4

COMPUTATIONAL RESULTS FOR AN EFFICIENT IMPLEMENTATION OF THE GOP ALGORITHM AND ITS VARIANTS

V. Visweswaran* and C. A. Floudas**

** Mobil Research and Development Corporation, Princeton, NJ*
*** Department of Chemical Engineering, Princeton University, Princeton, NJ*

ABSTRACT

Recently, Floudas and Visweswaran (1990, 1993) proposed a global optimization algorithm (GOP) for the solution of a large class of nonconvex problems through a series of primal and relaxed dual subproblems that provide upper and lower bounds on the global solution. Visweswaran and Floudas (1995a) proposed a reformulation of the algorithm in the framework of a branch and bound approach that allows for an easier implementation. They also proposed an implicit enumeration of all the nodes in the resulting branch and bound tree using a mixed integer linear (MILP) formulation, and a linear branching scheme that reduces the number of subproblems from exponential to linear. In this paper, a complete implementation of the new versions of the GOP algorithm, as well as detailed computational results of applying the algorithm to various classes of nonconvex optimization problems is presented. The problems considered including pooling and blending problems, problems with separation and heat exchanger networks, robust stability analysis with real parameter uncertainty, and concave and indefinite quadratic problems of medium size.

1 INTRODUCTION

Floudas and Visweswaran (1990, 1993) proposed a global optimization algorithm (GOP) for the solution of a large class of nonconvex problems. The algorithm solves the original problem iteratively through a series of primal and relaxed dual subproblems, which provide upper and lower bounds on the global solution. The algorithm has a guarantee of finite convergence to an ϵ-optimal solution; however, the nature of its cutting plane approach renders the implementation very difficult, especially in the steps leading to the choice of underestimators to be used during various iterations. To circumvent this problem, Visweswaran and Floudas (1995a) proposed the reformulation of the algorithm in the framework of a branch and bound approach. At each iteration, the gradients of the Lagrange function are used for branching, with the primal and relaxed dual problems at each node are used to provide upper and lower bounds on the global solution. The paper also addressed the question of implicit enumerations of all the nodes in the tree by using a mixed integer linear (MILP) formulation for the relaxed dual problem, and proposed a new branching scheme that only requires a linear number of relaxed dual subproblems at each iteration.

In this paper, a complete implementation of the new versions of the GOP algorithm, along with computational results, is discussed. The actual details of the implementation can be found in Appendix A, which discusses the various aspects involved in the implementation, including reduction tests and local enhancements at each node of the tree. In particular, the movement of data from one part of the program to another is discussed in detail. In the following sections, the results of applying the implementation to various classes of nonconvex optimization problems, including pooling and blending problems, problems with separation and heat exchanger networks, and quadratic problems from literature are described.

2 COMPUTATIONAL RESULTS

A complete description of the GOP and GOP/MILP algorithms can be found in Visweswaran and Floudas (1995a). These algorithms have been implemented in a complete package **cGOP** (Visweswaran and Floudas, 1995b). The details of the implementation can be found in Appendix A. In this section, we present the results of the application of the **cGOP** package to various problems in chemical engineering design and control and mathematical programming.

2.1 Heat Exchanger Network Problems

Heat exchanger network synthesis problems have traditionally been solved using a decomposition strategy, where the aims of targeting, selection of matches and optimization of the resulting network configuration are treated as independent problems. Given the minimum utility requirements and a set of matches, a superstructure of all the possible alternatives is formulated. The resulting optimization problem is nonconvex. In this section, two such superstructures of heat exchanger networks are solved using the GOP algorithm.

The problems solved in this section have the following form:

$$\text{OBJ} \quad = \quad \min \sum_{ij \in MA} \alpha_{ij} \left(\frac{\mathbf{Q_{ij}}}{\mathbf{U_{ij}} LMTD_{ij}}\right)^{\beta_{ij}}$$

s.t.

