, , 1–17 ()

Distributed Decomposition–based Approaches in Global Optimization

I.P. ANDROULAKIS, V. VISWESWARAN AND C.A. FLOUDAS*
Department of Chemical Engineering, Princeton University, Princeton, NJ 08544–5263

Abstract. Recent advances in the theory of deterministic global optimization have resulted in the development of very efficient algorithmic procedures for identifying the global minimum of certain classes of nonconvex optimization problems. The advent of powerful multiprocessor machines combined with such developments make it possible to tackle with substantial efficiency otherwise intractable global optimization problems. In this paper, we will discuss implementation issues and computational results associated with the distributed implementation of the decomposition–based global optimization algorithm, **GOP**, [5], [6]. The NP-complete character of the global optimization problem, translated into extremely high computational requirements, had made it difficult to address problems of large size.The parallel implementation made it possible to successfully tackle the increased computational requirements in in order to identify the global minimum in computationally realistic times. The key computational bottlnecks are identified and properly addressed. Finaly, results on an **Intel-Paragon** machine are presented for large scale Indefinite Quadratic Programming problems, with up to 350 quadratic variables, and Blending–Pooling Problems, with up to 12 components and 30 qualities.

Keywords: Global optimization, bilinear programming, blending, pooling, indefinite quadratic optimization

1. Introduction

The subject of the global optimization of nonconvex constrained problems has received significant attention [3], [4]. The existing approaches can be largely classified as *deterministic* and *probabilistic*. Deterministic methods include : Lipschitzian methods, [9]; Branch and Bound methods, [1]; Cutting Plane methods, [7]; Difference of Convex Function methods, [15]; Outer Approximation methods [10]; Reformulation-Linearization methods [14]; Interval methods, [8]. The probabilistic methods include among others : Random Searches, [11]; Clustering methods, [13].

Recently, a deterministic primal-relaxed dual global optimization method was proposed [5], [6], [17], for certain classes of nonconvex optimization problems. Employing duality theory, a global optimization algorithm, GOP, was developed. Through a series of *primal* and *relaxed dual* problems, that provide valid upper and lower bounds on the global solution, the global minimum is identified. The algorithm was shown to attain finite ϵ-convergence and ϵ-global optimality. Important theoretical properties, that exploit further the structure of the Lagrange function, [16], significantly enhanced the performance of GOP.

In this paper we present a *distributed* implementation of GOP that enhances the computational efficiency of the method. The proposed approach is applied to large scale Indefinite Quadratic Problems and large scale Blending and Pooling problems. Section 2 presents a brief review of the GOP method. In Section 3 the computationally intensive tasks of the

* Author to whom all correspondence should be addressed.

method are isolated and the proposed distributed computing implementation is discussed. Section 4 addresses the application of the method to large scale Indefinite Quadratic problems, and Section 5 presents results on quadratically constrained problems that correspond to large scale Blending and Pooling problems.

2. Review of the GOP Algorithm

The global optimization problem addressed by the GOP algorithm is stated as follows : Determine an ϵ–globally optimal solution of:

$$\min_{x,y} \; f(x,y)$$

$$\begin{aligned} s.t. \quad g(x,y) &\leq 0 \\ h(x,y) &= 0 \\ x \in X, \quad y &\in Y \end{aligned}$$

where X and Y are non–empty, compact, convex sets, $g(x,y)$ is an m– vector of inequality constraints, and $h(x,y)$ is a p–vector of equality constraints. The function $f(x,y)$, $g(x,y)$, and $h(x,y)$ are continuous, piecewise differentiable and are given in analytical form in $X \times Y$. The variables x and y are defined such that the following Condition (A) is satisfied:

CONDITION (A): For fixed $y = y^k$, $f(x,y)$ and $g(x,y)$ are convex in x, and $h(x,y)$ is affine in x, and for every $x = x^k$, $f(x,y)$ and $g(x,y)$ are convex in y , and $h(x,y)$ is affine in y .

Making use of duality theory, a global optimization algorithm, GOP, was proposed for the solution of the problem through a series of sub–problems that provide valid upper and lower bounds to the global solution. The GOP is a decomposition-based algorithm that decomposes the original problem into **primal** and **relaxed dual** subproblems. By projecting on the y variables, the primal problem takes the form :

$$v(y^k) = \min_{x} \quad f(x, y^K)$$

$$\begin{aligned} s.t. \quad g(x,y^K) &\leq 0 \\ h(x,y^K) &= 0 \end{aligned}$$

At the same time, the solution of the primal problem provides Lagrange multipliers λ^k, μ^k, for the inequality and equality constraints respectively. The Lagrange multipliers are subsequently used to formulate the Lagrange function $L(x, y, \lambda^K, \mu^K) = f(x,y) + \mu^K g(x,y) + \lambda^K h(x,y)$. By making use of several theoretical properties it was proved, [5], [6], that the solution of the dual problem corresponds to the solution of a series of **relaxed dual** problems in the y-space. The y-space is partitioned into sub–domains and each relaxed dual problem represents a valid underestimator of the original non–convex problem for that particular domain. Each relaxed dual problem is associated with a combination B_j of bounds of those x variables that appear in bilinear, xy, products in the Lagrange function. These x variables are denoted as *connected* variables. Each relaxed dual problem takes the form:

