Journal of Global Optimization **26:** 387–418, 2003.

Global Optimization of Multiplicative Programs

HONG-SEO RYOO[1] and NIKOLAOS V. SAHINIDIS[2,*]
[1]*University of Illinois at Chicago, Department of Mechanical and Industrial Engineering, 842 West Taylor Street, Chicago, IL 60607, USA;* [2]*University of Illinois at Urbana-Champaign, Department of Chemical and Biomolecular Engineering, 600 South Mathews Avenue, Urbana, IL 61801, USA (*Corresponding author. e-mail: nikos@uiuc.edu)*

Abstract. This paper develops global optimization algorithms for linear multiplicative and generalized linear multiplicative programs based upon the lower bounding procedure of Ryoo and Sahinidis [30] and new greedy branching schemes that are applicable in the context of any rectangular branch-and-bound algorithm. Extensive computational results are presented on a wide range of problems from the literature, including quadratic and bilinear programs, and randomly generated large-scale multiplicative programs. It is shown that our algorithms make possible for the first time the solution of large and complex multiplicative programs to global optimality.

Key words: Multiplicative programs, Branch-and-reduce, Greedy branching

1. Introduction

Multiplicative programs involve products of variables and functions in their objective or constraints. The most-studied multiplicative programs are linear multiplicative programs (LMPs) and generalized linear multiplicative programs ($GLMP$s):

$$(LMP)\left|\begin{array}{ll}\text{globmin} & f(\mathbf{x})=\prod_{j=1}^{p}\left(\mathbf{c}_j^T\mathbf{x}+d_j\right)\\ \text{s. t.} & \mathbf{Ax}\le\mathbf{b}\\ & \mathbf{x}\in\mathbf{X}:=\{\mathbf{x}\in\mathbb{R}^n:-\infty<\mathbf{l}\le\mathbf{x}\le\mathbf{u}<+\infty\}\end{array}\right.$$

where $\mathbf{c}_j\in\mathbb{R}^n$, $d_j\in\mathbb{R}$ for $j=1,\ldots,p$, $\mathbf{A}\in\mathbb{R}^{m\times n}$, $\mathbf{b}\in\mathbb{R}^m$, and $\mathbf{c}_j^T\mathbf{x}+d_j>0$ for $j=1,\ldots,p$ for all $\mathbf{x}\in\mathbf{X}$ for which $\mathbf{Ax}\le\mathbf{b}$; and

$$(GLMP)\left|\begin{array}{ll}\text{globmin} & f(\mathbf{x})=\sum_{i=1}^{t}\prod_{j=1}^{p_i}\left(\mathbf{c}_{ij}^T\mathbf{x}+d_{ij}\right)\\ \text{s. t.} & \mathbf{x}\in\mathbf{M}:=\{\mathbf{x}\in\mathbf{X}:\mathbf{Ax}\le\mathbf{b}\}\end{array}\right.$$

where $\mathbf{c}_{ij}\in\mathbb{R}^n$, and $d_{ij}\in\mathbb{R}$, $i=1,\ldots,t$, $j=1,\ldots,p_i$ and $\mathbf{A}\in\mathbb{R}^{m\times n}$, $\mathbf{b}\in\mathbb{R}^m$.

Above, we defined each term in the objective function of $GLMP$ to be the product of arbitrarily many, unconstrained in sign linear functions as opposed to

the conventional definition of $GLMP$ which involves the sum of the product of only two positive linear functions (e.g., [13, 31]). With our broader definition, one can see quadratic programming (QP), bilinear programming (BLP), as well as LMP fall into the category of $GLMP$.

LMP and $GLMP$ have attracted considerable attention in the literature because of their large number of practical applications in various fields of study, including microeconomics [8], financial optimization [16, 22], VLSI chip design [6, 21], decision tree optimization [1], portfolio optimization [16, 23], plant layout design [28], multicriteria optimization problems [9], robust optimization [26], and data mining/pattern recognition [2].

LMP is a quasiconcave program [7, 34], and may possess several local minima [12]. $GLMP$ is obviously multiextremal, for its special cases such as LMP, BLP, and QP are multiextremal. Both LMP and $GLMP$ are known to be $\mathcal{NP}$-hard problems [14, 24].

In the last decade, many solution algorithms have been proposed for globally solving LMP and $GLMP$, most notably by Konno and co-workers [11, 12, 14, 17, 15, 18]. Solution algorithms for multiplicative methods can be classified as parameterization-based methods [11, 12, 14, 17], outer-approximation methods [15, 18, 35], a vertex enumeration method [27], a method based on image space analysis [7], a primal and dual simplex method [31], and a cutting plane method in outcome space [4]. Heuristic methods have also been developed [3, 20].

On the computational front, problems reported solved thus far in the literature have been 'small' in size and 'simple' in terms of the objective function structure. To wit, for LMP with the objective function a product of two functions ($p = 2$), random instances of $(m, n) = (220, 200)$ are the largest size problems solved in the literature [12]. For $p = 5$, however, the size of largest problems solved drastically reduces to only $(m, n) = (20, 30)$ [18]. A similar situation holds for $GLMP$. For $GLMP$ with $t = 2$ and $p_1 = p_2 = 2$, the size of the largest problems solved in the literature is $(m, n) = (70, 50)$, while for $GLMP$ with $t = 4$ and $p_i = 2$, $(i = 1, \ldots, 4)$, the largest problems reported have a constraint matrix of size $(m, n) = (30, 50)$ [18]. Furthermore, no computational results for $GLMPs$ with $p_i > 2$ have been reported in the literature so far.

The primary contributions of this paper are two. First, we develop global optimization algorithms that make possible for the first time the solution of large-scale LMPs and $GLMP$s. Second, we propose a new branching scheme that can be used in the context of any rectangular branch-and-bound algorithm.

The paper is organized as follows. Section 2 presents the skeleton of the proposed algorithms. They are based on our earlier branch-and-reduce approach [29]. Section 3 presents the two bounding schemes that we propose for LMP and $GLMP$; these rely on our recent work on bounding monomials [30]. In Section 4, we develop greedy branching rules for rectangular branch-and-bound algorithms. The idea behind greedy branching is to select the branching variable and position such that the largest possible lower bound improvement is made in the immediate

descendants of the current node. A suitable measure of lower bound improvement is introduced for this purpose and used in the development of greedy branching rules for various classes of problems including multiplicative programs. Under this framework, it is shown that the classical bisection branching rule is greedy when applied to separable concave minimization problems. Section 5 briefly reviews a set of range contraction mechanisms we incorporate in our algorithms. The proposed algorithms are next illustrated on two examples from the literature in Section 6. Extensive computational experiments are reported in Section 7. Here, we solve a wide variety of test problems from the literature as well as randomly generated problems and present computational comparisons with earlier approaches. In summary, our algorithms can solve $LMPs$ with $p = 5$ and $(m, n) = (200, 200)$ in a matter of minutes on a computer equipped with a slow 133 MHz processor and only 64 MB of memory. We also report for the first time the solution of $GLMP$s with more than two products ($p_i > 2$) in every objective function term. Finally, concluding remarks are presented in Section 8.

2. The Branch-and-Reduce Algorithm

The branch-and-reduce global optimization algorithm of [29] constitutes the backbone of algorithms for LMP and $GLMP$ that are developed in this paper. We briefly summarize the steps of this algorithm.

Let P be the nonconvex problem to be solved and $\mathbf{X}$ be the simple bounds constraints as defined in LMP above. Also, let P_i be problem P defined over $\mathbf{X}_i \subseteq \mathbf{X}$.

ALGORITHM *branch-and-reduce;*

Initialization Step.

Set the lower and upper bounds of the search tree: $U \leftarrow +\infty$ and $L \leftarrow -\infty$. Put problem P in $\mathcal{A}$, the list of 'active' problems to be solved. Set $k = 1$ and go to Step k.1.

Iteration k:

Step k.1. If $\mathcal{A}$ is empty, stop. Otherwise, select and remove P_k from $\mathcal{A}$. Go to Step k.2.

Step k.2. Preprocess P_k using *feasibility-based range reduction techniques*. Lower and upper bound P_k. Update the lower and/or upper bounds of the search tree and delete all inferior nodes. If the upper and lower bounds are equal (or within a prespecified level), stop. Otherwise, postprocess P_k using both feasibility-based and *optimality-based range reduction techniques.* If the bounds on variables are substantially improved, repeat Step k.2. Otherwise, go to Step k.3.

Step k.3. Partition $\mathbf{X}_k$ into $\mathbf{X}_{k_1}$ and $\mathbf{X}_{k_2}$ and add two subproblems P_{k_1} and P_{k_2} to $\mathcal{A}$. Set $k \leftarrow k + 1$ and go to Step k.1.

Compared to classical branch and bound algorithms, branch and reduce places a considerable amount of effort in preprocessing and postprocessing steps aimed at reducing ranges of variables and tightening relaxations at every node of the search tree.

3. Lower Bounding

By the use of an additional variable for each linear function, LMP and $GLMP$ transform into equivalent programs whose objective function is, respectively, a monomial function of the type:

$$\prod_{j=1}^{p} y_j \tag{1}$$

and the sum of t monomial functions. Of course, the relationships among introduced variables and the linear functions in the original objective must be added to the constraint sets:

$$(LMP) \left| \begin{array}{ll} \min & f(\mathbf{y}) = \displaystyle\prod_{j=1}^{p} y_j \\ \text{s. t.} & \mathbf{x} \in \mathbf{M} \\ & y_j = \mathbf{c}_j^T \mathbf{x} + d_j, \quad j = 1, \ldots, p \end{array} \right.$$

and

$$(GLMP) \left| \begin{array}{ll} \min & f(\mathbf{y}) = \displaystyle\sum_{i=1}^{t} \prod_{j=1}^{p_i} y_{ij} \\ \text{s. t.} & \mathbf{x} \in \mathbf{M} \\ & y_{ij} = \mathbf{c}_{ij}^T \mathbf{x} + d_{ij}, \quad i = 1, \ldots, t;\ j = 1, \ldots, p_i \end{array} \right.$$

For a function f and optimization program P, in the sequel we use:

$\underline{f}$: a convex lower bounding function of f

$\underline{P}$: a convex lower bounding program of P

3.1. LOWER BOUNDING FOR *LMP*

For $j = 1, \ldots, p$, let $y_j^L = \min_{\mathbf{x} \in \mathbf{M}} \mathbf{c}_j^T \mathbf{x} + d_j$ and $y_j^U = \max_{\mathbf{x} \in \mathbf{M}} \mathbf{c}_j^T \mathbf{x} + d_j$. As $\mathbf{c}_j^T \mathbf{x} + d_j > 0$ for all $\mathbf{x} \in \mathbf{M}$, it follows that $y_j^L > 0$ for all j.