(Initial splitter mass balance)

$$\sum_{k' \in R_k} f^I_{k'} \;=\; \mathbf{F^k}$$

(Mixer balances at exchanger inlets)

$$f^I_{k'} + \sum_{k'' \in S_{k'}} f^B_{k',k''} - f^E_{k'} \;=\; 0; \quad \forall\, k \in HCT.$$

(Splitter balances at exchanger outlets)

$$f^O_{k'} + \sum_{k'' \in S_{k'}} f^B_{k'',k'} - f^E_{k'} \;=\; 0; \quad \forall\, k \in HCT.$$

(Energy balances at mixers)

$$\mathbf{T^k} f^I_{k'} + \sum_{k'' \in S_{k'}} f^B_{k',k''} t^O_{k''} - f^E_{k'} t^I_{k'} \;=\; 0; \quad \forall\, k \in HCT.$$

(Energy balances in exchangers)

$$\mathbf{Q_{ij}} \;=\; f^{E,j}_i (t^{I,j}_i - t^{O,j}_i) \quad \forall\, (ij) \in MA$$

$$\mathbf{Q_{ij}} \;=\; f^{E,i}_j (t^{O,i}_j - t^{O,i}_j) \quad \forall\, (ij) \in MA$$

$$LMTD_{ij} \;=\; \frac{2}{3} \times (DT1_{ij} \times DT2_{ij})^{1/2} \;+\; \frac{1}{6} \times (DT1_{ij} + DT2_{ij})$$

Here, U_{ij} are the fixed heat transfer coefficients. It should be noted that for fixed Q_{ij}, the objective function is convex. Therefore, by *projecting* on the flow rates f_i, the primal problem becomes convex in the remaining variables (the temperatures and temperature differences). Linearization of the Lagrange function ensures that the relaxed dual subproblems are LP subproblems in the flowrates.

Example 2.1 This example is taken from Floudas and Ciric (1989). In this problem, the objective is to determine the globally optimal network for a system of two hot streams and one cold stream. The superstructure of all possible solutions is shown in Figure 1. Based upon this superstructure, the model can be formulated as the following optimization problem :

$$\begin{array}{ll} \min & 1300\left[\dfrac{1000}{0.05\left[\frac{2}{3}(\Delta T_{11}\Delta T_{12})\right]+\frac{1}{6}(\Delta T_{11}+\Delta T_{12})}\right]^{0.6} \quad + \\ & 1300\left[\dfrac{600}{0.05\left[\frac{2}{3}(\Delta T_{21}\Delta T_{22})\right]+\frac{1}{6}(\Delta T_{21}+\Delta T_{22})}\right]^{0.6} \end{array}$$

$s.t.$

$$\begin{aligned} f_1^I + f_2^I &= 10 \\ f_1^I + f_{12}^B - f_1^E &= 0 \\ f_2^I + f_{21}^B - f_2^E &= 0 \\ f_1^O + f_{21}^B - f_1^E &= 0 \\ f_2^O + f_{12}^B - f_2^E &= 0 \\ 150 f_1^I + t_2^O f_{12}^B - t_1^I f_1^E &= 0 \\ 150 f_2^I + t_1^O f_{21}^B - t_2^I f_2^E &= 0 \\ f_1^E(t_1^O - t_1^I) &= 1000 \\ f_2^E(t_2^O - t_2^I) &= 600 \\ \Delta T_{11} = 500 - t_1^O, \quad \Delta T_{12} &= 250 - t_1^I \\ \Delta T_{21} = 350 - t_2^O, \quad \Delta T_{22} &= 200 - t_2^I \\ \Delta T_{11}, \Delta T_{12}, \Delta T_{21}, \Delta T_{22} &\geq 10 \end{aligned}$$

Considering the set of possible solutions inherent in Figure 1, it is obvious that the bypass streams (f_{12}^B and f_{21}^B) can never be simultaneously active, i.e. at least one of these streams has to be zero. Therefore, two different problems can be solved, one with $f_{12}^B = 0$ and another with $f_{21}^B = 0$. When the GOP algorithm is applied to the