$$\overline{\mu}_B(K) = \left\{ \begin{array}{l} \min\limits_{y \in Y, \mu_B} \ \mu_B \\ s.t. \\ \mu_B \geq L(x^{B_j}, y, \lambda^k, \mu^k)|_{x^k}^{lin} \\ \nabla_{x_i} L(x^{B_j}, y, \lambda^k, \mu^k)|_{x^k}^{lin} \leq 0 \ x_i^{B_j} = x_i^U \\ \nabla_{x_i} L(x^{B_j}, y, \lambda^k, \mu^k)|_{x^k}^{lin} \geq 0 \ x_i^{B_j} = x_i^L \end{array} \right\} \begin{array}{l} B_j \in CB \\ k \leq K \\ j \in U_{k,K} \end{array}$$

The first family of constraints represents the Lagrange underestimating cut while the next two families define the partitioning of the y-space. The constraints involving the gradients of the Lagrange function are denoted as *qualifying constraints*. They define the partitioning in the y-space. Furthermore a particular x variable is connected in and only if : $\nabla_{x_i} L(x^{B_j} y, \lambda^k, \mu^k)|_{x^k}^{lin}$ is a function of y. As it can be seen, solving the relaxed dual problem in the K^{th} iteration is equivalent to setting the x variables to a combination of their bounds, B_j, and solving for the corresponding domain of the y variables. Selecting the minimum of all these problems and the corresponding solutions from previous iterations a valid current lower bound on the global minimum is obtained. Once such a solution has been identified, the values of the y variables are updated and the primal problem is resolved. By solving the primal problem and updating the upper bound as the minimum solution found, a monotonically non–increasing sequence of upper bounds is generated. Solving the relaxed dual problems, a non-decreasing sequence of valid lower bounds is generated due to the accumulation of the previous constraints. As a result, the GOP algorithm attains finite convergence to an ϵ-global solution through successive iterations between the primal and relaxed dual problems. Clearly, the computational bottleneck of the algorithm manifests itself in the solution of 2^{NI_c} problems, where NI_c is the number of connected variables. The connected x variables form a sub–set of the original x–type variables. Additional theoretical properties, [16], allowed the substantial reduction of the number of connected variables at each iteration of the GOP algorithm.

3. Critical Computational Issues

Based on the analysis just presented, we can clearly identify that the GOP algorithm can potentially become very intensive from the computational point of view.

It was shown theoretically, [16], and observed computationally that obtaining tight bounds on the optimization variables, for both the x–type as well as the y–type, is very helpful in the convergence rate of the algorithm. As it is discussed in Section 3.1, in order to calculate tight variable bounds one has to solve $2(N_x + N_y)$ convex NLP's, where N_x and N_y if the total number of x–type and y–type variables respectively. Therefore, the search for tighter variable bounds problems can be computationally improved is these problems are solved in parallel.

The major computational bottleneck of the method is the solution of a potentially very large number of relaxed dual problems at a given iteration, 2^{NI_c}. Therefore, major emphasis has to be placed on the most efficient solution, from the computational point of view, of this large number of convex or linear optimization problems. To this end, the development of a distributed computing implementation of the solution of the relaxed dual problems is of primary importance.

Finally, issues related to the routing of the appropriate data, once a lower bound has been identified, will also be addressed. Such issues require the implementation of a parallel *routing/sorting* algorithm.

Figure 8 depicts the basic steps of the distributed implementation of the GOP algorithm and highlights the parallelized steps.

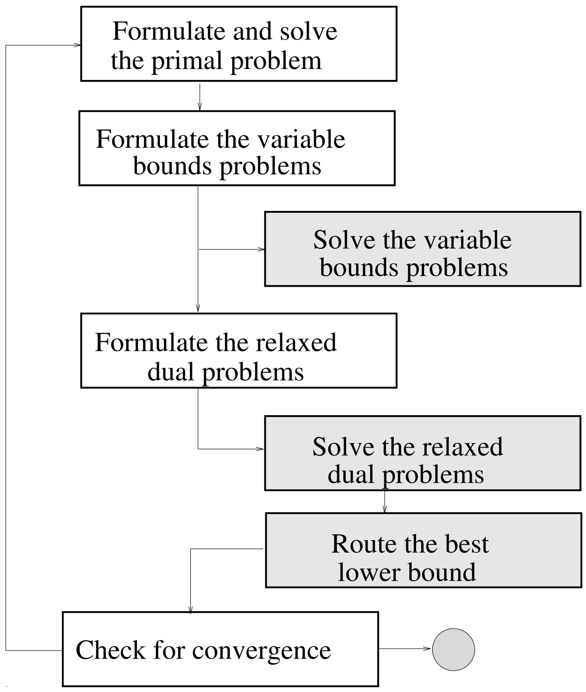

Figure 1. Flow Diagram of distributed GOP

3.1. Updating the Variable Bounds

Based on the analysis of the GOP algorithm, the set of problems that have to be solved at the K^{th} iteration, in order to provide a valid lower bounds on the global minimum, has the following form :

