Journal of Global Optimization **6:** 215–230, 1995.
© 1995 *Kluwer Academic Publishers. Printed in the Netherlands.*

A Relaxation Method for Nonconvex Quadratically Constrained Quadratic Programs*

FAIZ A. AL-KHAYYAL
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, U.S.A.

CHRISTIAN LARSEN
Department of Management, University of Odense, Odense, Denmark

and

TIMOTHY VAN VOORHIS
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, U.S.A.

(Received: 3 March 1993; accepted: 18 November 1994)

Abstract. We present an algorithm for finding approximate global solutions to quadratically constrained quadratic programming problems. The method is based on outer approximation (linearization) and branch and bound with linear programming subproblems. When the feasible set is nonconvex, the infinite process can be terminated with an approximate (possibly infeasible) optimal solution. We provide error bounds that can be used to ensure stopping within a prespecified feasibility tolerance. A numerical example illustrates the procedure. Computational experiments with an implementation of the procedure are reported on bilinearly constrained test problems with up to sixteen decision variables and eight constraints.

Key words: Quadratic constraints, quadratic programming, relaxation linearization technique, branch and bound, global optimization.

1. Introduction

In the petrochemical industry, feedstock streams are pooled to yield a blended product whose qualities are quadratic functions of the component streams. The problem of determining how much of each type of feedstock to purchase from each source and what the composition of the feedstocks should be for the most profitable blends can be expressed as a mathematical programming problem with a linear objective function and quadratic constraints [11, 13, 27]. Placement and layout problems in integrated circuit design involve quadratic restrictions that are usually satisfied by heuristic techniques [5, 7]. In recent years, the economics of manufacturing and production has led to an increase in the modularization of product subassemblies [2, 14, 22]. The modular design problem, in its simplest form, involves indefinite quadratic functions in the constraints [14, 22, 2, 9]. Other applications that give rise to mathematical models with quadratic constraints include chance-constrained programming [25], production planning [22], design

* This research was supported in part by National Science Foundation Grant DDM-91-14489.

of heat exchanger networks [27], minimax location problems [17], and certain stochastic games [10].

Recently, several algorithms for finding global solutions to special cases have been proposed. Notable among them are the papers of Foulds *et al.* [13], Sherali and Tuncbilek [23], and Visweswaran and Floudas [26]. Foulds *et al.* [13] extend the method of Al-Khayyal and Falk [3] for linearly constrained bilinear programs to handle bilinear constraints along the lines outlined in Al-Khayyal [2]. The approach is based on convex underestimation of all nonlinear terms over a (hyper)rectangular domain, and successively tightening the underestimates by partitioning the domain. Linear programming subproblems are solved with branch-and-bound employed for the bookkeeping. Sherali and Tuncbilek [23] consider general nonconvex polynomial programming problems and generate implied constraints, from the hyperrectangular set bounding the variables, that involve polynomial terms which are linearized (together with the original polynomial terms in the problem) by defining a new variable for each distinct nonlinear term. The resulting linear programming relaxation provides the bounds in a branch-and-bound setting where successive partitions of the hyperrectangle produce tighter bounds over the subsets of the original feasible region. In contrast, Floudas and Visweswaran [26, 27] employ generalized Benders decomposition and solve the subproblems by a dualization and relaxation scheme. The method is designed to handle general nonlinear functions, but the subproblems are considerably more tractable in the quadratic case.

In this paper, we will consider the general quadratic program (GQP):

$$\begin{aligned} \min\, & x^T Q^0 x + c^0 x \\ \text{s.t. } & x^T Q^p x + c^p x \le b^p \qquad p = 1, 2, \ldots, P \\ & Ax \le d \\ & \ell \le x \le L \end{aligned}$$

where $Q^0, \ldots, Q^P$ are real $n \times n$ matrices (possibly indefinite), $c^0, \ldots, c^P, \ell, L \in \mathbb{R}^n$ with $\ell \le L, b^1, \ldots, b^P \in \mathbb{R}$, A is a real $m \times n$ matrix, and $d \in \mathbb{R}^m$. We denote $S = \{x \in \mathbb{R}^n : Ax \le d\}, \Omega = \{x \in \mathbb{R}^n : \ell \le x \le L\}$ and $f(x) = x^T Q^0 x + c^0 x$. We assume the feasible set of GQP is nonempty so that an optimal solution exists.

Early global optimization procedures for this problem considered the special case when x is restricted to be binary (see [15] and references therein). For the case of no quadratic constraints ($P = 0$), Erenguc and Benson [8] and Pardalos and Rosen [20] decompose the objective function into separable form by determining all eigenvalues and eigenvectors of Q^0, and then solve the (transformed) separable problem by branch and bound. Clearly, this approach cannot be extended to the case where $P \ge 1$.

In this paper, we extend the method of Al-Khayyal and Larsen [4] for the case $P = 0$ to the case $P \ge 1$ by incorporating upper as well as lower linearizations of all bilinear terms. Our algorithm is closely related to (and independent of) the work of Foulds *et al.* [13] and can be interpreted as a special case of the method proposed by Sherali and Tuncbilek [23]. A linear programming relaxation of GQP is derived in Section 2 which is used to calculate a lower bound on the optimal objective value of GQP for a given Ω. This subproblem is used in Section 3 to develop a branch and bound algorithm for finding a global solution of GQP. As in the earlier work, branching amounts to subdividing Ω into smaller hyperrectangles. We include a concise proof that any convergent subsequence of points generated by the algorithm will converge to an optimal solution. In Section 4, a small numerical example is solved to illustrate the procedure. Section 5 reports on results of computational experiments using an implementation of the algorithm.