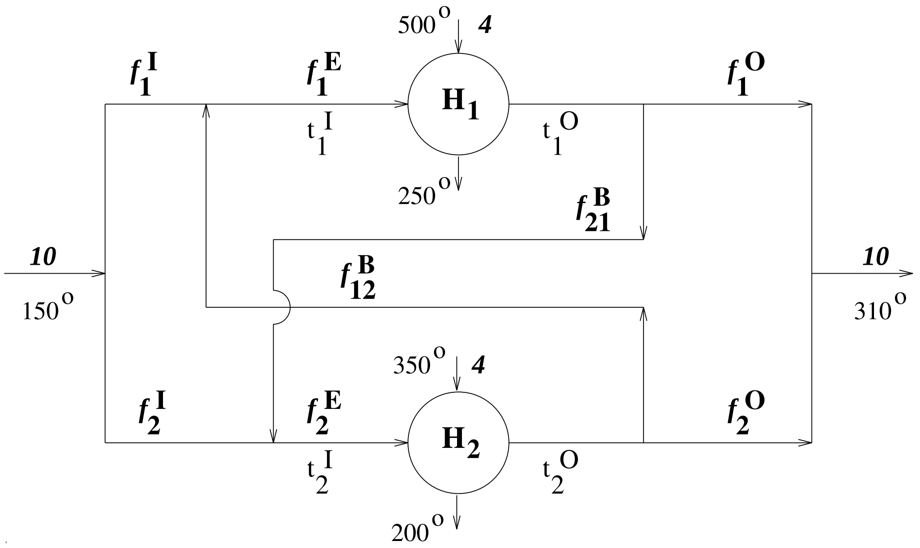

Figure 1 Heat Exchanger Network Superstructure For Example 2.1

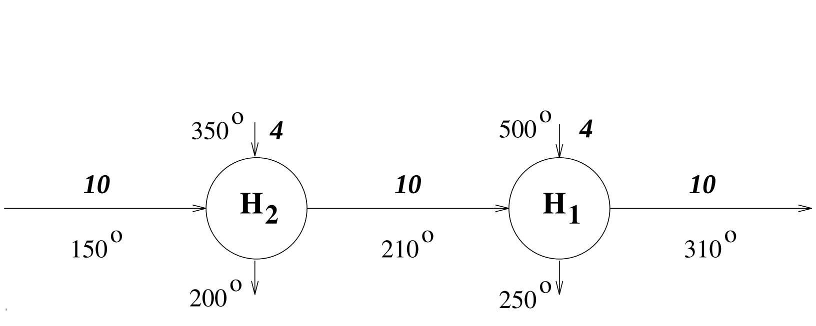

Figure 2 Optimal Configuration For Example 2.1

problem in this form, the optimal solution (given in Figure 2) is found in 11 iterations, needing 0.54 cpu seconds on an HP 730.

Example 2.2 This example is also taken from Floudas and Ciric (1989). It features three hot streams and two cold streams.

$$
\begin{aligned}
\min \quad & 1300\left[\frac{1000}{0.5\left[\frac{2}{3}(\Delta T_{11}\Delta T_{12})\right] + \frac{1}{6}(\Delta T_{11} + \Delta T_{12})}\right]^{0.6} + \\
& 1300\left[\frac{600}{1.0\left[\frac{2}{3}(\Delta T_{21}\Delta T_{22})\right] + \frac{1}{6}(\Delta T_{21} + \Delta T_{22})}\right]^{0.6} + \\
& 1300\left[\frac{600}{2.0\left[\frac{2}{3}(\Delta T_{31}\Delta T_{32})\right] + \frac{1}{6}(\Delta T_{31} + \Delta T_{32})}\right]^{0.6}
\end{aligned}
$$

$$
\begin{aligned}
\text{s.t.} \quad f_1^I + f_2^I + f_3^I &= 45 \\
f_1^I + f_{12}^B + f_{13}^B - f_1^E &= 0 \\
f_2^I + f_{21}^B + f_{23}^B - f_2^E &= 0 \\
f_3^I + f_{31}^B + f_{32}^B - f_3^E &= 0 \\
f_1^O + f_{21}^B + f_{31}^B - f_1^E &= 0 \\
f_2^O + f_{12}^B + f_{33}^B - f_2^E &= 0 \\
f_3^O + f_{13}^B + f_{23}^B - f_3^E &= 0 \\
100 f_1^I + t_2^O f_{12}^B + t_3^O f_{13}^B - t_1^I f_1^E &= 0 \\
100 f_2^I + t_1^O f_{21}^B + t_3^O f_{23}^B - t_2^I f_2^E &= 0 \\
100 f_3^I + t_3^O f_{31}^B + t_2^O f_{32}^B - t_3^I f_3^E &= 0 \\
f_1^E(t_1^O - t_1^I) = 2000, \quad f_2^E(t_2^O - t_2^I) = 1000, \quad f_3^E(t_3^O - t_3^I) &= 1500 \\
\Delta T_{11} = 210 - t_1^O, \quad \Delta T_{21} = 210 - t_2^O, \quad \Delta T_{31} &= 210 - t_3^O \\
\Delta T_{12} = 130 - t_1^I, \quad \Delta T_{22} = 160 - t_2^I, \quad \Delta T_{32} &= 180 - t_3^I \\
\Delta T_{11}, \Delta T_{12}, \Delta T_{21}, \Delta T_{22} \Delta T_{31}, \Delta T_{32} &\geq 10 \\
0 \leq f_1^I, f_2^I, f_3^I, f_1^O, f_2^O, f_3^O, &
\end{aligned}
$$