We first convert LMP into an equivalent separable program:

$$(LMP') \left| \begin{array}{ll} \text{globmin} & f(\mathbf{y}) = \sum_{j=1}^{p} \log(y_j) \\ \text{s. t.} & \mathbf{x} \in \mathbf{M} \\ & y_j = \mathbf{c}_j^T \mathbf{x} + d_j, \quad j = 1, \ldots, p \end{array} \right.$$

The monotonic increasing property of the logarithmic function directly yields the following:

PROPOSITION 1 *LMP and LMP' are equivalent programs.*

$\underline{LMP'}$ of separable LMP' is easy to construct:

$$(\underline{LMP'}) : \left| \begin{array}{ll} \min & \underline{f}(\mathbf{y}) = \displaystyle\sum_{j=1}^{p} \left(\alpha_j y_j + \beta_j\right) \\ \text{s. t.} & \mathbf{x} \in \mathbf{M} \\ & y_j = \mathbf{c}_j^T \mathbf{x} + d_j, \quad j = 1, \ldots, p \end{array} \right.$$

where $\alpha_j = (\log y_j^U - \log y_j^L)/(y_j^U - y_j^L)$, $\beta_j = \log y_j^L - \alpha_j y_j^L$ for $j = 1, \ldots, p$.

This lower bounding scheme is called `Loga` (the Logarithmic transformation method) and is primarily used for its simplicity. In addition, as noted in [30], `Loga` constructs a lower bounding function of (1) that is tighter than competing lower bounding functions in the interior of the box of variable bounds.

3.2. LOWER BOUNDING FOR *GLMP*

We propose the `rAI` (recursive Arithmetic Intervals) method for lower bounding of $GLMP$. `rAI` is a two-step procedure:

Step 1. Recursively replace each bilinear term in (1) with a new variable until (1) is replaced by a single variable. An example is:

$$\underbrace{\underbrace{\underbrace{\underbrace{y_1 y_2}_{=:y_{p+1}}\, y_3 \cdots y_{p-1}}_{=:y_{p+2}}\, y_p}_{\vdots}}_{=:y_{2p-2}}$$
$$=:y_{2p-1}$$

Step 2. Linearly lower bound each of the $p-1$ introduced variables, *i.e.*, the bilinear terms, with the maximum of two linear functions:

$$y_j = y_{j_1} y_{j_2} \geq \max \left\{ \begin{array}{l} y_{j_1} y_{j_2}^U + y_{j_1}^U y_{j_2} - y_{j_1}^U y_{j_2}^U \\ y_{j_1} y_{j_2}^L + y_{j_1}^L y_{j_2} - y_{j_1}^L y_{j_2}^L \end{array} \right\}$$

for all $j = p+1, \ldots, 2p-1$, where j_1 and j_2 are the indices of the problem variables whose product is identified with variable j. By interval arithmetic arguments, the bounds on the introduced variables are given by

$$y_j^L := \min\left\{y_{j_1}^L y_{j_2}^L,\ y_{j_1}^L y_{j_2}^U,\ y_{j_1}^U y_{j_2}^L,\ y_{j_1}^U y_{j_2}^U\right\}$$

and

$$y_j^U := \max\left\{y_{j_1}^L y_{j_2}^L,\ y_{j_1}^L y_{j_2}^U,\ y_{j_1}^U y_{j_2}^L,\ y_{j_1}^U y_{j_2}^U\right\}$$

for $j = p+1, \ldots, 2p-1$.

This scheme is also easy to implement and provides tight bounds: we show in [30] that `rAI` yields the convex (and also the concave) envelope of (1) over the unit hypercube. The so constructed $\underline{GLMP}$ is an $m' \times n'$ LP:

$$(\underline{GLMP}) \left| \begin{array}{ll} \min & \underline{f}(\mathbf{y}) = \displaystyle\sum_{i=1}^{t} y_{i,2p_i-1} \\ \text{s. t.} & \mathbf{x} \in \mathbf{M} \\ & y_{ij} = \mathbf{c}_{ij}^T \mathbf{x} + d_{ij}, \\ & \qquad i = 1, \ldots, t;\ j = 1, \ldots, p_i \\ & y_{ij} \geq y_{ij_1} y_{ij_2}^U + y_{ij_1}^U y_{ij_2} - y_{ij_1}^U y_{ij_2}^U, \\ & \qquad i = 1, \ldots, t;\ j = p_i+1, \ldots, 2p_i-1 \\ & y_{ij} \geq y_{ij_1} y_{ij_2}^L + y_{ij_1}^L y_{ij_2} - y_{ij_1}^L y_{ij_2}^L, \\ & \qquad i = 1, \ldots, t;\ j = p_i+1, \ldots, 2p_i-1 \end{array} \right.$$

where m' does not include the bounds on the x variables:

$$m' := m + 3\sum_{i=1}^{t} p_i - 2t$$

$$n' := n + 2\sum_{i=1}^{t} p_i - t.$$

4. Branching

4.1. GREEDY BRANCHING

A branching operation in rectangular branch-and-bound is concerned with two decisions: the selection of the branching variable and the determination of its branching position. In general, these two decisions are made based upon some strategy that aims to minimize the search tree size, thus expediting termination of the search process.

This section develops the 'greedy branching' scheme for a general rectangular branch-and-bound algorithm that selects the branching variable and the branching point in order to achieve the largest reduction in the relaxation gap — the difference between upper and lower bounds — in the immediate descendants of the current problem after branching.

Consider a separable nonconvex problem P defined over the set of variable bounds $[\mathbf{y}^L, \mathbf{y}^U]$ at the current node of a branch and bound algorithm. Suppose that y_k appears in nonconvex functions $f_{k_i}(y_k), i = 1, \ldots, n_k$, in the objective function and/or constraints of P.

4.1.1. *Greedy Branching Variable Selection*

For illustration, consider f_{k_1}. We define a measure of the nonconvexity that variable y_k contributes to P via function f_{k_1} as

$$\int_{y_k^L}^{y_k^U} \left[f_{k_1}(y_k) - \underline{f}_{k_1}(y_k) \right] \mathrm{d}y_k, \tag{2}$$

which is the area between $f_{k_1}(y_k)$ and $\underline{f}_{k_1}(y_k)$. According to this criterion, it follows that the variable contributing the maximum amount to the above measure of nonconvexities of the problem is given by:

$$j \in \underset{k}{\operatorname{argmax}} \left\{ \sum_{i=1}^{n_k} \int_{y_k^L}^{y_k^U} \left[f_{k_i}(y_k) - \underline{f}_{k_i}(y_k) \right] \mathrm{d}y_k \right\}.$$

The above definite integrals admit closed form solutions when f_{k_i}s have antiderivatives.

Consider LMP'. Recall that, for LMP' and $\underline{LMP'}$, $f_j(y_j) = \log y_j$ and $\underline{f}_j(y_j) = \alpha_j y_j + \beta_j$ where $\alpha_j = (\log y_j^U - \log y_j^L)/(y_j^U - y_j^L)$, $\beta_j = \log y_j^L - \alpha_j y_j^L$ for $j = 1, \ldots, p$.

PROPOSITION 2 *Consider LMP' and $\underline{LMP'}$. Variable y_j with*

$$j \in \underset{1 \le k \le p}{\operatorname{argmax}} \left\{ \log \left(\frac{(y_k^U)^{y_k^U}}{(y_k^L)^{y_k^L}} \right) - \frac{\alpha_k \left((y_k^U)^2 - (y_k^L)^2 \right)}{2} - (\beta_k + 1)(y_k^U - y_k^L) \right\}$$

is one that contributes most to the nonconvexity of LMP'.

Proof. As LMP' is separable, a greedy branching variable is one with the largest

$$\int_{y_k^L}^{y_k^U} \left[\log(y_k) - \alpha_k y_k - \beta_k \right] \mathrm{d}y_k.$$

Carrying out this definite integral yields the result. □

4.1.2. *Greedy Branching Point Selection*

For variable, say y_j, selected for branching, let:

$$f(y_j) := \sum_{i=1}^{n_j} f_{k_i}(y_j)$$

Its underestimator is:

$$\underline{f}(y_j) := \sum_{i=1}^{n_j} \underline{f}_{k_i}(y_j).$$

Selecting the greedy branching point for y_j so as to maximize the reduction in the underestimation gap by the maximum amount after the branching operation is equivalent to locating point $c_j \in [y_j^L, y_j^U]$ such that the quantity

$$\int_{y_j^L}^{c_j} \left\{ f(y_j) - \underline{f}^{\text{left}}(y_j) \right\} \mathrm{d}y_j + \int_{c_j}^{y_j^U} \left\{ f(y_j) - \underline{f}^{\text{right}}(y_j) \right\} \mathrm{d}y_j$$

is minimized, where $\underline{f}^{\text{left}}$ and $\underline{f}^{\text{right}}$ are appropriately derived over their respective bounds $[y_j^L, c_j]$ and $[c_j, y_j^U]$ using standard relaxation techniques [25]. The geometric interpretation of this greedy branching point selection rule is provided in Figure 1. Here, the branching point c_j is chosen such that $(c_j, f_j(c_j))$ and the line segment joining $(y_j^L, f_j(y_j^L))$ and $(y_j^U, f_j(y_j^U))$ as the base forms a triangle with the largest area that fits between f_j and $\underline{f}_j$. This area represents the amount of reduction in the relaxation gap after the interval $[y_j^L, y_j^U]$ is partitioned at c_j and explains why c_j brings about the largest reduction in the relaxation gap.