2. A Linear Programming Relaxation

In this section we construct a linear programming relaxation of GQP. First, we rewrite it as a generalized bilinear program GBP.

$$\begin{aligned} \min\, & x^T y^0 + c^0 x & & \\ \text{s.t. } & x^T y^p + c^p x \le b^p & & p = 1, \dots, P \\ & y^p = Q^p x & & p = 0, \dots, P \\ & m^p \le y^p \le M^p & & p = 0, \dots, P \\ & x \in S \cap \Omega. & & \end{aligned}$$

We have introduced extra variables $y^p = Q^p x$ for $p = 0, \dots, P$ in order to make the objective and the constraints bilinear. The vectors m^p and M^p are bounds on y^p, which are calculated as follows:

$$m_i^p = \min\{Q_i^p x : x \in \Omega\} = \sum_{j=1}^n \min(Q_{ij}^p \ell_j, Q_{ij}^p L_j) \qquad \begin{aligned} i &= 1, \dots, n, \\ p &= 0, \dots, P \end{aligned}$$

$$M_i^p = \max\{Q_i^p x : x \in \Omega\} = \sum_{j=1}^n \max(Q_{ij}^p \ell_j, Q_{ij}^p L_j) \qquad \begin{aligned} i &= 1, \dots, n, \\ p &= 0, \dots, P \end{aligned}$$

where Q_i^p is row i and Q_{ij}^p is element (i, j) of matrix Q^p.

It is well known, see Al-Khayyal and Falk [3] and Al-Khayyal [1], that the convex envelope of the two dimensional bilinear function $g(x, y) = xy$ on the hyperrectangle $\{(x, y) \in \mathbb{R}^2 : \ell \le x \le L, m \le y \le M\}$ is $\varphi(x, y) = \max\{\ell y + mx - \ell m, Ly + Mx - LM\}$ and that $\psi(x, y) = \min\{\ell y + Mx - \ell M, Ly + mx - Lm\}$ is the corresponding concave envelope.

By using this result a linear programming relaxation of GBP is

$$\min \quad \sum_{j=1}^n t_j^0 + c^0 x$$

$$
\begin{array}{lll}
\text{s.t.} & & \\
(LP_{GBP}) & t_j^p \geq \ell_j y_j^p + m_j^p x_j - \ell_j m_j^p & j=1,\dots,n,\ p=0,\dots,P \\
& t_j^p \geq L_j y_j^p + M_j^p x_j - L_j M_j^p & j=1,\dots,n,\ p=0,\dots,P \\
& t_j^p \leq \ell_j y_j^p + M_j^p x_j - \ell_j M_j^p & j=1,\dots,n,\ p=1,\dots,P \\
& t_j^p \leq L_j y_j^p + m_j^p x_j - L_j m_j^p & j=1,\dots,n,\ p=1,\dots,P \\
& \sum_{j=1}^n t_j^p + c^p x \leq b^p & p=1,\dots,P \\
& y^p = Q^p x & p=0,\dots,P \\
& x \in S \cap \Omega. &
\end{array}
$$

Here we have used extra variables t_j^p for $j = 1,\dots,n$ and $p = 0,\dots,P$ to approximate $x_j y_j^p$ by using the bounds on $x_j y_j^p$. Substituting back to the x space by $y^p = Q^p x$ we reach a linear programming relaxation of GQP,

$$
\begin{array}{lll}
\min & \sum_{j=1}^n t_j^0 + c^0 x & \\
\text{s.t.} & & \\
(LP_{GQP}) & t_j^p \geq \ell_j Q_j^p x + m_j^p x_j - \ell_j m_j^p & j=1,\dots,n,\ p=0,\dots,P \\
& t_j^p \geq L_j Q_j^p x + M_j^p x_j - L_j M_j^p & j=1,\dots,n,\ p=0,\dots,P \\
& t_j^p \leq \ell_j Q_j^p x + M_j^p x_j - \ell_j M_j^p & j=1,\dots,n,\ p=1,\dots,P \\
& t_j^p \leq L_j Q_j^p x + m_j^p x_j - L_j m_j^p & j=1,\dots,n,\ p=1,\dots,P \\
& \sum_{j=1}^n t_j^p + c^p x \leq b^p & p=1,\dots,P \\
& x \in S \cap \Omega. &
\end{array}
$$

For brevity, we will define $(\varphi_\Omega)_j^p(x) = \max\{\ell_j Q_j^p x + m_j^p x_j - \ell_j m_j^p,\ L_j Q_j^p x + M_j^p x_j - L_j M_j^p\}$, $j = 1,\dots,n, p = 0,\dots,P$ and $(\psi_\Omega)_j^p(x) = \min\{\ell_j Q_j^p x + M_j^p x_j - \ell_j M_j^p, L_j Q_j^p x + m_j^p x_j - L_j m_j^p\}$, $j = 1,\dots,n, p = 1,\dots,P$. We have put Ω as a subscript on the functions to emphasize that the basis for the derivation of these functions is the set Ω. We will put

$$
\begin{aligned}
\varphi_\Omega^p(x) &= \sum_{j=1}^n (\varphi_\Omega)_j^p(x), p = 0,\dots,P \quad \text{and} \\
\psi_\Omega^p(x) &= \sum_{j=1}^n (\psi_\Omega)_j^p(x), p = 1,\dots,P.
\end{aligned}
$$

We can now concisely write LP_{GQP} as

$$\min \quad \varphi_\Omega^0(x) + c^0 x$$

s.t.

$$(REL_{GQP}) \quad \begin{array}{ll} (\varphi_\Omega)_j^p(x) \le t_j^p \le (\psi_\Omega)_j^p(x) & j = 1, \dots, n,\ p = 1, \dots, P \\ \sum_{j=1}^{n} t_j^p + c^p x \le b^p & p = 1, \dots, P \\ x \in S \cap \Omega. & \end{array}$$

An important result for our convergence proof is the following.