The superstructure for this example is shown in Figure 3. There are a total of 27 variables and 19 constraints (of which six are bilinear). With a *projection* on the flow rates, there are six connected variables. The GOP algorithm requires a total of 39 iterations and 54.62 cpu seconds to solve this problem. The optimal solution found by the algorithm is given in Figure 4.

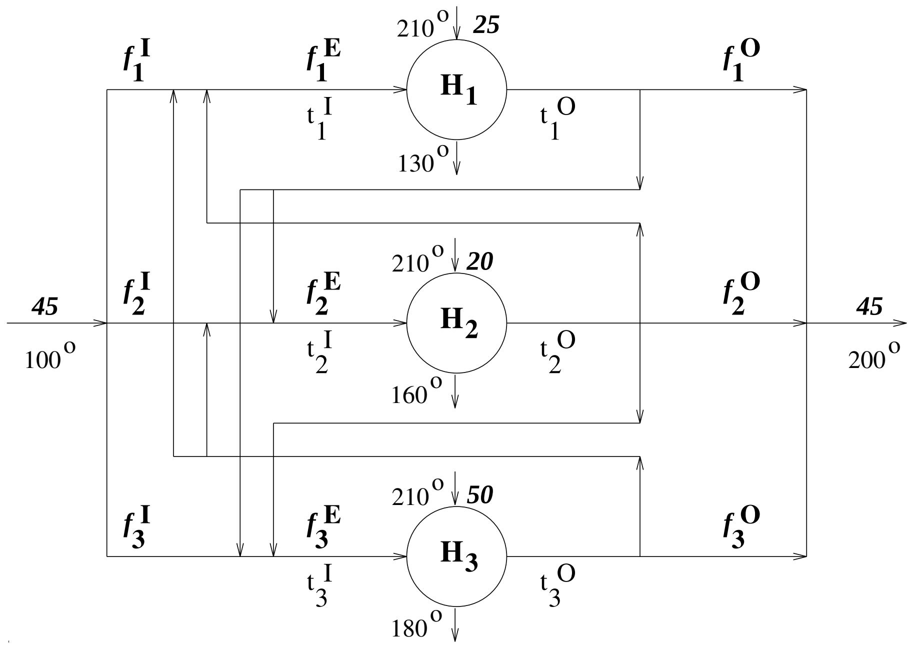

Figure 3 Heat Exchanger Network Superstructure For Example 2.2

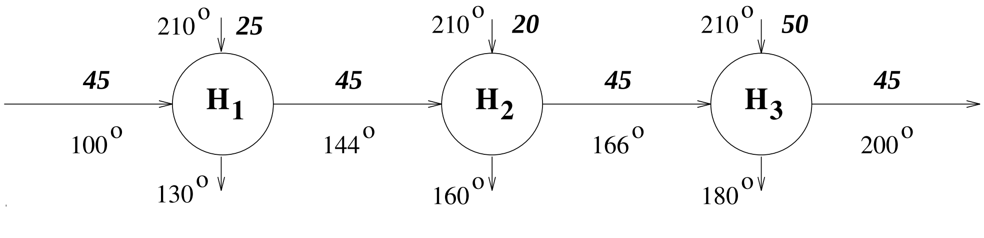

Figure 4 Optimal Configuration For Example 2.2

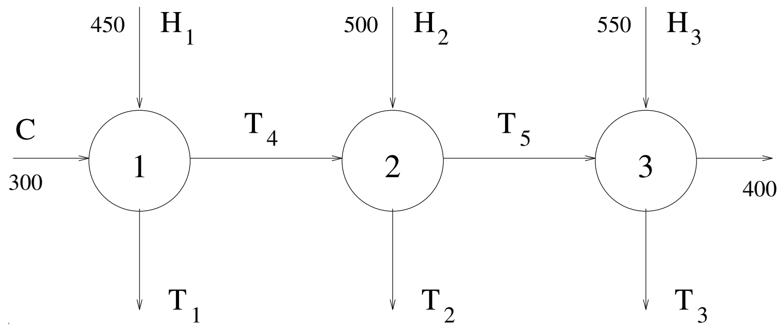

Figure 5 Heat Exchanger Example From Quesada and Grossmann (1993)