The greedy branching position selection rule is formally stated below:

THEOREM 1 *Let f be a univariate concave function of $y \in [y^L, y^U]$. The following hold:*
(i) If $f(y)$ is continuous on $[y^L, y^U]$ and differentiable on (y^L, y^U), branching at $c \in (y^L, y^U)$ satisfying

$$f'(c) = \frac{f(y^U) - f(y^L)}{y^U - y^L} \tag{3}$$

yields the maximum reduction in the relaxation gap in the subtree generated immediately after branching.
(ii) Suppose $f(y)$ is continuous and differentiable on (y^L, y^U). If there exists $c \in (y^L, y^U)$ satisfying (3) in (i), branching at c yields the maximum reduction in the relaxation gap in the subtree generated after branching. Otherwise, branching at

$$c \in \operatorname{argmax} \left\{ \lim_{y \downarrow y^L} f(y) - \underline{f}(y^L), \lim_{y \uparrow y^U} f(y) - \underline{f}(y^U) \right\}$$

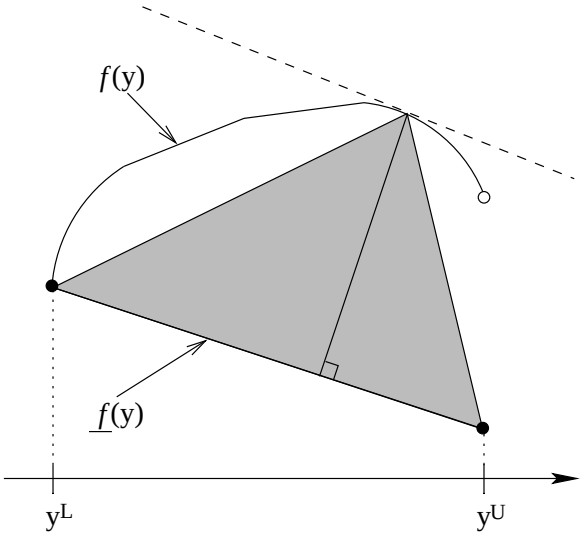

Figure 1. Geometric interpretation of greedy branching point selection rule.

yields the maximum reduction in the relaxation gap in the subtree generated after branching.

(iii) Suppose $f(y)$ *is discontinuous on* $[y^L, y^U]$ *and nondifferentiable on* (y^L, y^U). *Let* $y^L = y_1, \ldots, y_l = y^U$ *denote the points where* f *is nondifferentiable. Let*

$$c_k \in \operatorname{argmax}\left\{ f(c_i) - \underline{f}(c_i), i = 3, \ldots, l-1 \right\}$$

where $c_i \in (y_{i-1}, y_i)$ *satisfies (3) in (i). If there exists* $c_1 \in (y_1, y_2)$ *and* $c_l \in (y_{l-1}, y_l)$ *satisfying (3), then*

$$c \in \operatorname{argmax}\left\{ f(c_i) - \underline{f}(c_i), i = 1, k, l \right\};$$

if c_1 *exists but not* c_l, *then*

$$c \in \operatorname{argmax}\left\{ f(c_i) - \underline{f}(c_i), i = 1, k;\ \lim_{y \uparrow y_l} f(y) - \underline{f}(y_l) \right\};$$

if c_l *exists but not* c_1, *then*

$$c \in \operatorname{argmax}\left\{ \lim_{y \downarrow y_1} f(y) - \underline{f}(y_1);\ f(c_i) - \underline{f}(c_i), i = k, l \right\};$$

otherwise,

$$c \in \operatorname{argmax}\left\{ \lim_{y \downarrow y_1} f(y) - \underline{f}(y_1),\ f(c_k) - \underline{f}(c_k),\ \lim_{y \uparrow y_l} f(y) - \underline{f}(y_l) \right\}$$

is the greedy branching point that yields the maximum reduction in the relaxation gap in the subtree generated after branching.

If f is nonlinear on (y^L, y^U), c in (i) and (ii) are uniquely determined.

Proof. (i) Referring to Figure 1, we note that the base of the triangle whose area we want to maximize is fixed while the height of the triangle changes as a function of $y \in [y^L, y^U]$. The height of the triangle is defined as

$$h = \cos(\theta)(f(y) - \underline{f}(y)),$$

where

$$\theta = \arctan\left(\frac{f(y^U) - f(y^L)}{y^U - y^L}\right).$$

Therefore, finding a greedy branching point reduces to solving the following univariate concave maximization problem:

$$(MAXDEV) \left| \begin{array}{ll} \max & f(y) - \underline{f}(y) \\ \text{s. t.} & y^L \le y \le y^U \end{array} \right.$$

A global maximum of *MAXDEV* is attained at a point satisfying the first-order necessary condition for f or at an endpoint.

(ii) If there exists $c \in (y^L, y^U)$ satisfying (3), branching at c obviously reduces the relaxation gap by the largest amount.

For the other case, without loss of generality, let us re-define f as:

$$f(y) := \min\{f_m(y), f_L(y), f_U(y)\}$$

where

$$f_m(y) := \begin{cases} \lim_{y \downarrow y^L} f(y), & \text{if } y = y^L \\ \lim_{y \uparrow y^U} f(y), & \text{if } y = y^U \\ f(y), & \text{for } y \in (y^L, y^U) \end{cases}$$

$$f_L(y) := \begin{cases} f(y^L), & \text{if } y = y^L \\ +\infty, & \text{for } (y^L, y^U] \end{cases}$$

$$f_U(y) := \begin{cases} f(y^U), & \text{if } y = y^U \\ +\infty, & \text{for } [y^L, y^U) \end{cases}$$

This re-definition of f brings about the attainment of the supremum of *MAXDEV* at an endpoint of $[y^L, y^U]$.

(iii) As f is concave, f is continuous on $[y_{i-1}, y_i]$, differentiable on (y_{i-1}, y_i) for $i = 3, \ldots, l-1$, and continuous and differentiable on (y_{i-1}, y_i) for $i = 2, l$. Applying the proof of (i) above for $i = 2, l$ and the proof of (ii) for $i = 3, \ldots, l-1$, we prove the assertions.

The uniqueness result for the nonlinear case for (i) and (ii) follows from monotonicity of a nonlinear univariate concave function. □

COROLLARY 1 *Let y_k be the variable selected for branching in LMP'. Then, branching at*

$$c_k = \frac{y_k^U - y_k^L}{\log y_k^U - \log y_k^L}$$

yields the maximum reduction in the relaxation gap in the subtree generated after branching.

Proof. We have $f(y_k) = \log y_k$ with $f'(c_k) = c_k^{-1}$. As $\log y_k$ is continuous on $[y_k^L, y_k^U]$ and differentiable on (y_k^L, y_k^U), we apply Theorem 1.(i) and solve for c_k in $f'(c_k) = \alpha_k$, where

$$\alpha_k = \frac{\log y_k^U - \log y_k^L}{y_k^U - y_k^L}. \qquad \square$$

Next, we consider a separable concave quadratic program.

COROLLARY 2 *Let y_k be the variable selected for branching with $f(y_k) = -y_k^2$. Then, the bisection, i.e., branching at*

$$c_k = \frac{y_k^L + y_k^U}{2}$$

yields the maximum reduction in the relaxation gap in the subtree generated after branching.

Proof. Apply Theorem 1.(i) with $f(y_k) = -y_k^2$: set $f'(c_k) = -2c_k = \alpha_k = -(y_k^L + y_k^U)$ and solve for c_k. $\square$

REMARK 1 (i) Theorem 1 can alternatively be stated as follows:

Let f be a univariate concave function of $y \in [y^L, y^U]$. Then, branching at $c \in [y^L, y^U]$ satisfying

$$\frac{f(y^U) - f(y^L)}{y^U - y^L} \in \partial f(c)$$

yields the maximum reduction in the relaxation gap in the subtree generated after branching, where $\partial f(y^0)$ of f at y^0 is the set of $s \in \mathbb{R}$ satisfying

$$f(y) \leq f(y^0) + \langle s, y - y^0 \rangle$$

for all $y \in [y^L, y^U]$ (*i.e.*, the *superdifferential* of f.)

(ii) For concave programs, [19] proposes a branching rule to minimize the collective area of the underestimating gap of the child subproblems. Their rule branches at the same point as the greedy rule of Corollary 2.

REMARK 2 While the greedy branching concept above was presented in the context of separable formulations, it can be easily applied to nonseparable formulations as well. One possibility is convert nonseparable formulations to separable ones using standard transformations [25]. Note that this transformation does not need to be applied for the purpose of constructing a relaxation; it suffices to use it to calculate violations that are then transferred back to original problem variables. Yet another possibility is to work directly with the nonseparable formulations and define the nonconvexity measures in (2) for each variable after fixing all other variables to the solution of the relaxation or some other judiciously chosen point.

4.2. BRANCHING SCHEMES FOR MULTIPLICATIVE PROGRAMS

The greedy branching is proposed for LMP:

OPERATION *greedy branching — LMP*;
begin
select variable y_j for branching via Proposition 2;
compute c_j via Corollary 1;
branch at c_j;
end;

It is well-known that the gap between a bilinear function

$$f(\mathbf{y}) = y_i y_j$$

and its underestimator

$$\underline{f}(\mathbf{y}) = \max \left\{ \begin{array}{c} y_i y_j^U + y_i^U y_j - y_i^U y_j^U \\ y_i y_j^L + y_i^L y_j - y_i^L y_j^L \end{array} \right\}$$

is maximum at the midpoint of the box $[y_i^L, y_i^U] \times [y_j^L, y_j^U]$. This is exploited in our greedy-like branching scheme for $GLMP$. Let

$$mpv := y_i^{mp} y_j^{mp} - \max \left\{ \begin{array}{c} y_i^{mp} y_j^U + y_i^U y_j^{mp} - y_i^U y_j^U \\ y_i^{mp} y_j^L + y_i^L y_j^{mp} - y_i^L y_j^L \end{array} \right\}$$

where $y_i^{mp} := \frac{y_i^L + y_i^U}{2}$ and $y_j^{mp} := \frac{y_j^L + y_j^U}{2}$. We use mpv to compute the compounded violation for each introduced variable. This compounded violation for an introduced variable reflects the amount of nonconvexities in the current problem formulation due to the variable by taking into account the number of variables that appear in the monomial term containing this variable via introduced bilinear relationships.