LEMMA 1. *For any p,* $\max\{\psi_\Omega^p(x) - \varphi_\Omega^p(x) : x \in \Omega\} \le 0.5(L - \ell)|Q^p|(L - \ell)$, *where* $|Q^p|$ *is the matrix with entries* $|Q_{ij}^p|$.

Proof. See the proof of Lemma 1 in Al-Khayyal and Larsen [4]. q.e.d.

We will use this lemma as a guideline for establishing a *consistent* (see Horst [18]) branching rule for the branch and bound algorithm that is developed in the next section.

REMARK. The linear programming relaxation is derived as REL_{GQP} for notational simplicity and ease of presentation. We note that this formulation allows for two t-variables to be scaled values of the same quantity. For example, if $y_j^p = \alpha x_j$ and $y_j^{p'} = \beta x_j$, then $t_j^p = \alpha x_j^2$ and $t_j^{p'} = \beta x_j^2$. In practice of course, we would let $t_j^p = x_j^2$ and replace $x_j y_j^p$ with αt_j^p and $x_j y_j^{p'}$ with βt_j^p. Procedures for reductions with the fewest additonal variables are proposed by Hansen and Jaumard [16].

3. A Branch and Bound Algorithm

We now state a standard branch and bound algorithm for solving GQP (see Horst and Tuy [19] for the theory and framework of general branch and bound algorithms). The branching is based on subdividing Ω into smaller hyperrectangles. To each node q, we assign a value $\bar{z}^q$, which is a lower bound on $f(x)$ for any x feasible in GQP and belonging to Ω^q, where Ω^q is the relevant subset of Ω. We always extract the node q with the smallest $\bar{z}^q$ value and then solve the corresponding LP relaxation REL_{GQP} with Ω replaced by Ω^q. If the optimal solution of the LP relaxation is feasible in GQP, we assign it and its $f(\cdot)$ value to (x^*, z^*): the current best feasible solution and its criteria value. Instead of writing $(\varphi_{\Omega^q})_j^p(\cdot)$ and $(\psi_{\Omega^q})_j^p(\cdot)$, we will use the simpler notation $(\varphi_q)_j^p(\cdot)$ and $(\psi_q)_j^p(\cdot)$ in the remainder of the sequel. We will also put $f^0(x) = x^T Q^0 x$. The branch and bound algorithm is as follows.

1. Initialize $\Omega^0 := \Omega$, $z^* := \infty$, $\bar{z}^0 := -\infty$, $LIST := \{(\Omega^0, \bar{z}^0)\}$ and $k := 0$.
2. If $LIST = \emptyset$, STOP, x^* is optimal in (GQP).
3. Choose and remove instance $(\Omega^q, \bar{z}^q)$ from LIST with the smallest $\bar{z}^q$ value. Solve the linear program

$$v(q) = \min(\varphi_q)^0(x) + c^0 x$$

s.t.

$$(REL^q_{GQP})\quad \begin{array}{ll} (\varphi_q)^p_j(x) \le t^p_j \le (\psi_q)^p_j(x) & j = 1,\dots,n,\ p = 1,\dots,P \\ \sum_{j=1}^{n} t^p_j + c^p x \le b^p & p = 1,\dots,P \\ x \in S \cap \Omega^q. & \end{array}$$

If REL^q_{GQP} is infeasible go to Step 2. Else denote (x^q, t^q) the optimal solution of REL^q_{GQP}.

4. (a) If $v(q) \ge z^*$, go to Step 2.
   (b) If x^q is not feasible in GQP, go to Step 5.
   (c) (x^q is feasible in GQP and $v(q) < z^*$)
   If $f(x^q) < z^*$, update $z^* := f(x^q)$, $x^* := x^q$ and delete from LIST all instances with $\bar{z}^p \ge z^*$.
   If $f^0(x^q) > (\varphi_q)^0(x^q)$ go to Step 5, else go to Step 2.
5. Subdivide Ω^q into two hyperrectangles $\Omega^{k+1} = \{x \in \Omega^q : \ell^{k+1} \le x \le L^{k+1}\}$ and $\Omega^{k+2} = \{x \in \Omega^q : \ell^{k+2} \le x \le L^{k+2}\}$ such that $\Omega^q = \Omega^{k+1} \cup \Omega^{k+2}$ and $ri(\Omega^{k+1}) \cap ri(\Omega^{k+2}) = \emptyset$.
   Let $\bar{z}^r := v(q), r = k+1, k+2$. Append to $LIST(\Omega^r, \bar{z}^r), r = k+1, k+2$, and put $k := k+2$. Go to Step 3.

For clarification purposes we only branch the father hyperrectangle Ω^q into two hyperrectangles. It is of course obvious that, in Step 5, we can branch Ω^q into as many hyperrectangles as we wish.

The branching in Step 5 must be consistent. That is, when the algorithm generates a nested sequence of hyperrectangles $\{\Omega^q\}$ we must have

$$\max\{(L^q - \ell^q)|Q^p|(L^q - \ell^q) : p = 0,\dots,P\} \downarrow 0 \text{ for } q \to \infty.$$

This is of course in order to make the approximation of the quadratic functions in GQP gradually finer.

It is quite obvious that, when finite, the algorithm will terminate with an optimal solution. For the infinite case we state and prove the following convergence theorem.

THEOREM 1. *Assume GQP has an optimal solution. Let $\{x^q, t^q\}$ be the sequence of LP optimal solutions generated in Step 3 of the algorithm. Then the limit point of any convergent subsequence of $\{x^q\}$ will be optimal for GQP.*

Proof. Denote by $v(GQP)$ the optimal value of GQP. Because we always choose the instances where $\bar{z}$ is smallest, we have that $\{\bar{z}^q\}$ is a nondecreasing sequence of points bounded above by $v(GQP)$. After eventually making a subsequence of the $\{\Omega^q, \bar{z}^q\}$, the hyperrectangles will be nested, that is $\Omega^{q+1} \subseteq \Omega^q\ \forall q$.