2.2 Heat Exchanger Problems With Linear Cost Functionals

In this section, we apply the GOP algorithm the global optimization of several heat exchanger networks with fixed topology. The problems are taken from Quesada and Grossmann (1993) and assume linear cost functionals for the exchanger areas as well as arithmetic mean driving forces for the temperature differences between the exchanging streams. Under these assumptions, the problems reduce to the minimization of a sum of linear fractional functions (which is nonconvex) over a set of linear constraints.

In order to reduce these problems to a form where the GOP algorithm could be applied, we employ the ideas of Liu and Floudas (1993), which involve a difference of convex functions transformation. This involves use of eigenvalue analysis on the resulting fractional objective functions in order to determine the smallest quadratic terms that are needed to ''convexify'' the objective function. Since this method is very general and can be of use in various problems of this type, it is outlined in some detail here for one of the examples.

This example (Example 4 of Quesada and Grossmann, 1993) features a network of three exchangers used to heat one cold stream and cool three hot streams. This network is shown in Figure 5, with $FC_p = 10$ for all the streams. The minimum temperature of approach is $10^0 K$.

The problem formulation, featuring constraints for the heat balances, minimum temperature approaches and feasibility is shown below:

$$\min \quad \frac{Q_1}{2\Delta T_1} + \frac{Q_2}{2\Delta T_2} + \frac{Q_3}{2\Delta T_3}$$

Temperature Differences:

$$\begin{aligned} 2\Delta T_1 &= 150 + T_1 - T_4 \\ 2\Delta T_2 &= 500 + T_2 - T_4 - T_5 \\ 2\Delta T_3 &= 150 + T_3 - T_5 \end{aligned}$$

Heat Balances:

$$\begin{aligned} Q_1 &= 10(T_4 - 300) = 10(450 - T_1) \\ Q_2 &= 10(T_5 - T_4) = 10(500 - T_2) \\ Q_3 &= 10(400 - T_5) = 10(550 - T_3) \end{aligned}$$

Minimum Temperature Approaches:

$$\begin{aligned} T_1 - 300 &\geq 10 \qquad 450 - T_4 \geq 10 \\ T_2 - T_4 &\geq 10 \qquad 500 - T_5 \geq 10 \\ T_3 - T_5 &\geq 10 \end{aligned}$$

Feasibility:

$$T_1, T_2, T_3, T_4, T_5 \geq 0$$

The three heat balance equations can be used to eliminate three of the variables in the problem. Choosing the intermediate streams T_4 and T_5 as the independent variables leads to

$$\begin{aligned} T_1 &= 750 - T_4 \\ T_2 &= 500 + T_4 - T_5 \\ T_3 &= 150 + T_5 \end{aligned}$$

Using the minimum temperature approaches, tighter bounds on T_4 and T_5 are obtained:

$$\begin{array}{llllllllll} T_1 & \geq & 310 & \Rightarrow & 750 - T_4 & \geq & 310 & \Rightarrow & T_4 & \leq & 440 \\ T_2 & \geq & 10 + T_4 & \Rightarrow & 500 + T_4 - T_5 & \geq & 10 + T_4 & \Rightarrow & T_5 & \leq & 490 \end{array}$$

Similarly the temperature differences reduce to

$$\Delta T_1 = 450 - T_4$$

$$\Delta T_2 = 500 - T_5$$
$$\Delta T_3 = 150$$

Thus, the problem formulation reduces to

$$\min \quad 10000\left[\frac{T_4 - 300}{450 - T_4} + \frac{T_5 - T_4}{500 - T_5} + \frac{400 - T_5}{150}\right]$$

$$300 \leq T_4, T_5 \leq 400$$

Consider now the three individual terms inside the parentheses. For the sake of clarity, the factor of 10000 is omitted below.

- The first fractional term is
$$F_1 = \frac{T_4 - 300}{450 - T_4}.$$
The Hessian of this function is given by
$$\frac{\delta^2 F_1}{\delta T_4^2} = \frac{300}{(450 - T_4)^3}$$
which is always positive, since $T_4 \leq 400$. Therefore, this term is convex for all values of T_4 and T_5.