OPERATION *branching — GLMP*
begin
set $violation(ij) \leftarrow 0$ for all $i = 1, \ldots, t$ and $j = 1, \ldots, 2p_i - 1$;
for $i = 1, \ldots, t$ and $p_i \geq 2$ **do**
for $j = p_i + 1, \ldots, 2p_i - 1$ **do**
compute mpv for $y_{ij} = y_{ij_1} y_{ij_2}$;
set $violation(ij) \leftarrow violation(ij) + mpv$;
set $violation(ij_1) \leftarrow violation(ij_1) + mpv$;
set $violation(ij_2) \leftarrow violation(ij_2) + mpv$;
enddo
enddo
select $k \in \text{argmax}\{violation(ij) : i = 1, \ldots, t;\ j = 1, \ldots, p_i\}$;
bisect $[y_k^L, y_k^U]$;
end;

5. Range Reduction

Two types of range reduction mechanisms — feasibility-based and optimality-based — are proposed in [29] to accelerate the convergence of global optimization algorithms. In general, the former are applied in the preprocessing of a branch and bound node whereas both types are applied during the postprocessing phase of the node.

As for the feasibility-based range reduction, as LMP and $GLMP$ are linearly constrained, we improve bounds on variables via applying *linearity-based range reduction* to the set of all linear constraints $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ $(i = 1, \ldots, m)$ of the problem:

TECHNIQUE *linearity-based range reduction*;
for $i = 1, \ldots, m$ **do**
begin
compute $rU_i := \sum_{j=1}^{n} \max\{a_{ij} x_j^L, a_{ij} x_j^U\}$;
compute $rL_i := \sum_{j=1}^{n} \min\{a_{ij} x_j^L, a_{ij} x_j^U\}$;
if $rL_i > b_i$ **then**
stop. problem is infeasible;
elseif $rU_i < b_i$ **then**
constraint is redundant. delete it from problem formulation;
else
for $j = 1, \ldots, n$ **do**
if $a_{ij} > 0$ **then**

$$x_j^U \leftarrow \min\left\{x_j^U, \frac{b_i - rL_i + \min\{a_{ij}x_j^L, a_{ij}x_j^U\}}{a_{ij}}\right\};$$

elseif $a_{ij} < 0$ **then**

$$x_j^L \leftarrow \max\left\{x_j^L, \frac{b_i - rU_i + \max\{a_{ij}x_j^L, a_{ij}x_j^U\}}{a_{ij}}\right\};$$

endif;
enddo
endif;
end;
enddo

For $GLMP$, the bilinear relationships among introduced variables $y_k = y_i y_j$ can also be exploited to further improve the bounds on variables:

TECHNIQUE *bilinearity-based range reduction*;
begin

$$y_k^L \leftarrow \max\left\{y_k^L, \min\left(y_i^L y_j^L, y_i^L y_j^U, y_i^U y_j^L, y_i^U y_j^U\right)\right\};$$

$$y_k^U \leftarrow \min\left\{y_k^U, \max\left(y_i^L y_j^L, y_i^L y_j^U, y_i^U y_j^L, y_i^U y_j^U\right)\right\};$$

if $0 \notin [y_j^L, y_j^U]$ **then**

$$y_i^L \leftarrow \max\left\{y_i^L, \min\left\{\frac{y_k^L}{y_j^L}, \frac{y_k^L}{y_j^U}, \frac{y_k^U}{y_j^L}, \frac{y_k^U}{y_j^U}\right\}\right\};$$

$$y_i^U \leftarrow \min\left\{y_i^U, \max\left\{\frac{y_k^L}{y_j^L}, \frac{y_k^L}{y_j^U}, \frac{y_k^U}{y_j^L}, \frac{y_k^U}{y_j^U}\right\}\right\};$$

endif;
if $0 \notin [y_i^L, y_i^U]$ **then**

$$y_j^L \leftarrow \max\left\{y_j^L, \min\left\{\frac{y_k^L}{y_i^L}, \frac{y_k^L}{y_i^U}, \frac{y_k^U}{y_i^L}, \frac{y_k^U}{y_i^U}\right\}\right\};$$

$$y_j^U \leftarrow \min\left\{y_j^U, \max\left\{\frac{y_k^L}{y_i^L}, \frac{y_k^L}{y_i^U}, \frac{y_k^U}{y_i^L}, \frac{y_k^U}{y_i^U}\right\}\right\};$$

endif;
end;

As for the optimality-based range reduction, we utilize the range contraction tools for simple bounds constraints $\mathbf{x}^L \leq \mathbf{x} \leq \mathbf{x}^U$ that we developed in [29] (Corollaries 2 and 3). Let L be the objective value of the current node subproblem, $\mathbf{x}^R$ its primal solution, λ_j (j=1, ... , n) its dual prices, and U the known upper bound of the problem to be solved. Then:

TECHNIQUE *optimality-based range reduction*;
for $j = 1, \ldots, n$ **do**
  if $x_j^R = x_j^L$ and its dual price $\lambda_j > 0$, **then**
    $x_j^U \leftarrow \min\left\{x_j^U, x_j^L + \dfrac{U - L}{\lambda_j}\right\}$
  endif;
  if $x_j^R = x_j^U$ and its dual price $\lambda_j > 0$, **then**
    $x_j^L \leftarrow \max\left\{x_j^L, x_j^U - \dfrac{U - L}{\lambda_j}\right\}$
  endif;
enddo

Furthermore, we utilize the objective function cut

$$\text{objective function value} \leq U$$

for further effective range reduction. For example, we require for LMP that

$$\prod_{j=1}^{p} y_j \leq U$$

and improve the upper bounds on variables y_j $(j = 1, \ldots, p)$ via

$$y_j^U \longleftarrow \min\left\{y_j^U, \frac{U}{\prod_{i \neq j} y_i^L}\right\}.$$

6. Illustrative Examples

This section demonstrates the steps of the proposed algorithms on two problems from the literature.

6.1. *LMP* ILLUSTRATIVE EXAMPLE

The following example is taken from [7] and is a modified version of a problem that originally appeared in [10]:

$$(Ex-1)\left|\begin{array}{ll} \text{globmin} & (x_1+x_2)(x_1-x_2+7) \\ \text{s. t.} & FS1 := \left\{\mathbf{x} \in \mathbb{R}^2 \;\middle|\; \begin{array}{c} 2x_1 + x_2 \leq 14 \\ x_1 + x_2 \leq 10 \\ -4x_1 + x_2 \leq 0 \\ 2x_1 + x_2 \geq 6 \\ x_1 + 2x_2 \geq 6 \\ x_1 - x_2 \leq 3 \\ x_1 \leq 5 \end{array}\right\} \end{array}\right.$$

$Ex-1$ is first converted into an equivalent problem by introducing two additional variables for the linear functions in the objective and then taking the logarithmic transformation on the objective function:

$$(Ex-1') \left| \begin{array}{ll} \text{globmin} & \log y_1 + \log y_2 \\ \text{s. t.} & FS1' := \left\{ (\mathbf{x}, \mathbf{y}) \in \mathbb{R}^4 \left| \begin{array}{c} \mathbf{x} \in FS1 \\ y_1 - x_1 - x_2 = 0 \\ y_2 - x_1 + x_2 - 7 = 0 \end{array} \right. \right\} \end{array} \right.$$

The initialization step sets $U \leftarrow +\infty$ and $L \leftarrow -\infty$. After y_1 and y_2 are minimized and maximized over $FS1$, and *linearity-based range reduction* is applied to the last two constraints of $FS1'$, the algorithm improves the bounds on variables to:

$$\mathbf{X} \leftarrow \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^4 : (1.99, 7.99, 9.97, 1) \leq (\mathbf{x}, \mathbf{y}) \leq (2.01, 8.01, 10, 1.01)\}$$

During the process of minimization and maximization of the introduced variables, feasible solutions to the problem are obtained. In this example, we find that the best solution amongst all these solutions is $\mathbf{x} = (2, 8)$ with $f(\mathbf{x}) = 10$. Hence, $U \leftarrow 10$. Next, $\mathbf{X}_1 \leftarrow \mathbf{X}$, $P_1 \leftarrow Ex-1'$, P_1 is put in $\mathcal{A}$, and P_1 is selected for solution.

Before lower bounding, the algorithm checks if the current formulation of P_1 can be further improved by using *linearity-based range reduction* in conjunction with the objective function cut. Through this, the bounds on variables improve to:

$$\mathbf{X}_1 \leftarrow \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^4 : (2, 8, 10, 1) \leq (\mathbf{x}, \mathbf{y}) \leq (2, 8, 10, 1)\}$$

When $\underline{P_1}$ constructed on $\mathbf{X}_1$ is solved, $L \leftarrow 10$ is obtained at solution $\mathbf{x} = (2, 8)$. Hence, the current best solution is updated to $\mathbf{x}^* \leftarrow (2, 8)$. Finally, as $U = L$, the algorithm terminates. The algorithm proves global optimality of $\mathbf{x}^* = (2, 8)$ at the root node of the branch-and-reduce search tree.

6.2. *GLMP* ILLUSTRATIVE EXAMPLE

The following example is taken from [11]:

$$(Ex-2) \left| \begin{array}{ll} \text{globmin} & x_1 + (x_1 - x_2 + 5)(x_1 + x_2 - 1) \\ \text{s. t.} & FS2 := \left\{ \mathbf{x} \in \mathbb{R}^2 \left| \begin{array}{c} 2x_1 + 3x_2 \geq 9 \\ 3x_1 - x_2 \leq 8 \\ -x_1 + 2x_2 \leq 8 \\ x_1 + 2x_2 \leq 12 \\ x_1 \geq 0 \end{array} \right. \right\} \end{array} \right.$$

Again, the problem is first converted into an equivalent problem:

$$(Ex-2)\left|\begin{array}{ll}\text{globmin} & y_{1,1}+y_{2,3}\\ \text{s. t.} & FS2' := \left\{(\mathbf{x},\mathbf{y})\in\mathbb{R}^5 \left| \begin{array}{c}\mathbf{x}\in FS2\\ y_{1,1}=x_1\\ y_{2,1}=x_1-x_2+5\\ y_{2,2}=x_1+x_2-1\\ y_{2,3}=y_{2,1}y_{2,2}\end{array}\right.\right\}\end{array}\right.$$

$Ex-2$ is made P_1, put in $\mathcal{A}$, and then selected for solution. Preprocessing — minimizing and maximizing the introduced variables in turn and applying *linearity-* and *bilinearity-based range reduction* — improves the bounds on variables to:

$$\mathbf{X}_1 \leftarrow \{(\mathbf{x},\mathbf{y})\in\mathbb{R}^5 : (0,3,0,0,2,0)\le(\mathbf{x},\mathbf{y})\le(2,5,2,2,6,4)\}$$

During this process, the algorithm updates $U\leftarrow 4$ with $\mathbf{x}^*=(0,3)$.