$$\overline{\mu}_{\mathrm{B}}^{\mathrm{K}} \geq \min_{B_j \in CB} \left\{ \begin{array}{l} \min\limits_{y \in Y, \mu_B} \ \mu_B \\ s.t. \\ Previous\ Lagrange\ Functions \\ Previous\ Qualifying\ Constraints \\ \mu_B \geq L(x^{B_j}, y, \lambda^k, \mu^k)|_{x^k}^{lin} \\ \nabla L(x^{B_j}, y, \lambda^k, \mu^k)|_{x^k} \leq 0 \quad if \ \ x_i^{B_j} = x_i^U \\ \nabla L(x^{B_j}, y, \lambda^k, \mu^k)|_{x^k} \geq 0 \quad if \ \ x_i^{B_j} = x_i^L \end{array} \right.$$

Therefore, one observes that the quality of the lower bound $\overline{\mu}_{\mathrm{B}}^{\mathrm{K}}$ depends qualitatively on the underestimating constraint on μ_B, and computationally on the number of problems that have to be solved, 2^{NI_c}. The following observations will establish the connection between the quality of the variable bounds and the aforementioned issues. To determine the number of connected variables at each iteration K we identify any sign changes of the gradient of the linearization of the Lagrange function around the solution of the primal problem, x^k, for the current bounds on x and y variables. To detect whether a particular qualifying constraint changes sign, one has to identify upper and lower bounds on the qualifying constraints over the interval of interest. Clearly, the sign of the qualifying constraints, which detects the existence of a connected variable, is strongly affected by the range of the variable bounds. Furthermore, it can be observed that the current Lagrange cut, $\mu_B \geq L(x^{B_j}, y, \lambda^k, \mu^k)|_{x^k}^{lin}$, is the linearization of the Lagrange function at the solution of the primal problem. Therefore, the quality of this underestimator strongly depends on the quality of the corresponding variable bounds. Consequently, it is very important for the computational efficiency of the algorithm, to obtain the tightest possible bounds of the x and the y variables.

In order to identify the tightest possible variable bounds, we have to calculate the maximum and minimum possible values of all the variables within the current domain of interest. Based on the partitioning induced by the GOP algorithm, the domain of interest for the solution of the relaxed dual problem, is defined by three set of constraints : (a) original convex constraints, (b) Original convexified constraints, and (c) previous qualifying constraint. Sets (a) and (b), define implicitly the range of variables with respect to the original problem. Obviously, any convex constraint, that is convex inequality and/or affine equality, will not alter the convexity characteristics of the problem and thus can be used. Any convexification of the original non–convex constraints will be an overestimation of the feasible region, and it would restrict the domain for the purpose of identifying tighter variable bounds. In addition, the current domain of interest, over which the new lower bound will be sought, is implicitly defined by the set of the previous qualifying constraints. Hence, the optimization problems, whose solution will define the current tightest possible lower and upper bounds of the optimization variables is:

$$\begin{array}{ll} \min\limits_{x \in X_i}, \min\limits_{x \in Y_i} y_i & \max\limits_{x \in X_i}, \max\limits_{x \in Y_i} y_i \\ s.t. & s.t. \\ Original\ Convex\ Constraints & Original\ Convex\ Constraints \\ Original\ Convexified\ Constraints & Original\ Convexified\ Constraints \\ Previous\ Qualifying\ Constraints & Previous\ Qualifying\ Constraints \end{array}$$

It was also observed computationally that the frequency at which these problems are solved can be treated as a decision variable. In other words, for certain classes of problems, (e.g. indefinite quadratic), computing tight bounds once at the very beginning was adequate, whereas for other classes of problems, (e.g. pooling and blending), the variable bounds had to be updated at each iteration. It is clear that the total number of variable bounds problems that have to be solved are $2(N_x + N_y)$, implying that for large scale optimization problems the framework of distributed computing is needed. With respect to the implementation, first we identify whether it is worth solving the bounds problems in parallel (a minimum of 10 problems per processing element is assumed). Then, the vector of variables is divided into smaller groups and these groups are assigned to nodes who are responsible for solving the variable bounds problems associated with variables. The master node is then collecting the partial vector. The collection process has an unavoidable sequential character but the gains from solving the variable bounds in parallel outperform any potential performance degradation.

3.2. Solving the Relaxed Dual Problems

The parallel solution of the relaxed dual problems aims at addressing the need to reduce the computational burden associated with the solution of 2^{NIc} problems at each iteration. At each iteration a large number of problems are solved, and probably a large number of them will be feasible and will represent valid lower bounds for that particular domain. As the algorithm proceeds and better lower and upper bounds are generated, most of these solutions will not be needed. Nevertheless, every feasible solution that is generated has to be stored at least temporarily, in order to guarantee the convergence of the GOP algorithm. As a result, an efficient implementation of the GOP needs to address the issues related to the storage of the generated solutions. Parallel computing architectures can address effectively this problem allowing for the distributed storage of the generated lower bounding solutions.