- The third term
$$F_3 = \frac{400 - T_5}{150},$$
is a linear term and therefore always convex.

- The second term is
$$F_2 = \frac{T_5 - T_4}{500 - T_5}.$$
The Hessian of F_2 is given by
$$H_2 = \begin{bmatrix} 0 & \frac{-1}{y^2} \\ \frac{-1}{y^2} & \frac{2x}{y^3} \end{bmatrix}$$
where $x = 500 - T_4$ and $y = 500 - T_5$. The eigenvalues of this Hessian are given by
$$E_1, E_2 = \frac{x \pm \sqrt{x^2 + y^2}}{y^3}$$

It can be seen that the second eigenvalue (for the negative value of the square root) will *always* be negative. Thus, the Hessian has mixed eigenvalues, indicating that the second term in the objective is nonconvex.

In order to ''convexify'' this term, a quadratic term in one or more of the variables can be added. Suppose that the term αT_4^2 is added. Then, the term becomes

$$F_2' = \frac{T_5 - T_4}{500 - T_5} + \alpha T_4^2$$

The Hessian of this term is given by

$$H_2' = \begin{bmatrix} 2\alpha & \frac{-1}{y^2} \\ \frac{-1}{y^2} & \frac{2x}{y^3} \end{bmatrix}$$

where again $x = 500 - T_4$ and $y = 500 - T_5$. The eigenvalues of this Hessian are given by

$$E_1, E_2 = \frac{1}{y^3}\left[x + \alpha y^3 \pm \sqrt{(x - \alpha y^3)^2 + y^2}\right]$$

For the second eigenvalue to be positive for all values of T_4 and T_5, the term in the square brackets must be positive. In other words,

$$x + \alpha y^3 \pm \sqrt{(x - \alpha y^3)^2 + y^2} \geq 0$$

This leads to the inequality

$$\alpha \geq \frac{1}{4xy}$$

Since $100 \leq x, y \leq 200$, we obtain

$$\alpha \geq \frac{1}{40000}$$

Thus, adding the term $\frac{1}{40000}T_4^2$ to F_2 is sufficient to make this term convex. The net result of this is that the objective function can now be written as

$$\min \quad 10000\left[\frac{T_4 - 300}{450 - T_4} + \frac{T_5 - T_4}{500 - T_5} + \frac{400 - T_5}{150} + \frac{T_4^2}{40000}\right] - \frac{T_4^2}{4}$$

where the first term is convex, and the second term is concave. By the addition of an extra variable and renaming all the variables, the problem now becomes

$$\min \quad 10000\left[\frac{y_1 - 300}{450 - y_1} + \frac{y_2 - y_1}{500 - y_2} + \frac{400 - y_2}{150} + \frac{y_1^2}{40000}\right] - 0.25x_1y_1$$

Problem Name	*Problem Size*		*GOP Algorithm*	
	Variables	Constraints	Iterations	CPU (sec)
Example 1	12	13	4	0.09
Example 2	12	13	3	0.06
Example 4	11	9	3	0.10
Example 5	11	9	8	0.20
Example 7	26	30	4	0.11

Table 1 Heat Exchanger Network Problems from Quesada and Grossmann (1993) with variables eliminated as detailed in Section 2.2

$$\begin{aligned} x_1 - y_1 &= 0 \\ 300 \leq x_1, y_1, y_2 &\leq 400 \end{aligned}$$

Now the problem satisfies the conditions of the GOP algorithm, being a convex problem in y for all fixed x and a linear problem in x for all fixed y.

Similar reductions were obtained for all the example problems given in Quesada and Grossmann (1993). The results of applying the GOP algorithm to these problems is given in Table 1. Note that in all the cases, the problems reduced to either one or two variable unconstrained problems. Consequently, the subproblems solved by the algorithm are very small in size, as shown in the CPU times taken to converge to the optimum.

2.3 Pooling and Blending Problems

Pooling and blending problems are a feature of models for most chemical processes. In particular, for problems relating to refinery and petrochemical processing, it is often necessary to model not only the product flows but the properties of intermediate streams as well. These streams are usually combined in a tank or pool, and the pool is used in downstream processing or blending. The presence of these streams in the model introduces nonlinearities, often in a nonconvex manner. The nonconvexities arise from the interactions between the qualities of the input streams and the blended products.