Next, *linearity-* and *bilinearity-based range reduction* are applied to constraint set $FS2'$ but with no improvement on variable bounds. Hence, $\underline{P_1}$ is constructed on $\mathbf{X}_1$:

$$(\underline{P_1})\left|\begin{array}{ll}\text{min} & y_{1,1}+y_{2,3}\\ \text{s. t.} & FS2' := \left\{(\mathbf{x},\mathbf{y})\in\mathbb{R}^5 \left| \begin{array}{c}\mathbf{x}\in FS2\\ y_{1,1}=x_1\\ y_{2,1}=x_1-x_2+5\\ y_{2,2}=x_1+x_2-1\\ 6y_{2,1}+2y_{2,2}-y_{2,3}\le 12\\ 2y_{2,1}-y_{2,3}\le 0\end{array}\right.\right\}\end{array}\right.$$

Solving $\underline{P_1}$ updates $L\leftarrow 2$, $\mathbf{x}^*\leftarrow(0,4)$, and $U\leftarrow 3$. As inequalities $-x_1+2x_2\ge 8$, $x_1\ge 0$, and $y_{1,1}\ge 0$ are active at $\mathbf{x}^*=(0,4)$, the optimality-based range reduction (OBRR) can be performed on these constraints. Here, OBRR is performed only on $x_1\ge 0$ and $y_{1,1}\ge 0$ using their dual multipliers 1 and 1, respectively. Applying OBRR to $x_1\ge 0$ with $\lambda=1$,

$$x_1^U \leftarrow \min\left\{2, 2-\frac{3-2}{1}\right\}$$

and the bounds on x_1 improve to $0\le x_1\le 1$. Likewise, applying OBRR to $y_{1,1}\ge 0$ with $\lambda=1$ improves bounds to $0\le y_{1,1}\le 1$. This improvement on bounds is deemed substantial, so the bounding step of the algorithm is repeated.

The possibility of further tightening of the current formulation of P_1 is first checked with *linearity-* and *bilinearity-based range reduction*. This improves the bounds on variables to:

$$\mathbf{X}_1 \leftarrow \{(\mathbf{x},\mathbf{y})\in\mathbb{R}^5 : (0,4,0,0.99,3,2.95)\le(\mathbf{x},\mathbf{y})$$

$$\leq (0.09, 4.0086, 0.17, 1.00, 3.02, 3)\}$$

When tighter $\underline{P_1}$ is constructed with these new bounds on variables and solved, $L \leftarrow 3$ and $\mathbf{x}^* \leftarrow (0, 4)$. Now, as $U = L$, the algorithm terminates at the root node of the search tree with global optimum $f^* = 3$ at $\mathbf{x}^* = (0, 4)$.

7. Computational Experiments

In this section, we provide extensive computational results with the proposed algorithms for LMP and $GLMP$ on problems from the literature as well as on random instances of problems of various sizes and objective function structures. We also provide comparative computational results with prior approaches for solving LMP and $GLMP$. Our algorithms are coded in Fortran 90 and compiled with O3 option. All our computations are carried out on an IBM RS/6000 Model 43P equipped with a 133 MHz Power PC processor and 64 MB memory. Unless noted otherwise, we use CPLEX 6.0 as the LP solver. We adopt the following notation:

ε_a:	absolute termination criterion (*i.e.*, search process terminates if $U - L \leq \varepsilon_a$)
ε_r:	relative termination criterion (*i.e.*, search process terminates if $U - L \leq \varepsilon_r \lvert L \rvert$)
N_T:	total number of branch-and-reduce iterations (nodes in search tree)
N_O:	node where global optimum is found
N_M:	maximum number of nodes stored during search process
Time:	CPU seconds required in solving a problem
Avg:	average performance by an algorithm on a set of random problems
Std:	standard deviation of performances by an algorithm on a set of random problems

7.1. SMALL TEST PROBLEMS FROM THE LITERATURE

7.1.1. *LMP and GLMP Benchmarks*

We collected LMP and $GLMP$ problems from the literature and solved them using $\varepsilon_a = 10^{-6}$. These results are summarized in Table 1. Clearly, these problems are easy to solve with the proposed algorithms as most are solved at the root node of the branch-and-bound tree. It should be noted that, in many cases, the first LP relaxation of these problems does not provide an exact lower bound. Instead, the LP bound becomes exact only after several rounds of solving this LP and strengthening

Table 1. Computational results on small test problems from the literature

Problem						Proposed algorithm			
Name	Source	Type	t	p_i	(m, n)	N_T	N_O	N_M	Time
FP1	[7]	LMP	1	2	(6,2)	1	0	1	0.0
FP2	[7]	LMP	1	2	(7,2)	1	1	1	0.0
FP3	[7]	$GLMP^*$	1	2	(6,2)	1	0	1	0.0
KK1	[11]	$GLMP$	2	(1,2)	(4,2)	1	1	1	0.0
KK2	[12]	$GLMP$	2	(1,2)	(4,2)	5	0	2	0.0
KKY	[15]	$GLMP$	3	(1,2,2)	(4,2)	3	0	2	0.0
SS1	[31]	$GLMP$	2	(1,2)	(4,2)	17	13	5	0.1
SS2	[31]	$GLMP$	1	(1,2)	(4,2)	1	0	1	0.0
Th	[35]	LMP	1	2	(8,4)	1	0	1	0.0

*The second linear function takes negative values.

it via the optimality-based range reduction technique as illustrated in the previous section.

7.1.2. *Quadratic and Bilinear Programming Benchmarks*

Quadratic and bilinear programs are special cases of $GLMP$. We use a set of separable quadratic programs from [33] and a set of quadratic and bilinear programs from [32] to further test our algorithm for $GLMP$. These problems are solved using $\varepsilon_a = 10^{-6}$. To better compare our results with those reported in [33], we used OSL 2 as the LP solver for this experiment.

Our results, along with those from [33] on separable quadratic programs are summarized in Table 2. Problems and problem names reported in this table are taken from [33]. Under entry (m, n) of this table, we report the sizes of the original problems as well as the LP relaxation solved by the algorithm of [33]. Under entry (m, n, t) of the table, we report the sizes of the LP relaxation problems solved by the proposed algorithm.

Our results, along with those from [32], on quadratic and bilinear programs are recorded in Table 3. Problems and their names used here are taken from [32].

As seen in Tables 2 and 3, our multiplicative programming algorithms are competitive with the specialized QP algorithms for these problems: we usually take fewer nodes for the larger and more difficult problems of these two problem sets. Runs for all algorithms reported in these two tables were done on the same IBM RS/6000 computer.

Table 2. Computational results on separable concave quadratic programming test problems from the literature and comparison with results reported in [33]

Problem		$\underline{GLMP}$	Algorithm of [33]				Proposed algorithm			
Name	(m,n)	(m,n,t)	N_T	N_O	N_M	Time	N_T	N_O	N_M	Time
BSJ2	(5,3)	(20,15,6)	7	0	4	0.1	9	0	3	0.1
BSJ4	(4,6)	(34,30,12)	1	1	1	0.1	5	1	2	0.1
FP1	(1,5)	(26,30,10)	7	7	3	0.2	11	8	4	0.1
FP2	(2,6)	(32,30,12)	1	0	1	0.1	1	1	1	0.1
FP3	(10,13)	(75,65,26)	1	0	1	0.1	1	1	1	0.0
FP4	(5,6)	(35,30,12)	1	1	1	0.1	1	1	1	0.0
FP5	(11,10)	(61,50,20)	5	1	3	0.2	1	1	1	0.2
FP6	(5,10)	(55,50,20)	9	8	4	0.2	7	0	3	0.2
FP7a	(10,20)	(110,10,40)	73	30	9	1.2	43	39	7	1.9
FP7b	(10,20)	(110,100,40)	83	32	9	1.2	39	34	7	1.7
FP7c	(10,20)	(110,100,40)	67	32	9	1.3	41	35	5	1.8
FP7d	(10,20)	(110,100,40)	59	28	9	0.9	53	42	7	2.1
FP7e	(10,20)	(110,100,40)	181	66	21	2.6	153	100	17	4.0
FP8	(10,24)	(130,120,48)	3	3	2	0.2	7	5	2	0.7
KR	(5,2)	(11,10,4)	3	1	2	0.1	7	3	2	0.0
M1	(11,20)	(111,100,40)	187	4	5	2.5	47	5	3	1.9
M2	(21,30)	(171,150,60)	281	1	5	7.1	79	3	2	5.2
Pan1	(4,3)	(19,15,6)	3	3	2	0.1	7	0	2	0.1
Pan2	(1,5)	(26,25,10)	7	2	3	0.1	9	7	2	0.1
PhR1	(5,6)	(35,30,12)	1	0	1	0.1	3	0	2	0.1
PhR2	(5,6)	(35,30,12)	0	0	0	0.0	3	0	2	0.1
PhR3	(5,6)	(35,30,12)	1	1	1	0.1	9	0	2	0.1
PhR11	(4,3)	(19,15,6)	1	1	1	0.1	7	0	2	0.1
PhR12	(4,3)	(19,15,6)	1	1	1	0.1	5	1	2	0.1
PhR13	(10,3)	(25,15,6)	1	1	1	0.1	7	0	2	0.0
PhR14	(10,3)	(25,15,6)	1	0	1	0.1	3	0	2	0.0
PhR15	(4,4)	(24,20,8)	3	0	2	0.1	19	1	4	0.1
PhR20	(9,3)	(24,15,6)	1	1	1	0.1	5	0	2	0.0
RV1	(5,10)	(55,50,20)	33	2	4	0.3	21	11	2	0.3
RV2	(10,20)	(110,100,40)	99	13	12	1.1	51	32	8	1.3
RV3	(20,20)	(120,100,40)	203	23	24	2.4	97	33	10	2.3
RV7	(20,30)	(170,150,60)	161	5	11	2.9	73	5	10	2.9
RV8	(20,40)	(200,180,80)	165	49	18	3.2	101	81	16	5.4
RV9	(20,50)	(270,250,100)	439	90	65	8.0	268	58	33	15.4
Z	(5,3)	(20,15,6)	7	0	4	0.1	7	0	3	0.1