Based on the theoretical analysis of the method, it is clear that all the relaxed dual problems that have to be solved, have the same functional form, and only the bound combinations of the x–type variables will be different. Therefore, what distinguishes one relaxed dual problem problem from the others is the bound combination at which the linearization will be computed, as well as the qualifying constraints that have to be included. As can be seen in Figure 2, the y–domain is partitioned based on the signs of the qualifying constraints. In this simple illustration we assume that there exist 2 connected variables that give rise to four bound combinations, that is four possible sign combinations of the qualifying constraints. A particular node in our parallel architecture will be solely responsible for solving the primal problem and preparing all the necessary data for the formulation of the relaxed dual problems. Subsequently, each node, based on the number of connected variables that have been identified, determines whether it will be responsible for solving any relaxed dual problems. The next step will be, for every node, to proceed on the solution of the relaxed dual problems corresponding to the bound combinations that have been assigned to it. Once the assigned problems have been solved, all the feasible solutions are stored in the local CPU's and only the best lower bound generated at each processing element is being propagated to the "master" node. This issue brings us naturally to the third implementational issue associated with the distributed implementation of the GOP algorithm, that is the routing of the best lower bound.

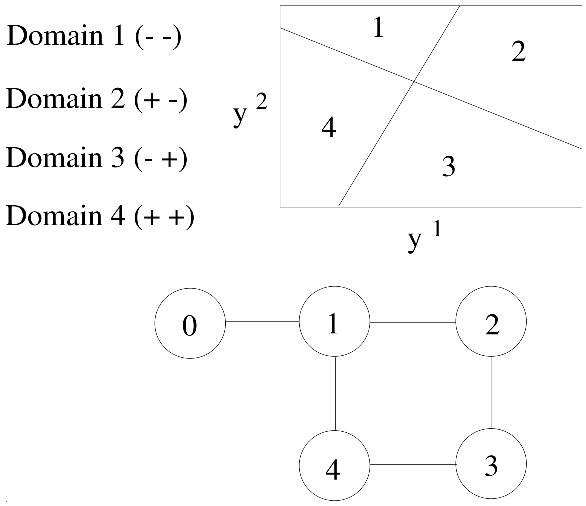

Figure 2. Parallel Solution of the Relaxed Dual Problems.
$- : \nabla_{x_i} L(\boldsymbol{x}^B, \boldsymbol{y}, \lambda^k, \mu^k) \leq 0\,,\ \ + : \nabla_{x_i} L(\boldsymbol{x}^B, \boldsymbol{y}, \lambda^k, \mu^k) \geq 0$

3.3. Routing of the Best Lower Bound

Poor data communication in parallel architectures can create substantial bottlenecks, thus degrading the overall performance of the algorithm. Based on the previous section it is clear that for the "master" node to proceed with the solution of the next primal problem only information related to the best lower bound is needed. Furthermore, it is rare to envision a situation in which hundreds of processing elements attempt to, almost simultaneously, access a particular node in order to provide certain data. The queuing problems that would arise will be very significant. Therefore, we implemented a routing algorithm which would, in $\lfloor log(P) + 1 \rfloor$ steps, where P is the number of nodes, transmit to node 0 the best lower bound. In pseudo-code form, the implemented algorithm is as follows :

```
if(mod(node, 2) > 0)
    send lower_bound to (node-1)
else{
    flag = 1
    mult = 4
    while(flag == 1){
         recv lower_bound
         compare/update lower_bound
         if(mod(node, mult) > 0){
              send lower_bounds to (node-mult/2)
              flag = 1
         }
         mult = 2 * mult
    }
 }
```

Figure 3, depicts a situation in which processing nodes $1, \ldots, 7$, through a series of "transimit" and "receive-compare" operations communicate the minimum of 7 numbers to node unit 0.

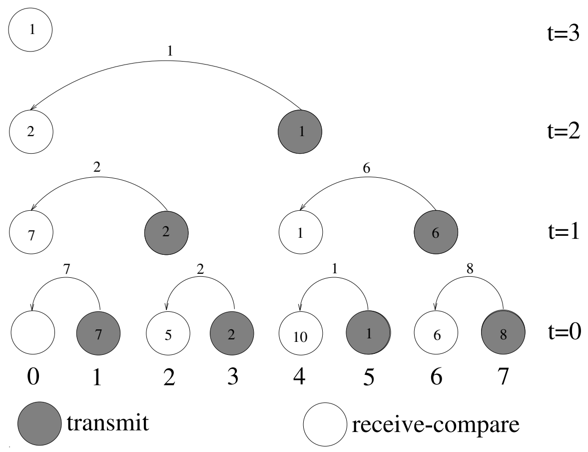

Figure 3. Routing of the Best Lower Bound

Each processing element will either receive a lower bound from another processing element and compare it with its current best, or it will transmit its current best lower bound to another processing element, whose index is defined by the particular algorithm we have implemented. The positive features of such an implementation are that the generated solutions are stored locally and only a very small number of data is transmitted through the network. The last message that will be received by node "0" will contain the current best lower bound.