Traditionally, pooling problems have been solved using successive linear programming (SLP) techniques. The first SLP algorithm (Method of Approximation Programming) was proposed by Griffith and Stewart (1961). Subsequently, SLP algorithms have

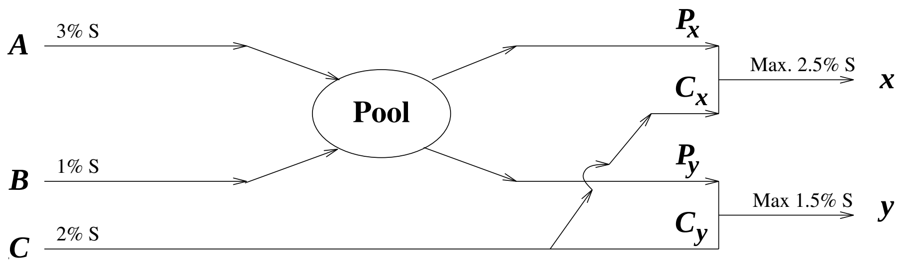

Figure 6 The Haverly Pooling Problem

been proposed by Lasdon *et al.* (1979), Palacios-Gomez *et al.* (1982) and Baker and Lasdon (1985) among others. These algorithms have been applied to pooling problems by Haverly (1978) and Lasdon *et al.* (1979). SLP algorithms have the advantage that they can utilize existing LP codes and can handle large scale systems easily. However, to guarantee convergence to the global solution, they require convexity in the objective function and the constraints. For this reason, these methods cannot be relied upon to determine the best solution for all pooling problems.

Various formulations have been proposed for pooling and blending problems. In the following sections, we consider the application of the GOP algorithm to three of these formulations, namely, the Haverly Pooling problem, two pooling problems from Ben-Tal and Gershovitz (1992), and a multiperiod tankage quality problem commonly occuring in refineries.

The Haverly Pooling Problem

In his studies of the recursive behavior of linear programming **(LP)** models, Haverly (1978) defined a pooling problem as shown in Figure 6. Three substances A, B and C with different sulfur contents are to be combined to form two products x and y with specified maximum sulfur contents. In the absence of a pooling restriction, the problem can be formulated and solved as an LP. However, when the streams need to be pooled (as, for example, when there is only one tank to store A and B), the LPmust be modified. Haverly has shown that without the explicit incorporation of the effect of the economics associated with the sulfur constraints on the feed selection process, a recursive algorithm for solving a simple formulation having only a pool balance cannot find the global solution. Lasdon *et al.* (1979) added a pool quality constraint to the formulation. This complete NLP formulation is shown below :

$$\min \quad 6A + 16B + 10(C_x + C_y) - \quad 9x - 15y$$

$s.t.$

$$P_x + P_y - A - B = 0 \quad \} \qquad \textit{pool balance}$$

$$\left.\begin{array}{rcl} x - P_x - C_x & = & 0 \\ y - P_y - C_y & = & 0 \end{array}\right\} \qquad \textit{component balance}$$

$$p.(P_x + P_y) - 3A - B = 0 \quad \} \qquad \textit{pool quality}$$

$$\left.\begin{array}{rcl} p.P_x + 2.C_x - 2.5x & \leq & 0 \\ p.P_y + 2.C_y - 1.5y & \leq & 0 \end{array}\right\} \qquad \textit{product quality constraints}$$

$$\left.\begin{array}{rcl} x & \leq & x^U \\ y & \leq & y^U \end{array}\right\} \qquad \textit{upper bounds on products}$$

where p is the sulfur quality of the pool; its lower and upper bounds are 1 and 3 respectively. This problem was solved by both Haverly (1979) and Lasdon *et al.* (1979). In all cases, however, the global optimum could not always be determined, the final solution being dependent on the starting point.

More recently, Floudas and Aggarwal (1990) solved the problem using the Global Optimum Search (Floudas *et al.*, 1989). They had to reformulate the problem by adding variables and constraints, and despite being they were successful in finding the global minimum from 28 out of 30 starting points, they could not mathematically guarantee that the algorithm would converge to the global minimum.