Due to Lemma 1

$$\begin{aligned} 0 \le f^0(x^q) - (\varphi_q)^0(x^q) &\le (\psi_q)^0(x^q) - (\varphi_q)^0(x^q) \\ &\le 0.5(L^q - \ell^q)|Q^0|(L^q - \ell^q) \end{aligned}$$

so the consistency property secures

$$\lim_{q\to\infty}(f^0(x^q) - (\varphi_q)^0(x^q)) = 0.$$

In a similar fashion we conclude from

$$(\varphi_q)^q_j(x^q) \le (t^p_j)^q \;\le\; (\psi_q)^p_j(x^q) \text{ and } (\varphi_q)^p_j(x^q) \\ \le\; x^q_j Q^p_j x^q \le (\psi_q)^p_j(x^q)\ \forall(j,p,q)$$

that

$$\lim_{q\to\infty}((t^p_j)^q - x^q_j Q^p_j x^q) = 0\ \forall(j,p).$$

Thus, for any convergent subsequence of $\{x^q, t^q\}$ with limit point $(\bar{x}, \bar{t})$, we have that $\bar{x}$ is feasible in GQP, implying that $f^0(\bar{x}) + c^0\bar{x} \ge v(GQP)$. It only remains to show that $\bar{x}$ is optimal in GQP. Here we use that, because we are looking at a path in the branch and bound tree of nested hyperrectangles, we have

$$v(q) = \bar{z}^{q+1} \le v(GQP).$$

Because $f^0(x^q) = \varphi^0_q(x^q)$ converges to 0, this implies that $f^0(\bar{x})+c^0\bar{x} \le v(GQP)$. Thus $\bar{x}$ is optimal in GQP. q.e.d.

The proof can be regarded as a realization of the general proof scheme in Horst [18] for showing convergence to a global optimum in a branch and bound context.

It is essential and not just a good heuristic, that we, in Step 3, choose the instance with the smallest lower bound $\bar{z}^q$. Violating this rule could produce an infinite sequence of infeasible points $\{x^q\}$ converging to some point which is feasible but not optimal in GQP.

If we are satisfied with an approximate solution both with respect to optimality and feasibility, we could stop further branching when

$$\max\{(L^q - \ell^q)|Q^p|(L^q - \ell^q) : p = 0, \dots, P\} \le \epsilon$$

for some presepecified tolerance ϵ.

In the following section we will demonstrate the performance of the algorithm on a small numerical example.

4. An Example

The example we considered is the following

$$\min[x_1, x_2]\begin{bmatrix} 1 & 2 \\ -1 & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [1,1]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$s.t.$

$$(E) \qquad [x_1, x_2]\begin{bmatrix} -1 & 1 \\ 3 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \le 6$$

$$x_1 + x_2 \le 4$$
$$0 \le x_1 \le 4$$
$$0 \le x_2 \le 4.$$

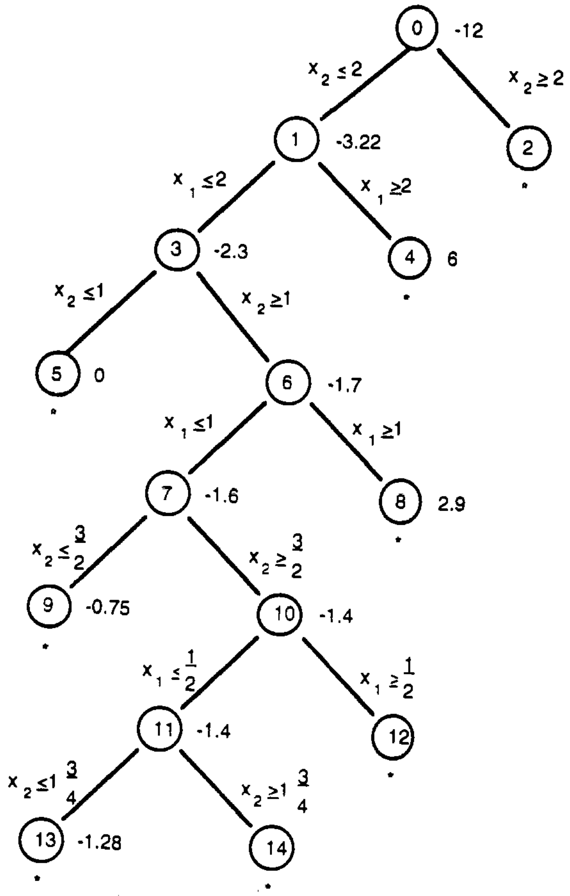

Fig. 1. Branch and bound tree for example.

The optimal solution of E is $(x_1, x_2) = (0, 3^{0.5})$.

Using the branch and bound algorithm we grew the tree shown in Figure 1. The corresponding node information is given in Table 1. In the third column x^q is the optimal x solution of the LP relaxation and if this relaxation is infeasible we put in an asterisk. In the last column we have calculated, when x^q is feasible in GQP, its objective function value $f(x^q)$.

We stopped the iteration process by concluding that $x^{13} = (0; 1.75)$ is an approximate optimal solution.

The reader can see that we accepted the solution of node 8 as feasible although this is not strictly true. However, if we had created two sons of node 8, this would not have affected the outcome of the algorithm. Due to the rule of always taking the instance with the smallest lower bound, we would still have ended at node 13 (and its successors) and would never have examined the sons of node 8. This clarifies the remark made after the proof of Theorem 1.