4. Large Scale Indefinite Quadratic Problems

In this section, we will consider the application of the distributed version of the GOP algorithm to large scale Indefinite Quadratic Optimization problems. The generic formulation of [12] is considered. According to that formulation the optimization problem is stated as follows:

$$
\begin{aligned}
&\min_{x,y\in\Omega} \psi(x,y) = \sum_{i=1}^{n} \lambda_i (x_i - \overline{\omega_i})^2 + d^T y \\
&\Omega = \{(x,y) : A_1 x + A_2 y \leq b\ ,\ \ x \geq 0, y \geq 0\} \\
&x, \lambda, \overline{\omega} \in \Re^n \\
&y, d \in \Re^k \\
&A_1 \in \Re^{m\times n}, A_2 \in \Re^{m\times k}
\end{aligned}
$$

By construction, we generate half of the eigenvalues, λ_i, positive and half negative. Reportedly, this is the most difficult problem to address since the solution, unlike strictly concave problems, may not lie on a vertex point. Several runs, for different problem sizes, were performed and the results are analyzed with respect to (i) the effect of the linear constraints, and linear variables; (ii) the effect of linear constraints; (iii) the effect of linear variables. Finally, some very computationally intensive tasks are discussed. In all runs we denote by k the number of linear variables, m the number of linear constraints, and n the number of quadratic variables.

4.1. Combined Effect of the Linear Constraints and Linear Variables

The first set of computational results aims at demonstrating the effect on the performance of the GOP for different values of k and m. Typical results are tabulated in Table 1.

Based on these results the following observation can be made : *as long as* $\frac{k+m}{n} \geq 1$ *and* $\frac{m}{n} \geq 1$ *the GOP algorithm identifies the global minimum with maximum efficiency*.The GOP algorithm takes two iterations and solves only two primal problems and two relaxed dual problems. Qualitatively, this implies that, for this particular structure of problems such a combination of the parameters forces the solution to lie close to a vertex point. The GOP algorithm, identifies that fact very efficiently and converges in the minimum number of iterations.

Table 1. Combined Effect of m and k. N = m+n, M = m + 2(k+n).

k	m	n	N	M	CPU(s)
80	120	100	180	480	4.49
100	100	100	200	500	5.42
100	200	200	300	800	10.3
100	300	100	200	700	9.38
150	150	250	400	950	12.3
200	200	100	300	800	6.26
200	200	200	400	1000	7.86
200	200	220	420	1040	17.4
200	200	250	450	1100	4.50
200	300	100	300	900	5.08
300	300	100	400	1100	9.58
400	200	100	500	1200	13.2

Table 2. Effect of m. n = 100, k = 200.

m	NI_c	Itn	PE	CPU(s)
1	0	2	2	2.45
5	5	18	32	30.1
10	9	30	32	96.3
20	8	4	32	44.5
40	8	4	32	60.1
60	3	4	4	11.3
100	0	2	2	5.42
200	0	2	2	5.76
300	0	2	2	7.62

4.2. Effect of the Number of Linear Constraints

The next computational experiment aimed at isolating the effect of the linear constraints in the behavior of GOP. The computational results have been tabulated in Table 2 for constant values of $k = 200$, and $n = 100$.

As it can be seen, increasing the number of linear constraints results in an increase in the computational effort initially. This can be seen by observing both NI_c as well as the number of the required interactions. As m increases though the addition of linear constraints in the feasible region increases the number of vertices in the polytope and as a result the GOP becomes once again very efficient in determining the global minimum with minimal effort. The results are also depicted in Figure 4.

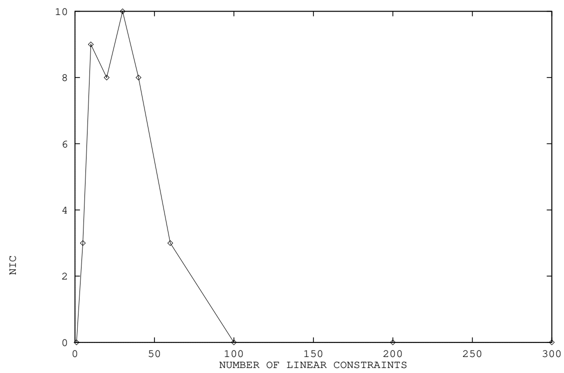

Figure 4. Effect of m

Table 3. Effect of k. n = 100, m = 100.

k	NI_c	Itn	PE	CPU(s)
0	1	2	2	3.17
20	1	2	2	3.80
40	3	2	2	6.30
60	3	2	4	11.3
80	4	2	8	28.1
90	1	2	4	7.45
100	1	2	2	5.42
130	5	2	32	32.2.
150	6	2	32	41.8
200	6	2	32	54.2
300	9	3	32	192.
400	10	2	32	198.

4.3. Effect of the Number of Linear Variables

The effect of increasing the number of linear variables in the behavior of the GOP is more complex. The computational results of Table 3 indicate the complex nature of the effect of the value of k.

It is clear from Figure 5 that k has a non monotonic effect and eventually, the larger the value of k the more difficult the problem becomes. It should be pointed out that the difficulty, in terms of GOP, is expressed through the number of connected variables, NI_c, and the number of iterations for the GOP to converge. The CPU is an indication as well, but for the large problems of Section 4.1 the large CPU time were due to the large size of the resulting convex sub-problems that had to be solved.