Table 3. Computational results on quadratic and bilinear programming test problems from the literature and comparison with results reported in [32]

Problem		Algorithm of [32]				Proposed algorithm			
Name	(m,n)	N_T	N_O	N_M	Time	N_T	N_O	N_M	Time
AF	(2,2)	21	20	5	0.1	21	15	4	0.1
AF1	(10,10)	25	1	5	0.4	1	1	1	0.1
AF1a	(10,10)	1	1	1	0.1	1	1	1	0.0
AF2	(13,10)	9	0	2	0.3	9	7	3	0.1
BK	(7,8)	25	18	10	0.9	27	19	9	1.2
C-M0	(2,2)	3	0	2	0.1	3	0	2	0.0
C-M1	(5,5)	1	0	1	0.1	3	3	2	0.3
C-M3a	(10,10)	55	23	7	3.6	19	10	7	1.9
C-M3b	(10,10)	13	0	2	0.7	11	0	2	1.6
C-M3c	(10,10)	1	0	1	0.2	1	0	1	0.2
C-M4	(10,10)	1	0	1	0.2	1	0	1	0.1
GI1	(3,3)	1	1	1	0.1	465	465	54	1.9
JK1	(1,4)	92	70	6	0.8	198	195	8	0.5
JK2	(1,3)	171	167	13	0.4	98	88	13	0.2
K1	(6,4)	1	1	1	0.1	1	0	1	0.0
K3	(11,11)	117	0	28	3.5	95	0	12	1.6
SS	(7,4)	163	163	36	1	185	0	46	0.5
Vh1	(4,4)	1	0	1	0.0	1	0	1	0.0
Vh2	(4,4)	1	1	1	0.0	1	1	1	0.0

7.2. COMPARATIVE COMPUTATIONAL RESULTS WITH PRIOR APPROACHES

7.2.1. *Random LMPs*

For extensive testing of our proposed algorithm, we generate random LMPs by using three random problem generation schemes from the literature:

$$(rLMP1)\quad \left|\begin{array}{ll} \text{globmin} & \displaystyle\prod_{j=1}^{p} \mathbf{c}_j^T\mathbf{x} \\ \text{s. t.} & \mathbf{Ax} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}\right.$$

where the real elements of $\mathbf{c}$, $\mathbf{A}$, and $\mathbf{b}$ are pseudo-randomly generated in the range [0,100]. This corresponds to the way [12, 18] generates their random LMP prob-

lems;

$$(rLMP2)\quad \left|\begin{array}{ll} \text{globmin} & \displaystyle\prod_{j=1}^{2}(\mathbf{c}_j^T\mathbf{x}+d_j) \\ \text{s. t.} & \mathbf{Ax}\le\mathbf{b} \\ & \mathbf{x}\ge\mathbf{0} \end{array}\right.$$

whose constraint matrix elements a_{ij} are generated in $[-1, 1]$ via $a_{ij} := 2\alpha - 1$ where α are pseudo-random numbers in [0,1], and the right hand side values are generated via $b_i := \sum_j a_{ij} + 2\beta$, where β are pseudo-random numbers in [0,1]. This agrees with the way random numbers are generated in [35]; and, finally

$$(rLMP3)\quad \left|\begin{array}{ll} \text{globmin} & \displaystyle\prod_{j=1}^{p}\mathbf{c}_j^T\mathbf{x} \\ \text{s. t.} & \mathbf{Ax}\ge\mathbf{b} \\ & \mathbf{1}\le\mathbf{x}\le\mathbf{u} \end{array}\right.$$

where the elements of **c** and **A** are chosen pseudo-randomly from the set of integers $\{1, 2, \ldots, 10\}$ and the elements of **b** and **u** are, respectively, obtained via the expressions $b_i := \sum_j a_{ij}^2$ and $u := \max_{i=1,2,\ldots,m}\{b_i\}$, just as in [3, 20].

For all problems, we solved 10 different random instances for each size and present statistics of the results. In all our computational experiments, we used an IBM RS/6000 Model 43P equipped with a 133 MHz Power PC processor and 64 MB memory. Whenever we present comparative computational results with those obtained by earlier algorithms on different computers, we provide a description of the computational platform used as well as the LINPACK score of each computer as reported in [5]. The latter provides the speed at which a dense system of 100 linear equations is solved using standard programs from LINPACK libraries in a FORTRAN environment.

First, we solve $rLMP1$ using $\varepsilon_a = 10^{-5}$, the termination criterion that is used in [12, 18]. The results are provided in Table 4, alongside those from [12, 18] obtained on a SUN4/75 computer. As seen in the LINPACK scores, the computer used in our study is approximately 60% faster than the one used in [12, 18]. Yet, as Table 4 demonstrates our algorithm is up to two orders of magnitude faster than the algorithms of [12, 18] and we are able to present results on much larger problems.

Table 5 shows the growth of computing time requirements as a function of p for $rLMP1$. In this table, entry r_p is computed by

$$r_p := \frac{\text{AvgTime for } p = i}{\text{AvgTime for } p = 2}$$

Table 4. Computational results on $rLMP1$ and comparison with results reported in [12, 18]

$rLMP1$		Algorithm of [12, 18]	Proposed algorithm	
p	(m,n)	Avg(Std)Time on SUN4/75†	Avg(Std)Time on RS/6000‡	Avg(Std)N_T
2	(20,30)	0.46 (0.05)	0.3 (0.1)	9.0 (3.1)
	(30,50)	1.06 (n/a)	0.8 (0.5)	14.0 (9.8)
	(70,50)	7.37 (n/a)	1.8 (0.7)	15.5 (7.0)
	(80,100)	17.58 (n/a)	3.8 (1.2)	14.9 (6.7)
	(100,100)	*	5.8 (2.0)	17.5 (8.5)
	(120,100)	34.43 (n/a)	8.6 (2.2)	21.3 (7.1)
	(120,120)	*	8.9 (2.7)	15.8 (6.8)
	(130,150)	59.68 (n/a)	14.7 (4.7)	20.5 (9.5)
	(170,150)	105.49 (n/a)	19.8 (5.2)	18.2 (6.5)
	(180,200)	206.64 (n/a)	37.6 (11.5)	18.7 (8.3)
	(200,200)	*	50.1 (13.0)	25.8 (6.2)
	(220,200)	263.04 (n/a)	48.5 (11.7)	20.5 (5.6)
3	(20,30)	1.27 (0.25)	0.8 (0.3)	39.4 (20.2)
	(80,100)	43.98 (9.05)	16.0 (5.0)	86.6 (30.9)
	(100,100)	59.12 (17.80)	25.6 (9.6)	90.6 (24.4)
	(100,120)	115.25 (28.02)	28.9 (10.6)	96.6 (34.2)
	(120,120)	178.57 (43.13)	35.3 (10.0)	82.1 (40.9)
	(120,140)	181.43 (41.12)	43.9 (16.8)	97.3 (37.8)
	(150,140)	381.21 (93.80)	62.1 (17.4)	100.8 (27.3)
	(150,160)	427.02 (127.63)	72.9 (23.3)	105.5 (37.9)
	(200,180)	914.09 (129.88)	122.9 (41.3)	90.4 (48.5)
	(200,200)	*	149.0 (60.2)	87.3 (46.9)
4	(20,30)	14.21 (10.46)	2.6 (0.8)	158.2 (64.8)
	(50,40)	49.05 (46.44)	10.4 (4.0)	222.89 (89.4)
	(50,60)	95.05 (32.49)	13.6 (5.1)	196.9 (76.3)
	(60,80)	155.10 (66.54)	28.1 (6.3)	262.5 (61.7)
	(80,100)	330.55 (101.87)	56.1 (17.2)	294.4 (86.3)
	(100,100)	524.49 (210.27)	61.0 (21.1)	243.8 (117.8)
	(100,120)	617.51 (141.65)	86.1 (35.9)	305.9 (126.8)
	(120,120)	1154.83 (381.51)	94.2 (23.3)	271.4 (70.2)
	(200,200)	*	396.3 (189.4)	301.4 (171.7)
5	(20,30)	1170.36 (950.53)	6.0 (2.0)	370.8 (108.2)
	(100,100)	*	197.9 (38.4)	830.8 (148.3)
	(120,120)	*	245.4 (97.0)	686.0 (285.2)
	(200,200)	*	1381.1 (860.1)	1047.5 (693.1)

†LINPACK score of 4.1.
‡LINPACK score of 6.7.
n/a, Not provided in [12, 18].
*Problems of this size not solved in [12, 18].

Table 5. Growth in computing time as a function of p

$rLMP1$	Normalized AvgTime							
	Algorithm of [18]				Proposed Algorithm			
(m,n)	r_2	r_3	r_4	r_5	r_2	r_3	r_4	r_5
(20,30)	1	3	31	2544	1	3	10	23
(100,100)	$*$	$*$	$*$	$*$	1	4	10	34
(120,120)	$*$	$*$	$*$	$*$	1	4	11	28
(200,200)	$*$	$*$	$*$	$*$	1	3	8	28

$*$: Problems of this size not solved in [18].

Table 6. Computational results on $rLMP2$ ($p = 2$) and comparison with results reported in [35]

$rLMP2$	Algorithm of [35]		Proposed algorithm	
(m,n)	Time on PS2†	N_T	Avg(Std)Time on RS/6000‡	Avg(Std)N_T
(10,20)	1.85	9	0.1 (0.1)	6.2 (4.3)
(20,20)	3.65	16	0.2 (0.1)	7.0 (2.8)
(22,20)	4.05	14	0.2 (0.1)	8.8 (4.2)
(20,30)	5.04	16	0.3 (0.1)	8.0 (3.6)
(35,50)	19.88	20	1.0 (0.4)	11.0 (3.5)
(45,60)	81.22	26	1.2 (0.3)	13.3 (4.9)
(45,100)	240.50	30	3.9 (1.2)	15.2 (6.0)
(60,100)	290.12	30	5.6 (1.2)	14.8 (3.8)
(70,100)	511.38	30	6.5 (3.0)	17.5 (7.2)
(70,120)	560.75	31	9.0 (1.8)	17.2 (4.8)
(100,100)	635.06	32	7.6 (1.0)	13.3 (4.3)
(102,150)	2823.13	40	15.9 (2.9)	24.8 (7.0)
(102,190)	3187.93	40	21.4 (3.5)	28.4 (7.5)
(72,199)	2298.88	45	18.3 (6.2)	25.5 (8.3)
(110,199)	8068.84	49	22.7 (3.0)	21.7 (5.7)

†: LINPACK score of 0.15.
‡: LINPACK score of 6.7.

for $i = 2$, 3, 4, and 5 for each problem size. For comparison, we provide the same information for the algorithm of [18] in Table 5. Clearly, our algorithm exhibits much slower growth of computational requirements.