TABLE I. Solution of example

q	ℓ^q, L^q	x^q	$v(q)$	$f(x^q)$
0	(0, 0);(4, 4)	(2, 2)	−12	
1	(0, 0);(4, 2)	(1.56, 1.66)	−3.22	
2	(0, 2);(4, 4)	*		
3	(0, 0);(2, 2)	(0.69, 1, 62)	−2.31	
4	(2, 0);(4, 2)	(2, 0)	6	6
5	(0, 0);(2, 1)	(0, 0)	0	0
6	(0, 1);(2, 2)	(0, 1.86)	−1.71	
7	(0, 1);(1, 2)	(0, 1.82)	−1.64	
8	(1, 1);(2, 2)	(1, 1.13)	2.88	≥2.88
9	(0, 1);(1.1.5)	(0, 1.5)	−0.75	−0.75
10	(0, 1.5);(1, 2)	(0, 1.75)	−1.38	
11	(0, 1.5);(0.5, 2)	(0, 1.75)	−1.38	
12	(0.5, 1.5);(1, 2)	*		
13	(0, 1.5);(0.5, 1.75)	(0, 1.74)	−1.28	−1.29
14	(0, 1.75);(0.5, 2)	*		

5. Computational Results

To test the computational behavior of the algorithm, problems of varying sizes were randomly generated and solved. The test problems had bilinear objective functions and constraints, and are special cases of (GQP) with

$$\begin{aligned}
x^T &= (w^T, z^T) \\
Q^p &= \begin{pmatrix} 0 & \frac{1}{2}B^p \\ \frac{1}{2}B^{p^T} & 0 \end{pmatrix} && p = 0, 1, \ldots, P \\
c^p &= (c_w^p, c_z^p) && p = 0, 1, \ldots, P \\
A &= 0 \\
d &= 0 \\
\ell_i &= 0 \quad i = 1, 2, \ldots, n \\
L_i &= 20 \quad i = 1, 2, \ldots, n.
\end{aligned}$$

All problems had an even number n of variables, with $w, z \in \mathbb{R}^{n/2}$. Hence, the problems may be written in the following form:

$$\begin{aligned}
&\text{Min } w^T B^0 z + c_w^0 w + c_z^0 z \\
&\text{subject to } w^T B^p z + c_w^p w + c_z^p z \le b^p && p = 1, 2, \ldots, P \\
&\qquad 0 \le w_i \le 20 && i = 1, \ldots, n/2 \\
&\qquad 0 \le z_i \le 20 && i = 1, \ldots, n/2.
\end{aligned}$$

All problem parameters were integers randomly generated according to the following specifications. Each objective function coefficient c_i^0 and constraint coefficient

c_i^p is an integer between -10 and 10. Likewise each entry in B^0 is an integer between -10 and 10. Matrices B^p, $p = 1, 2, \ldots, P$, were produced by randomly selecting half of the entries in each row to be 0 and randomly generating an integer between -10 and 10 for the other entries in that row. Hence, the matrix density of B^p, $p = 1, 2, \ldots, P$, was taken to be 50 percent. Thus, each matrix $Q^p, p = 1, 2, \ldots, P$, has density of 25 percent. The number of constraints P was set equal to half the total number of variables in each problem. For each constraint, the right-hand-side constant b^p was a randomly generated integer between -100 and 100. The above specifications were selected to yield nontrivial, numerically stable problems having active nonlinear constraints.

Although the program used to solve the test problems closely follows the algorithm presented earlier, a few modifications were introduced for computational purposes. In particular, linearizations with the fewest number of variables were constructed in accordance with the Remark following Lemma 1. As in the algorithm, the program always chooses the branch with the best lower bound and creates and solves a linear relaxation for that branch. Branching is done in a similar, but not identical fashion as in the example of Section 4. The program seeks to reduce the maximum difference between linear variables t_{ij} and the product $w_i z_j$ which they replace. Given that some variable t_{ij} has the maximum difference $\mid t_{ij} - w_i z_j \mid$, either w_i or z_j will be chosen as the branching variable to reduce that difference. The choice between w_i and z_j is based on the fact that the convex envelope of the function $w_i z_j$ is tight at the boundaries of the rectangle under consideration. If either variable is at its upper or lower bound, then $\mid t_{ij} - w_i z_j \mid = 0$. This difference increases when both w_i and z_j move away from their respective bounds. Therefore, the variable chosen as the branching variable is the one which is farthest from either of its bounds at the solution of the linear relaxation. As in the example problem, the region of the branching variable is divided in half to form two subproblems.

As noted in Section 3, to achieve an approximate solution, branching may be stopped when an appropriate ϵ convergence criterion is satisfied. In the program, this criterion is given by the maximum difference between $\mid t_{ij} - w_i z_j \mid$, over all $i = 1, \ldots, n/2$, and $j = 1, \ldots, n/2$, being less than $\epsilon = 0.00005$. This criterion is consistent with the conditions required for the convergence proof, since

$$\max_{ij} \mid t_{ij} - w_i z_j \mid \leq \epsilon \Rightarrow \sum_i \sum_j (b_{ij}^p t_{ij} - b_{ij}^p w_i z_j) \leq \alpha_p \epsilon \quad p = 1, \ldots, P$$

(where b_{ij}^p is the entry in row i and column j of matrix B^p and α_p is a constant related to the number of variables and the magnitude of the coefficients of the bilinear terms in constraint p). This is similar to the condition necessary for the convergence proof, which relies on the difference between the original quadratic constraint terms $x^T Q^p x$ for each constraint p and their linearized replacement terms approaching zero. For the test problems, there are no more than 32 quadratic terms for any constraint and each coefficient has an absolute value of 10 or less. Hence, when the ϵ convergence criterion is satisfied, the maximum possible constraint