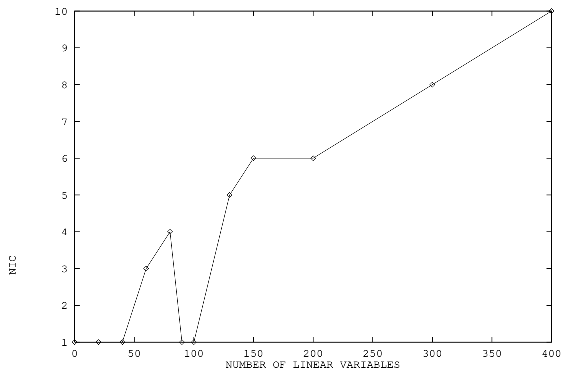

Figure 5. Effect of k

Table 4. Some computationally intensive tasks.

k	m	n	NI_c	Itn	PE	CPU(s)
30	20	100	9	54	32	100.
20	20	150	7	7	64	14.0
20	20	200	16	60	64	1847
300	100	200	8	3	32	54.0
50	50	220	10	3	64	31.6
50	50	250	14	16	64	796.
50	50	275	16	3	64	1800
50	50	300	17	3	64	3260
75	75	300	15	3	64	942.
75	75	350	11	3	64	209.

4.4. Effect of the Number of Nonlinear Variables

Finally, the last set of computational experiments deals with certain instances which are computationally very intensive in terms of the connected variables and number of iterations. The results have been summarized in Table 4.

It is important to notice from Table 4 the fact that although the absolute size of the problems might not be that large ($k = 30$, $m = 20$, and $n = 100$ for instance) the difficulty of the problem is noticeable both in terms of the number of iterations required as well as in terms of the number of connected variables. Note that for all the representative runs of Table 4 the relations between k, m, and n that define an "*easy*" problem for GOP are violated. By combining the theoretical advances of the GOP, along with the distributed implementation of the algorithm we were able to address problems of significant size.

As a last qualitative remark we will observe the computational requirements, in terms of the total CPU time, as a function of the the number of connected variables. As can be seen from Figure 6 there exists a linear relationship between the logarithm of the CPU time and the number of connected variables, NI_c as expected since the number of relaxed duals increases exponentially with NI_c.

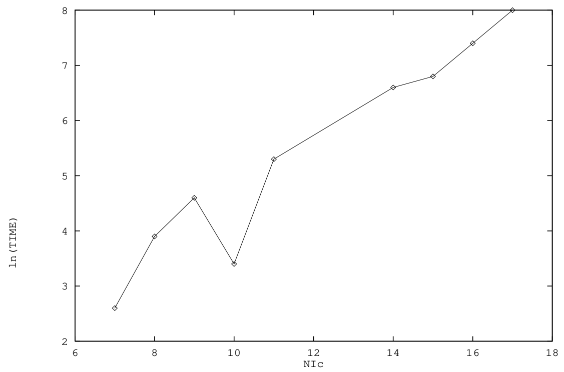

Figure 6. Time vs. NI_s

Summarizing, the computational results for indefinite quadratic problems we can observe that :

- problems of 400 linear variables, 100 nonlinear variables, 200 linear constraints, and 500 bound constraints can be solved in 13.2 s. as shown in Table 1.
- increasing the size of the linear linear constraints makes the problem easier for the GOP (e.g., problems with 300 linear constraints, 200 linear variables, 100 nonlinear variables require 7.6 s.)
- increasing the number of linear variables to 400 while maintaining 100 nonlinear variables and 100 constraints increases the CPU to 198 s., and
- problems that correspond to increasing the number of nonlinear variables up to 350 can still be solved with reasonable computational effort, as it is shown in Table 4.

5. Large Scale Blending and Pooling Problems

In this section we will discuss the solution of a specific formulation of pooling/blending problems using the GOP. Such problems are very often encountered in various chemical processes. They describe the situation in which a set of components ($ncomp$) with given level of certain qualities ($nqual$) is to be mixed in a given number of pools ($npool$) in order to produce a number of products ($nprod$) with prespecified characteristics. The situation is described in Figure 7.

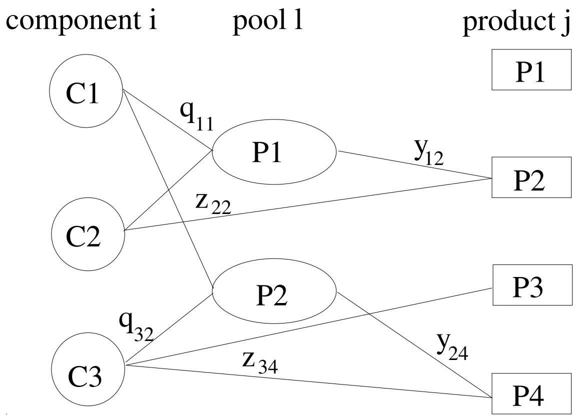

Figure 7. Pooling/Blending Problems

We use the reformulation proposed by [2] in which one considers the fractional flow rates q_{il}. We address a generalized version of the blending and pooling problem in which every component stream is allowed to directly reach any pool as well as any product. According to that assumption, the blending and pooling problem can be defined as follows:

$$\min_{q,x,z} - \sum_{j}^{nprod} \sum_{l}^{npool} (d_j - \sum_{i}^{ncomp} c_i q_{il}) y_{lj} - \sum_{j}^{nprod} \sum_{i}^{ncomp} (d_j - c_i) z_{ij}$$

s.t.