$rLMP2$ are next solved using $\varepsilon_a = 10^{-6}$ as in [35]. Table 6 summarizes our computational results and those reported in [35]. The results from [35] are obtained

Table 7. Comparative results on $rLMP2$

	Normalized values			
$rLMP2(p=2)$	Algorithm of [35]		Proposed algorithm	
(m,n)	Time	N_T	(Avg)Time	(Avg)N_T
(10,20)	1	1	1	1
(20,20)	2	2	1	1
(20,30)	3	2	2	1
(45,60)	44	3	8	2
(70,100)	276	3	46	3
(100,100)	343	4	54	2
(102,150)	1526	4	114	4
(110,199)	4362	5	162	4

on an IBM-PS2 Model 80 computer (much slower than our computer as shown by the LINPACK scores). Comparative results on $rLMP2$ by the two algorithms are provided in Table 7, whose entries are the normalized requirements in (average) CPU seconds and the total number of iterations with respect to the ones for 10×20 $rLMP2$ problem obtained by computing the ratio:

$$\frac{\text{Time } (N_T) \text{ on } m \times n \text{ problem}}{\text{Time } (N_T) \text{ on } 10 \times 20 \text{ problem}}$$

Table 7 also shows that our algorithm possesses a good average-case performance. For example, compare the very first and the last (Avg)Time r_2 entries in this table for our algorithm. We see that our algorithm requires only 162 times more computing time for solving problems that are 109 times larger than the base problems in terms of the constraint matrix size. This is almost linear. In contrast, the CPU requirements of the algorithm of [35] grow an order of magnitude faster.

We next compare our (exact) algorithm to two heuristic methods proposed for LMP from the literature [3, 20]. For this experiment, we solve various sized $rLMP3$ using two termination criteria, $\varepsilon_r = 0.005$ and $\varepsilon_a = 10^{-5}$. Our results as well as those from [3, 20] are summarized in Table 8. In this table, r for heuristic solutions denotes the average efficiency rating defined by

$$r := 1 - \frac{z_{\mathrm{H}} - z_{\min}}{z_{\max} - z_{\min}}$$

where z_{H} is the objective function value returned by the heuristic methods of [3, 20], z_{min} and z_{max} are the global minimum and maximum, respectively, of the problem over the corresponding set of efficient extreme points $\mathbf{M}_{E,ex} := \mathbf{M}_E \cap \mathbf{M}_{ex}$, where $\mathbf{M}_E$ and $\mathbf{M}_{ex}$ are the set of efficient solutions and the set of extreme points of

Table 8. Computational results on *rLMP*3 and comparison with heuristic results reported in [3, 20].

*rLMP*3		Heuristic of [3]		Heuristic of [20]		Proposed algorithm	
p	(m, n)	r	Time[1]	r	Time[2]	AvgTime[3] ($\varepsilon_r = 0.005$)	AvgTime[3] ($\varepsilon_a = 10^{-5}$)
2	(25,20)	1.000	0.227	1.000000	0.25	0.1	0.2
	(25,30)	1.000	0.241	0.999995	0.42	0.1	0.2
	(30,40)	1.000	0.389	1.000000	0.78	0.2	0.4
	(40,30)	0.999	0.328	0.999997	0.61	0.2	0.3
	(40,50)	1.000	0.504	0.999991	1.66	0.4	0.5
	(50,40)	0.999	0.453	0.999936	1.45	0.4	0.6
	(50,60)	1.000	0.556	0.998955	4.17	0.6	1.0
	(60,70)	1.000	1.070	0.992580	7.45	0.8	1.2
3	(25,20)	0.985	0.321	0.999991	0.34	0.2	0.4
	(25,30)	0.960	0.469	0.999935	0.39	0.3	0.6
	(30,40)	0.987	0.543	0.999802	1.22	0.7	1.2
	(40,30)	0.993	0.609	0.999971	0.91	0.5	1.0
	(40,50)	0.920	0.967	0.997156	2.02	1.0	1.7
	(50,40)	0.993	0.908	0.998923	1.68	0.8	1.4
	(50,60)	0.995	1.298	0.992008	4.26	1.7	3.3
	(60,70)	0.969	2.495	$*$	$*$	2.0	4.07
4	(25,20)	0.998	0.426	0.999912	0.38	0.5	1.0
	(25,30)	0.992	0.598	0.999022	1.11	0.8	1.3
	(30,40)	0.986	1.019	0.999941	1.25	1.5	2.9
	(40,30)	0.978	0.998	0.997706	1.19	1.0	1.8
	(40,50)	0.969	1.539	0.998237	2.17	2.8	5.0
	(50,40)	0.969	1.587	0.990254	1.84	2.5	4.2
	(50,60)	0.980	1.901	$*$	$*$	4.4	8.9
5	(10,20)	0.993	0.353	0.999994	1.84	0.5	0.8
	(20,10)	0.998	0.294	0.999159	0.16	0.3	0.5
	(25,30)	0.995	0.962	0.987431	2.23	1.7	3.1

[1]CPU seconds on IBM ES/9000 (LINPACK score of 86).
[2]CPU seconds on HP Apollo 715/75 (LINPACK score of 29).
[3]CPU seconds on RS/6000 (LINPACK score of 6.7).
$*$Problems of this size not solved in [20].

Table 9. Comparative results on $rLMP3$ ($p = 3$)

$rLMP3$ ($p = 3$)	Normalized AvgTime		
(m, n)	Heuristic of [3]	Heuristic of [20]	Proposed algorithm
(25,20)	1	1	1
(25,30)	2	1	2
(30,40)	2	4	3
(40,30)	2	3	2
(40,50)	3	6	5
(50,40)	3	5	4
(50,60)	4	13	8
(60,70)	8	$*$	9

$*$Problems of this size not solved in [20].

the feasible set $\mathbf{M}$ ($= \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} \geq \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$) of $rLMP3$. Hence, the closer r is to 1, the better the quality of the heuristic solution is. Computational results in [3, 20] are, respectively, obtained on an IBM ES/900 Model 831 computer and on an HP Apollo Model 715/15 computer. Both are faster than our computer as shown by the LINPACK scores.

The performance of the three algorithms as a function of the problem size for $p = 3$ is shown in Table 9. The entries in this table are the normalized average CPU times on $m \times n$ $rLMP3$ with respect to the average time on 25×20 $rLMP3$ for $p = 3$ by the three algorithms:

$$\frac{\text{(Avg)Time on } m \times n \text{ problem}}{\text{(Avg)Time on } 25 \times 20 \text{ problem}}$$

One can see from these results that the computational requirements of our global optimization algorithm grow similarly to those of the heuristic procedures developed for solving LMP.

7.2.2. *Random GLMPs*

We follow the random number generation scheme of [18, 15] and generate the elements of $\mathbf{c}_{ij}$, $\mathbf{A}$, and $\mathbf{b}$ pseudo-randomly from the range [0,100] and set $d_{i,j} = 0$. These problems are solved using $\varepsilon_a = 10^{-5}$, the same termination criterion as the one used in [18, 15]. Our computational results are provided in Table 10, alongside those from [18, 15]. The results reported in [18, 15] are obtained on a SUN4/280S workstation. Clearly, we are able to solve much larger problems than those reported in the past.

In order to examine how the average-case performance of our algorithm for $GLMP$ is affected by problem size, we provide Table 11. For each of the problem sizes $(m, n) = (30, 20)$, $(30, 50)$, and $(70, 50)$, an entry $r_{i,p}$ ($i = 2, 3, 4$) in Table

Table 10. Computational results on randomly generated $GLMP$ and comparison with results reported in [18, 15]

$GLMP$			Algorithm of	Proposed algorithm	
t	p	(m, n)	[18, 15] on SUN4/280S†	Avg(Std)Time on RS/6000‡	Avg(Std)N_T
2	(1,2)	(150,100)	48.8 (n/a)	26.4 (3.9)	60.7 (10.6)
		(150,150)	65.7 (n/a)	45.2 (13.8)	70.8 (20.3)
		(200,150)	118.7 (n/a)	61.4 (30.3)	64.0 (30.8)
		(200,200)	145.9 (n/a)	86.8 (37.6)	67.8 (35.6)
		(250,200)	230.9 (n/a)	103.2 (24.2)	63.8 (14.1)
		(250,250)	316.9 (n/a)	141.2 (14.5)	73.5 (8.9)
		(300,250)	416.0 (n/a)	169.5 (61.8)	72.3 (22.6)
		(300,300)	454.0 (n/a)	294.5 (125.0)	96.6 (32.4)
		(350,300)	637.1 (n/a)	276.5 (164.9)	73.6 (29.6)
	(2,2)	(30,20)	5.4 (1.8)	2.3 (0.6)	121.7 (28.9)
		(30,50)	25.9 (5.1)	9.0 (4.3)	176.0 (75.6)
		(70,50)	55.6 (14.8)	20.3 (8.2)	211.6 (74.1)
	(3,3)	(30,20)	*	17.5 (6.0)	827.8 (297.8)
		(30,50)	*	73.6 (41.9)	1326.9 (754.4)
		(70,50)	*	180.8 (54.5)	1997.6 (668.2)
	(4,4)	(30,20)	*	88.7 (43.1)	3387.8 (1627.4)
		(30,50)	*	715.0 (404.0)	12702.9 (6996.3)
		(70,50)	*	1936.6 (1028.7)	14984.3 (5487.8)
3	(2,2,2)	(30,20)	49.3 (33.1)	7.8 (3.4)	388.5 (179.1)
		(30,50)	202.7 (74.2)	23.9 (13.6)	457.9 (251.3)
		(70,50)	1087.7 (900.4)	57.3 (28.0)	673.2 (378.6)
	(3,3,3)	(30,20)	*	80.7 (27.2)	3417.8 (1044.2)
		(30,50)	*	733.9 (605.0)	12321.0 (7672.4)
		(70,50)	*	1323.1 (468.7)	13638.4 (4683.6)
	(4,4,4)	(30,20)	*	1333.6 (1234.9)	29417.2 (12623.5)
4	(2,2,2,2)	(30,20)	416.5 (233.2)	12.2 (4.3)	603.6 (234.5)
		(30,50)	3897.6 (2158.6)	58.2 (25.9)	1256.2 (596.9)
		(70,50)	*	141.5 (62.2)	1734.0 (734.4)
	(3,3,3,3)	(30,20)	*	546.0 (269.6)	18469.9 (7058.1)
		(30,50)	*	3620.9 (2053.6)	50751.7 (22798.2)
		(70,50)	*	12016.0 (4988.7)	84138.3 (27161.2)
	(4,4,4,4)	(30,20)	*	4872.5 (1811.0)	131316.9 (47267.9)

†LINPACK score of 4.1.
‡LINPACK score of 6.7.
*Problems of this size not solved in [18, 15].