TABLE II. Computational performance of algorithm on test problems

n	LPs solved	CPU time (min)	Avg time (min) per LP
4	17	0.0102	0.0002
4	1	0.0015	0.0003
4	1	0.0013	0.0003
4	1	0.0015	0.0003
4	1	0.0015	0.0003
4	7	0.0042	0.0002
4	1	0.0015	0.0003
4	97	0.0573	0.0003
4	65	0.0380	0.0003
4	1	0.0015	0.0005
8	398	5.091	0.0120
8	131	1.748	0.0125
8	65	0.895	0.0130
8	85	1.040	0.0112
8	111	1.272	0.0107
8	39	0.390	0.0094
8	10	0.125	0.0119
8	133	2.025	0.0147
8	1	0.012	0.0102
8	15	0.166	0.0105
12	37	4.961	0.1331
12	187	29.656	0.1579
12	52	7.150	0.1369
12	39	6.103	0.1559
12	127	17.015	0.1334
12	129	19.539	0.1508
12	1187	145.157	0.1216
12	200	23.866	0.1187
12	2188	201.795	0.0916
12	140	17.699	0.1257
16	309	219.906	0.7102
16	2585	2415.808	0.9329
16	139	122.540	0.8800
16	174	133.121	0.7636
16	341	246.044	0.7201
16	113	86.388	0.7629
16	171	132.991	0.7760
16	310	234.248	0.7539
16	2659	2667.208	1.0014
16	31	24.772	0.7971

TABLE III. Average growth in computational effort for ten test problems

n	Avg # of LPs	Min # of LPs req	Max # of LPs req	Std Dev of # of LPs	Avg CPU time per problem (min)	Avg CPU time per LP (min)
4	19.2	1	97	33.8	0.0119	0.0003
8	98.8	1	398	116.1	1.2763	0.0121
12	428.6	37	2188	706.5	47.2940	0.1097
16	683.2	31	2659	1026.6	628.301	0.9180

violation is bounded above by a constant times 10^{-2}, where the constant depends on the exact magnitude of the constraint coefficients and is close to 1 for $n = 16$ (and decreases with n). In practice, constraints would be violated by much less than 10^{-2} if, at the solution to the LP relaxation, many original variables are at their bounds (where the linearization is tight) and the few small signed differences $(b^p_{ij}t_{ij} - b^p_{ij}w_iz_j)$ sum to nearly zero. This behavior was observed in all of the test problems in our experiments.

To ensure that no constraint was significantly violated, LP solutions (w^*, z^*, t^*) were considered feasible to the original problem only when solution (w^*, z^*) violated no constraint by more than 0.00005. Similiarly, with each $| t_{ij} - w_iz_j | \leq \epsilon$, the linear underestimate is within $\beta\epsilon$ of the objective function (i.e. $f^0(x) - \varphi^0(x) \leq \beta\epsilon$), where once again β is a constant related to the number of variables and the magnitude of the objective function coefficients. With coefficients of magnitude 10 or less and with no more than $(\frac{1}{2}n)^2$ linear terms t_{ij}, this difference was bounded above by a constant of order of magnitude 10^{-3} for smaller values of n and 10^{-2} for larger values of n and in practice was very small.

Ten test problems were generated for each $n = 4, 8, 12$, and 16. The primary result of interest was the number of linear programs solved in the course of finding the global solution to the original bilinear program. To put this number in a better perspective, we also recorded the time required to solve all of these linear programs. Each LP was solved separately, using no information generated at previous branches. Hence, these run-times do not reflect any improvements which could be achieved by taking advantage of the similarities between related branches in solving the new LPs. The tests were run on a SUN Sparc station IPC, using a rudimentary simplex code found in the book Numerical Recipes in C [21]. While faster simplex codes can be used, our primary interest was to investigate the increase in the number of LPs solved and in the computer times required to solve them as a function of n.

Results are exhibited in the tables below. Table II shows the number of linear programs and the CPU time required for each individual problem. The final column shows the total CPU time spent solving the LPs divided by the number of LPs solved. Table III shows the average number of LP subproblems solved for different values of n. To get an idea of the variance among problems with the same number

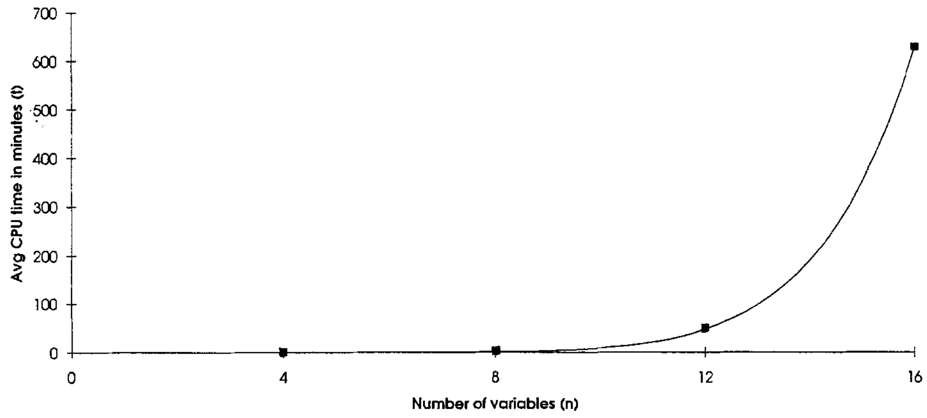

Fig. 2. Growth in CPU time as a function of n.