$$\sum_{l}^{npool} \sum_{j}^{nprod} q_{il} y_{lj} + \sum_{j}^{nprod} z_{ij} \leq A_i , \ \forall i$$

$$\sum_{j}^{nprod} y_{lj} \leq S_l , \ \forall l$$

$$\sum_{l}^{npool} y_{lj} + \sum_{i}^{ncomp} z_{ij} \leq D_j , \ \forall j$$

$$\sum_{l}^{npool} (\sum_{i}^{ncomp} C_{ik} q_{il} - P_{jk}) y_{lj} + \sum_{i}^{ncomp} (C_{ik} - P_{jk}) z_{ij} \leq 0 , \ \forall j, k$$

$$\sum_{i}^{ncomp} q_{il} = 1 , \ \forall l$$

$$q_{il} \geq 0\ ,\ \ y_{lj} \geq 0\ ,\ \ z_{ij}\ \geq\ 0$$

Clearly, this corresponds to a quadratically constrained problem with a quadratic objective function. Different instances of randomly generated Blending and Pooling Problems based on the above reformulation were generated and solved. Typical results are shown in Table 5.

Table 5. Blending and Pooling Problems

ncomp	nprod	npool	nqual	nvar	ncon	NI_c	PE	CPU(s)
5	5	3	5	55	186	15	64	15.3
10	4	4	9	96	300	15	64	36.3
10	4	4	16	96	356	15	64	42.0
10	4	4	18	96	372	16	64	39.0
10	4	4	25	96	428	16	64	44.9
10	4	5	30	110	468	20	64	843.
12	4	4	9	112	336	16	64	37.3
12	4	4	25	112	464	16	64	50.1
12	4	4	30	112	504	16	64	44.9
12	4	5	4	128	330	20	64	869.

The total number of variables for each instance is $nvar = ncomp \times nprod + npool \times nprod + ncomp \times nprod$ and the total number of constraints $ncon = 2 \times ncomp + nprod + 2 \times nprod + 2 \times npool \times nqual + 2 \times nprod$. The distributed implementation of the GOP allowed us to address problems with 20 connected variables, which require the solution of 1,048,576 relaxed dual problems, in very realistic computational times. For instance, as can be seen from Table 5 pooling problems of 128 variables and 330 constraints can be solved within 840–870 s., even though they have 20 connected variables.

6. Conclusions

In this paper, the distributed implementation of a decomposition–based global optimization algorithm, GOP, was presented. The main computational bottlenecks of the method were identified and these are comprised of (i) the distributed solution of a set of convex problems providing tight variable bounds, (ii) the distributed solution of the relaxed dual problems that provide valid lower bounds on the global minimum, and (iii) communication issues related to the routing of the necessary data. An efficient parallelization on the **Intel Paragon** machine was developed and results for large scale problems were presented. These include large scale Indefinite Quadratic Problems with up to 350 quadratic variables, as well as Blending–Pooling Problems with up to 12 components and 30 qualities.

Acknowledgments

Financial support from the National Science Foundation under Grants CBT–8857013, CTS–9221411, the Air Force Office of Scientific Research, as well as Exxon Co., Amoco Chemicals Co., Mobil Co., and Tennessee Eastman Co. is gratefully acknowledged. The authors wish to acknowledge the University of San Diego Super Computing Center for providing access to the Intel Paragon parallel machine.

References

1. F. A. Al-Khayyal. Jointly Constrained Bilinear Programs and Related Problems: An Overview. *Computers in Mathematical Applications*, 19:53–62, 1990.
2. A. Ben-Tal, G. Eiger, and V. Gershovitz. Global Minimization by Reducing the Duality Gap. *Mathematical Programming*, 63:193–212, 1994.
3. C. A. Floudas and P. M Pardalos. *Recent Advances in Global Optimization*. Princeton Series in Computer Science. Princeton University Press, Princeton, New Jersey, 1992.
4. R. Horst and P.M. Pardalos. *Handbook of Global Optimization: Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 1994.
5. C. A. Floudas and V. Visweswaran. A Primal-Relaxed Dual Global Optimization Approach. *J. Opt. Th. Appl.*, 78(2):187–225, 1993.
6. C. A. Floudas and V. Visweswaran. A Global Optimization Algorithm (GOP) for Certain Classes of Nonconvex NLPs: I. Theory. *Comp. chem. Eng.*, 14:1397–1417, 1990.
7. Tuy H., Thieu. T.V., and N.Q. Thai. A Conical Algorithm for Globally Minimizing a Concave Function Over a Closed Convex Set. *Math. of Oper. Res.*, 10(3):498–514, 1985.
8. E. R. Hansen. Global Optimization using Interval Analysis: The One-Dimensional Case. *J. Opt. Th. Appl.*, 29:331, 1979.
9. P. Hansen, B. Jaumard, and S-H. Lu. Global Optimization of Univariate Lipschitz Functions: I. Survey and Properties. *Mathematical Programming*, 55(3):251–272, 1992.
10. R. Horst, N. V. Thoai, and J. De Vries. A New Simplicial Cover Technique in Constrained Global Optimization. *J. Global. Opt.*, 2:1–19, 1992.
11. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671, 1983.
12. P.M. Pardalos and J.B. Rosen. Global optimization approach to the linear complementarity problem. *SIAM J. Sci. Stat. Computing*, 9(2):341–353, 1988.
13. A. H. G. Rinnooy-Kan and G. T. Timmer. Stochastic Global Optimization Methods. Part I: Clustering Methods. *Mathematical Programming*, 39:27, 1987.
14. H. Sherali and C. H. Tuncbilek. A Global Optimization Algorithm for Polynomial Programming Problems Using a Reformulation-Linearization Technique. *J. Global. Opt.*, 2:101–112, 1992.
15. H. Tuy. A general deterministic approach to global optimization via d.c. programming. In J. Hiriart-Urruty, editor, *FERMAT Days 1985: Mathematics for Optimization*, pages 273–303. Elsevier Sci. Publishers, 1985.
16. V. Visweswaran and C. A. Floudas. New Properties and Computational Improvement of the GOP Algorithm For Problems With Quadratic Objective Function and Constraints. *J. Global. Opt.*, 3(3):439–462, 1993.

