Fellowship.

## References

V. Chvátal, *Linear Programming* (W.H. Freeman and Company, New York, 1983).

J.E. Falk and K.R. Hoffman, "A successive underestimation method for concave minimization problems," *Mathematics of Operations Research* 1(3) (1976) 251-259.

D.H. Glinsman and J.B. Rosen, "Constrained concave quadratic global minimization by integer programming, "Technical Report 86-37, Computer Science Department, University of Minnesota, Minneapolis, MN (1986).

R. Horst, "An algorithm for nonconvex programming problems," *Mathematical Programming* 10 (1976) 312-321.

B. Kalantari, "Large scale global minimization of linearly constrained concave quadratic functions and related problems," Ph.D. diss., University of Minnesota, Minneapolis, MN (1984).

P.M. Pardalos and J.B. Rosen, "Methods for global concave minimization: A bibliographic survey," *SIAM Review* 28(3) (1986) 367-379.

P.M. Pardalos and J.B. Rosen, "Constrained global optimization: Algorithms and applications," in: G. Goos and J. Hartmans, eds., *Lecture Notes in Computer Science* 268, Springer-Verlag, Berlin 1987).

A.T. Phillips and and J.B. Rosen, "Anomalous acceleration in parallel linear programming," Technical Report UMSI 88/58, University of Minnesota Supercomputer Institute, Computer Science Department, University of Minnesota, Minneapolis, MN (1988).

A.T. Phillips and J.B. Rosen, "A parallel algorithm for constrained concave quadratic global minimization," Technical Report 87-48, Computer Science Department, University of Minnesota, Minneapolis, MN (1987a).

A.T. Phillips and J.B. Rosen, "A parallel algorithm for constrained concave quadratic global minimization: Computational aspects, "Technical Report UMSI 87/101, University of Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN (1987b).

M.J. Quinn, *Designing Efficient Algorithms for Parallel Computers* (McGraw-Hill, New York, 1987).

J.B. Rosen, "Global minimization of a linearly constrained concave function by partition of feasible domain," *Mathematics of Operations Research* 8(2) 1987 215-230.

J.B. Rosen and P.M. Pardalos, "Global minimization of large-scale constrained concave quadratic problems by separable programming," *Mathematical Programming* 34(2) (1986) 163-174.

J.B. Rosen and M. van Vliet, "A parallel stochastic method for the constrained concave global minimization problem," Technical Report 87-31, Computer Science Department, University of Minnesota, Minneapolis, MN (1987).

E. Stuart, J.B. Rosen and A.T. Phillips, "Fast approximate solution to constrained global minimization problems," Technical Report 88-9, Computer Science Department, University of Minnesota, Minneapolis, MN (1988).