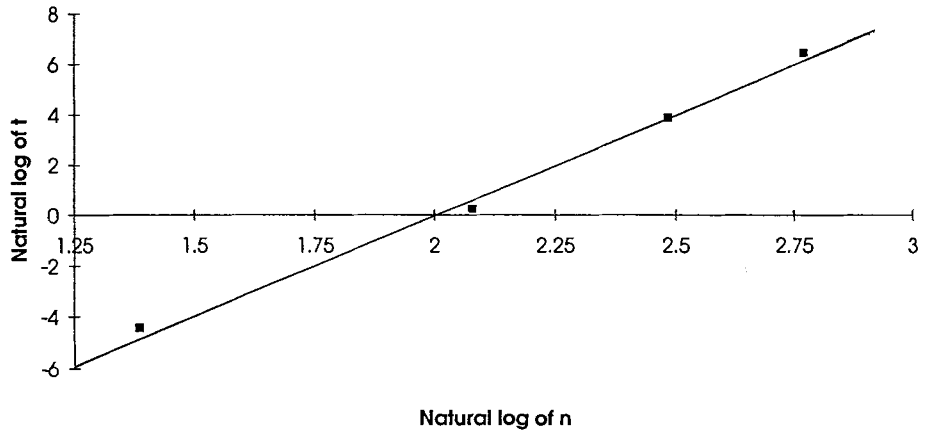

Fig. 3. Regression line for polynomial growth.

of variables, the fewest number of LPs required, the most LPs required, and the standard deviation of the number of LPs required is also given. The average computer time required to solve each problem is also reported. The final column was derived by extracting the computer time spent executing the simplex method (from total run time) for the set of test problems and dividing this by the total number of LPs solved. This shows the growth in the average computer time required to solve an LP as n increases.

Based on the above statistics, this algorithm is a reasonable approach for solving bilinearly constrained quadratic problems of up to 12 variables. As can be seen in Figure 2, the computer time required to solve these quadratic programs increases dramatically with n. Two mathematical models were considered as possible methods of quantifying this relationship. One model considered a polynomial relationship between run-time t and n so that $t = c_1 n^{c_2}$. Taking the natural logarithm of each side produces the equation $\ln t = \ln c_1 + c_2 \ln n$. Hence, this model would seem

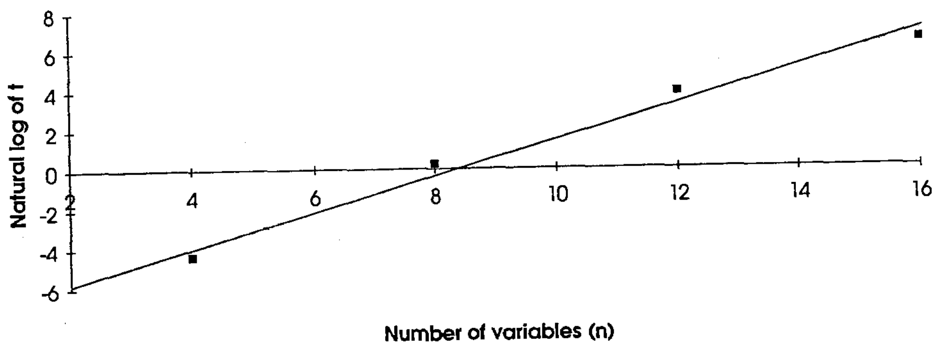

Fig. 4. Regression line for exponential growth.

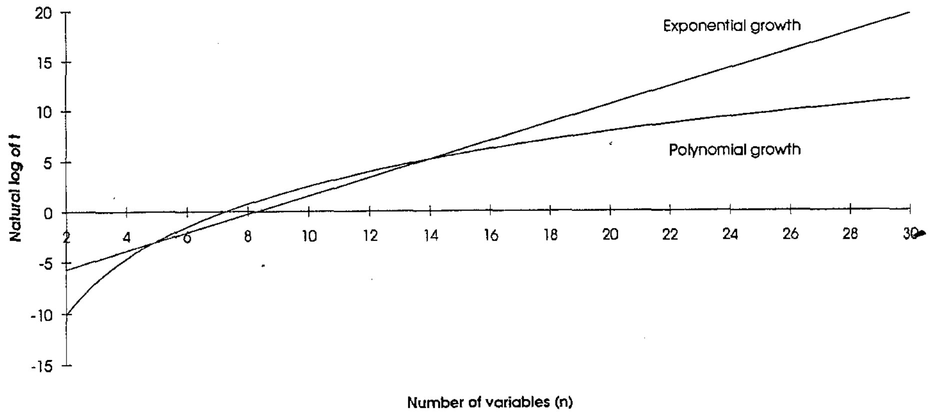

Fig. 5. Comparison of polynomial and exponential models.

appropriate if a nearly linear relationship exists between $\ln t$ and $\ln n$. This relationship is pictured in Figure 3 and does in fact appear linear. Linear regression was used to find constants c_1 and c_2 and produced the equation $t = 1.79 \times 10^{-7} n^{7.826}$. From the figure, it is clear that the correlation between $\ln n$ and $\ln t$ is very high and accounts for most of the variance between data points. In fact, the r^2 statistic for this regression model is 99.42 percent. The other model considered an exponential relationship $t = c_1 e^{c_2 n}$. Taking the natural logarithm of both sides in this case produces the equation $\ln t = \ln c_1 + c_2 n$. Thus, a linear relationship between n and $\ln t$ would suggest that this model is appropriate. Figure 4 shows this relationship, for which a linear fit could also be postulated. Again, linear regression was used to produce the equation $t = 5.341 \times 10^{-4} e^{0.906n}$. Once again, the correlation between $\ln t$ and n is obvious and a very high r^2 value of 98.36 percent is achieved by the regression model.

Of course, both regression models are somewhat limited by the lack of distinct data points. Additional data points (especially for $n > 16$) would be helpful in specifying which model more accurately represents the actual relationship between t and n. As n increases, the two models diverge rapidly. This is pictured graphically in Figure 5, which shows the predicted increase in $\ln t$ as a function of n using both the polynomial and exponential models. As expected, the models yield similar predicted values of $\ln t$ for n between 4 and 16, but for $n > 16$, the exponential model predicts increasingly longer run-times. For example, when $n = 30$, the polynomial model would predict an average CPU time of 64,984 minutes (45 days). The exponential model yields a predicted value of 342,257,019 minutes (651 years). Whichever model is used, it is clear that as n gets above 12, this technique rapidly loses its computational attractiveness.