17. V. Visweswaran and C.A. Floudas. A Global Optimization Algorithm (GOP) for Certain Classes of Nonconvex NLPs: II. Application of Theory and Test Problems. *Comp. &Chem. Eng.*, 14:1419–1434, 1990.

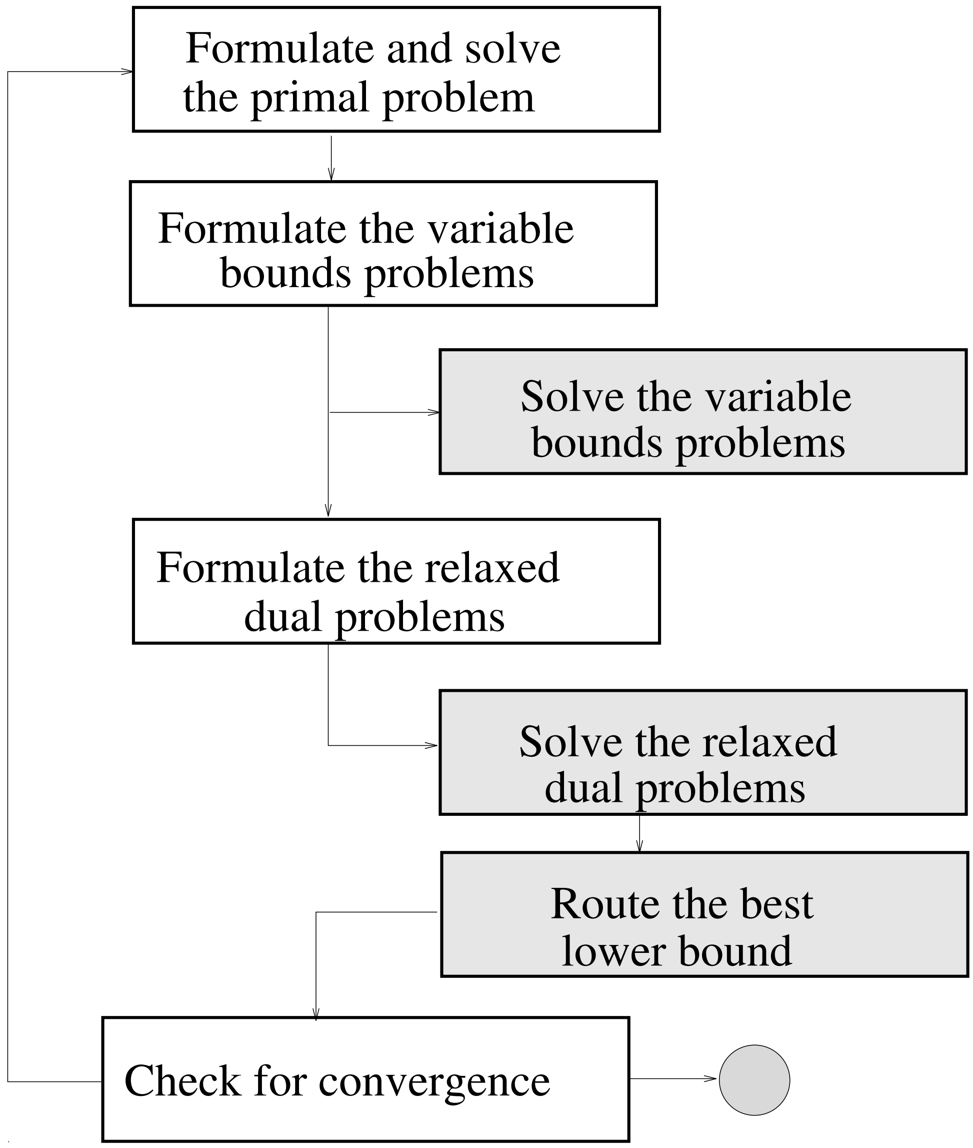

Figure 8. Flow Diagram