Table 11. Growth in computing time as a function of t for proposed algorithm for $GLMP$, and comparison with results reported in [15]

		Normalized AvgTime					
$GLMP$		Algorithm of [15]			Proposed Algorithm		
p	(m,n)	$r_{2,p}$	$r_{3,p}$	$r_{4,p}$	$r_{2,p}$	$r_{3,p}$	$r_{4,p}$
2	(30,20)	1	9	77	1	3	5
	(30,50)	1	8	150	1	3	6
	(70,50)	1	20	$*$	1	3	7
3	(30,20)	$*$	$*$	$*$	1	5	31
	(30,50)	$*$	$*$	$*$	1	10	49
	(70,50)	$*$	$*$	$*$	1	7	66
4	(30,20)	$*$	$*$	$*$	1	15	55

$*$: Problems of this size not solved in [15].

Table 12. Growth in computing time as a function of p for proposed algorithm for $GLMP$, and comparison with results reported in [15]

		Normalized AvgTime					
$GLMP$		Algorithm of [15]			Proposed Algorithm		
t	(m,n)	$r_{t,2}$	$r_{t,3}$	$r_{t,4}$	$r_{t,2}$	$r_{t,3}$	$r_{t,4}$
2	(30,20)	1	$*$	$*$	1	8	38
	(30,50)	1	$*$	$*$	1	8	79
	(70,50)	1	$*$	$*$	1	9	95
3	(30,20)	1	$*$	$*$	1	10	171
4	(30,20)	1	$*$	$*$	1	45	399

$*$: Problems of this size not solved in [15].

11 represents

$$r_{i,p} = \frac{\text{AvgTime for } t = i \text{ for } p}{\text{AvgTime for } t = 2 \text{ for } p}$$

for each fixed value of $p = 2, 3$, and 4. Likewise, each entry $r_{t,i}$ $(i = 2, 3, 4)$ in Table 12 is computed by

$$r_{t,i} = \frac{\text{AvgTime for } p = i \text{ for } t}{\text{AvgTime for } p = 2 \text{ for } t}$$

for each fixed value of $t = 2, 3$, and 4.

For comparison with other approaches, we provide $r_{t,2}$ $(t = 2, 3, 4)$ values obtained from the results reported in [15] in the lower parts of Tables 11 and 12. Clearly, our algorithms exhibit a slower growth of computational requirements as the problem size increases.

8. Conclusions

In this paper, we developed specialized global optimization algorithms for multiplicative programs based upon bounding of monomial functions and a new greedy branching principle for rectangular branch-and-bound. Extensive computational experiments with the proposed algorithms on problems from the literature and also randomly generated instances suggest that our algorithms are very efficient for solving multiplicative programs. These algorithms make possible for the first time the solution of large and complex multiplicative programs to global optimality. Even when compared with a specialized algorithm for solving quadratic and bilinear programming problems and when tested against heuristic approaches for linear multiplicative programs, the proposed algorithms performed competitively.

Acknowledgements

The authors acknowledge partial financial support from the National Science Foundation under award ECS-0098770.

References

1. Bennett, K.P. (1994), Global tree optimization: A non-greedy decision tree algorithm. *Computing Sciences and Statistics* 26, 156–160.
2. Bennett, K.P. and Mangasarian, O.L. (1994), Bilinear separation of two sets in $n-$space. *Computational Optimization and Applications* 2, 207–227.
3. Benson, H.P. and Boger, G. M. (1997), Multiplicative programming problems: analysis and efficient point search heuristic. *Journal of Optimization Theory and Applications* 94(2), 487–510.
4. Benson, H.P. and Boger, G.M. (2000), Outcome-space cutting-plane algorithm for linear multiplicative programming. *Journal of Optimization Theory and Applications* 104(2), 301–322.
5. Dongarra, J.J. Performance of Various Computers Using Standard Linear Equations Software. Technical Report Technical Report CS-85-89, Computer Science Department at University of Tennessee, Knoxville, TN and Mathematical Sciences Section at Oak Ridge National Laboratory, Oak Ridge, TN 37831, 1985.

6. Dorneich, M.C. and Sahinidis, N.V. (1995), Global optimization algorithms for chip design and compaction. *Engineering Optimization* 25(2), 131–154.
7. Falk, J.E. and Palocsa, S.W. (1994), Image space analysis of generalized fractional programs. *Journal of Global Optimization* 4(1), 63–88.
8. Henderson, J.M. and Quandt, R.E. (1971), *Microeconomic Theory*. 2nd edition, McGraw-Hill, New York.
9. Keeney, R.L. and Raiffa, H. (1993), *Decisions with Multiple Objective*. Cambridge University Press, Cambridge, MA.
10. Konno, H. and Kuno, T. (1989), Linear Multiplicative Programming. Technical Report IIISS 89-13, Institute of Human and Social Sciences, Tokyo Institute of Technology, Tokyo, Japan.
11. Konno, H. and Kuno, T. (1990), Generalized linear multiplicative and fractional programming. *Annals of Operations Research* 25, 147–162.
12. Konno, H. and Kuno, T. (1992), Linear multiplicative programming. *Mathematical Programming* 56, 51–64.
13. Konno, H. and Kuno, T. (1995), Multiplicative programming problems. In: R. Horst and P.M. Pardalos, (eds.), *Handbook of Global Optimization*,Kluwer Academic Publisher, Norwell, MA. p.369–405.
14. Konno, H., Kuno, T. and Yajima, Y. (1992), Parametric simplex algorithms for a class of $\mathcal{NP}-$complete problems whose average number of steps is polynomial. *Computational Optimization and Applications* 1, 227–239.
15. Konno, H., Kuno, T. and Yajima, Y. (1994), Global optimization of a generalized convex multiplicative function. *Journal of Global Optimization* 4, 47–62.
16. Konno, H., Shirakawa, H. and Yamazaki, H. (1993), A mean-absolute deviation-skewness portfolio optimization model. *Annals of Operations Research* 45, 205–220.
17. Konno, H., Yajima, Y. and Matsui. T. (1991), Parametric simplex algorithms for solving a special class of nonconvex minimization problems. *Journal of Global Optimization* 1, 65–81.
18. Kuno, T., Yajima, Y. and Konno, H. (1993), An outer approximation method for minimizing the product of several convex functions on a convex set. *Journal of Global Optimization* 3(3), 325–335.
19. Liu, M.-L., Sahinidis, N.V. and Shectman, J.P. (1996), Planning of chemical process networks via global concave minimization. In: I.E. Grossmann, (ed.), *Global Optimization in Engineering Design*, Vol. 9, *Nonconvex Optimization and Its Applications*, Kluwer Academic Publishers, Norwell, MA, p. 195–230.
20. Liu, X.J., Umegaki, T. and Yamamoto, Y. (1999), Heuristic methods for linear multiplicative programming. *Journal of Global Optimization*, 4(15), 433–447.
21. Maling, K., Mueller, S.H. and Heller, W.R. (1982), On finding most optimal rectangular package plans. In: *Proceedings of the 19th Design Automation Conference*, p. 663–670.
22. Maranas, C.D., Androulakis, I.P., Floudas, C.A., Berger, A.J. and Mulvey J.M.(1997), Solving long-term financial planning problems via global optimization. *Journal of Economic Dynamics & Control*, 21, 1405–1425.
23. Markowitz, H.M. (1991), *Portfolio Selection*. Basil Blackwell Inc., Oxford, 2nd edition.
24. Matsui, T. (1996), $\mathcal{NP}-$Hardness of linear multiplicative programming and related problems. *Journal of Global Optimization* 9(2), 113–119.
25. McCormick, G.P. (1983), *Nonlinear Programming. Theory, Algorithms, and Applications*. Wiley Interscience, New York.
26. Mulvey, J.M., Vanderbei, R.J. and Zenios, S.A. (1995), Robust optimization of large-scale systems. *Operations Research* 43, 264–281.
27. Pardalos, P.M. (1990), Polynomial time algorithms for some classes of constrained quadratic problems. *Optimization* 21(6), 843–853.
28. Quesada, I. and Grossmann, I.E. (1996), Alternative bounding approximations for the global optimization of various engineering design problems. In I.E. Grossmann, (ed.), *Global Optim-*

ization in Engineering Design, Vol. 9 *Nonconvex Optimization and Its Applications*, Kluwer Academic Publishers, Norwell, MA, p 309–331.
29. Ryoo, H.S. and Sahinidis, N.V. (1996), A branch and reduce approach to global optimization. *Journal of Global Optimization* 8(2), 107–138.
30. Ryoo, H.S. and Sahinidis, N.V. (2001), Analysis of bounds for multilinear functions. *Journal of Global Optimization* 19(4), 403–424,
31. Schaible, S. and Sodini, C. (1995), Finite algorithm for generalized linear multiplicative programming. *Journal of Optimization Theory and Applications* 87(2), 441–455.
32. Shectman, J.P. (1999), *Finite Algorithms for Global Optimization of Concave Programs and General Quadratic Programs*. PhD thesis, University of Illinois at Urbana, 1999.
33. Shectman, J.P. and Sahinidis, N.V. (1998), A finite algorithm for global minimization of separable concave programs. *Journal of Global Optimization* 12(1), 1–36.
34. Swarup, K. (1966), Quadratic Programming. *Cahiers du Centre d'Études de Recherche Opérationnelle* 8, 223–234.
35. Thoai, N.V. (1991), A global optimization approach for solving the convex multiplicative programming problems. *Journal of Global Optimization* 1(4), 341–357.