Finding faster methods to solve, or at least bound, the LPs produced by the algorithm could significantly decrease the run-time. The solutions to the larger problems tend to have an increasing percentage of variables at either their upper or lower bounds. This tendancy serves to keep the number of LPs which must be solved fairly manageable for most problems up to 16 variables. However, nonlinear programming techniques which serve to accelerate convergence in regions which contain good solutions hold promise as a method of further decreasing the number of LPs which must be solved. The global convergence property of the algorithm provides a strong motivation for continuing work to improve the computational behavior of the algorithm for large values of n.

6. Concluding Remarks

In this paper we have developed a branch and bound algorithm for solving GQP and have shown convergence to a global optimum. Because in each node we solve an LP relaxation of the true problem, the algorithm can be regarded as an outer approximation scheme combined with branch and bound. In this respect it is not surprising that we can only show infinite convergence to a global optimum. So a realistic application of the algorithm is to accept an approximate optimal solution, which could be slightly infeasible. An interesting subject, which to our knowledge has not been reported in the literature, is then how to transform an approximate infeasible optimal solution to an approximate feasible optimal solution.

References

1. Al-Khayyal, F. A. (1990), Jointly Constrained Bilinear Programs and Related Problems: An Overview, *Computers and Mathematics with Applications* **19**, 53–62.
2. Al-Khayyal, F. A. (1992), Generalized Bilinear Programming, Part I: Models, Applications and Linear Programming Relaxation, *European Journal of Operational Research* **60**, 306–314.
3. Al-Khayyal, F. A. and Falk, J. E. (1983), Jointly Constrained Biconvex Programming, *Mathematics of Operations Research* **8**, 273–286.
4. Al-Khayyal, F. A. and Larsen, C. (1990), Global Optimization of a Quadratic Function Subject to a Bounded Mixed Integer Constraint Set, *Annals of Operations Research* **25**, 169–180.

5. Al-Khayyal, F. A. and Pardalos, P. M. (1991), Global Optimization Algorithms for VLSI Compaction Problems, *Arabian Journal for Science and Engineering* **16**, 335–339.
6. Baron, D. P. (1972), Quadratic Programming with Quadratic Constraints, *Naval Research Logistics Quarterly* **17**, 253–260.
7. Eben Chaime, M. (1990), The Physical Design of Printed Circuit Boards: A Mathematical Programming Approach, Ph.D. Dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.
8. Erenguc, S. S. and Benson, H. P. (1991), An Algorithm for Indefinite Integer Quadratic Programming, *Computers & Mathematics with Applications* **21**, 99–106.
9. Evans, D. H. (1963), Modular Design – A Special Case in Nonlinear Programming, *Operations Research* **11**, 637–647.
10. Filar, J. A. and Schultz, T. A. (1995), Bilinear Programming and Structured Stochastic Games, *Journal of Optimization Theory and Applications* (forthcoming).
11. Floudas, C. A. and Aggarwal, A. (1990), A Decomposition Strategy for Global Optimum Search in the Pooling Problem, *ORSA Journal on Computing* **2**, 225–235.
12. Floudas, C. A. and Visweswaran, V. (1993), A Primal-Relaxed Dual Global Optimization Approach, *Journal of Optimization Theory and Its Applications* **78**, 187–225.
13. Foulds, L. R., Haughland, D., and Johnston, K. (1992), A Bilinear Approach to the Pooling Problem, *Optimization* **24**, 165–180.
14. Goldberg, J. B. (1984), The Modular Design Problem with Linear Separable Side Constraints: Heuristics and Applications, Ph.D. Dissertation, Industrial and Operations Engineering Department, University of Michigan, Ann Arbor, Michigan.
15. Hansen, P. (1979), Methods of Nonlinear 0-1 Programming, *Annals of Discrete Mathematics* **5**, 53–70.
16. Hansen, P. and Jaumard, B. (1992), Reduction of Indefinite Quadratic Programs to Bilinear Programs, *Journal of Global Optimization* **2**, 41–60.
17. Hao, E. P. (1982), Quadratically Constrained Quadratic Programming: Some Applications and a Method for Solution, *ZOR* **26**, 105–119.
18. Horst, R. (1986), A General Class of Branch and Bound Methods in Global Optimization with Some New Approaches to Concave Minimization, *Journal of Optimization Theory and Applications* **51**, 271–291.
19. Horst, R. and Tuy, H. (1993), *Global Optimization: Deterministic Approaches*, Second Edition, Springer-Verlag, Berlin.
20. Pardalos, P. M. and Rosen, J. B. (1987), *Constrained Global Optimization: Algorithms and Applications*, Lecture Notes in Computer Science 268. Springer-Verlag, Berlin.
21. Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1988), *Numerical Recipes in C*, Cambridge University Press, Cambridge.
22. Rutenberg, D. P. and Shaftel, T. L. (1971), Product Design: Sub-Assemblies for Multiple Markets, *Management Science* **18**, B220–B231.
23. Sherali, H. D. and Tuncbilek, C. H. (1992), A Global Optimization Algorithm for Polynomial Programming Problems Using a Reformulation-Linearization Technique, *Journal of Global Optimization* **2**, 101–112.
24. Shor, N. Z. (1990), Dual Quadratic Estimates in Polynomial and Boolean Programming, *Annals of Operations Research* **25**, 163–168.
25. Van de Panne, C. (1966), Programming with a Quadratic Constraint, *Management Science* **12**, 748–815.
26. Visweswaran, V. and Floudas, C. A. (1993), New Properties and Computational Improvement of the GOP Algorithm for Problems with Quadratic Objective Functions and Constraints, *Journal of Global Optimization* **3**, 439–462.
27. Visweswaran, V. and Floudas, C. A. (1990), A Global Optimization Algorithm (GOP) for Certain Classes of Nonconvex NLP's: II. Applications of Theory and Test Problems, *Computers and Chemical Engineering* **14**, 1417–1434.