The GOP Algorithm

By *projecting* on the pooling quality p, the problem becomes linear in the remaining variables. Hence, p is chosen as the ''y'' variable. From the constraint set, it can be seen that only P_x and P_y are the *connected* variables. Hence, four relaxed dual subproblems need to be solved at each iteration. Three cases of the pooling problem have been solved using the GOP and GOP/MILP algorithms. The data for these three cases, as well as the average number of iterations required by the algorithms to converge, are given in Table 2. It can be seen that in all cases, the algorithms require less than 15 iterations to identify and converge to the global solution.

Case	Bounds		Cost	Optimal Solution		GOP		GOP/MILP	
	x^U	y^U	of B	$f^\star$	$p^\star$	Iter.	CPU	Iter.	CPU
I	100	200	\$16	-\$400	1.0	12	0.22	12	0.49
II	600	200	\$16	-\$600	3.0	12	0.21	12	0.45
III	100	200	\$13	-\$750	1.5	14	0.26	14	0.56

Table 2 Data and results for the Haverly Pooling Problem

Pooling Problems From Literature

We have also applied the GOP algorithm to two pooling problems taken from Ben-Tal and Gershovitz (1992). The following notation is used for these problem models :

$$\begin{aligned} \{1, 2, \cdots, i, \cdots, I\} &\equiv \text{set of components} \\ \{1, 2, \cdots, j, \cdots, J\} &\equiv \text{set of products} \\ \{1, 2, \cdots, k, \cdots, K\} &\equiv \text{set of qualities} \\ \{1, 2, \cdots, l, \cdots, L\} &\equiv \text{set of pools} \end{aligned}$$

The following variable sets are present in the model :

$$\begin{aligned} x_{il} &\quad - \quad \text{amount of component } i \text{ allocated to pool } l \\ y_{lj} &\quad - \quad \text{amount going from pool } l \text{ to product } j \\ z_{ij} &\quad - \quad \text{amount of component } i \text{ going to product } j \\ p_{lk} &\quad - \quad \text{level of quality } k \text{ in pool } l \end{aligned}$$

The parameters in the problem are :

$$\begin{aligned} A_i &\quad - \quad \text{Upper bounds for component availabilities} \\ D_j &\quad - \quad \text{Upper bounds for product demands} \\ S_l &\quad - \quad \text{Upper bounds for pool sizes} \\ Q_{jk} &\quad - \quad \text{Upper bounds for product qualities} \\ q_{ik} &\quad - \quad \text{Level of quality } k \text{ in component } i \\ c_i &\quad - \quad \text{Unit price of component } i \\ d_j &\quad - \quad \text{Unit price of product } j \end{aligned}$$

Using this notation, these pooling problems have the following form:

$$\max \quad -\sum_i \sum_l c_i x_{il} + \sum_l \sum_j d_j y_{lj} + \sum_i \sum_j (d_j - c_i) z_{ij}$$

Problem No.	*Problem Size*				*GOP Algorithm*	
	I	J	K	L	Iterations	CPU (HP730)
1.	4	2	1	1	7	0.95
2.	5	5	2	1	41	5.80

Table 3 Pooling Problems From Ben-Tal and Gershovitz (1992).

$$
\begin{aligned}
s.t. \quad \sum_l x_{il} + \sum_j z_{ij} &\le A_i \\
\sum_i x_{il} + \sum_j y_{lj} &= 0 \\
\sum_i x_{il} &\le S_l \\
-\sum_i q_{ik} x_{il} + p_{lk} \sum_j y_{lj} &= 0 \\
\sum_l y_{lj} + \sum_i z_{ij} &\le D_j \\
\sum_l (p_{lk} - Q_{jk}) y_{lj} + \sum_i (q_{ik} - Q_{jk}) z_{ij} &\le 0
\end{aligned}
$$

The data for these problems can be found in Ben-Tal and Gershovitz (1992). The results of application of the GOP algorithm to these problems is given in Table 3.

Multiperiod Tankage Quality Problem

This example concerns a multiperiod tankage quality problem that arises often in the operations of refineries. The models for these problems are similar to the pooling problem of the previous section.

In order to develop the mathematical formulation, the following sets are defined :

$$
\begin{aligned}
PR &= \{p\} \equiv \text{set of products} \\
CO &= \{c\} \equiv \text{set of components} \\
T &= \{t\} \equiv \text{set of time periods} \\
QL &= \{l\} \equiv \text{set of qualities}
\end{aligned}
$$

For this problem, there are 3 products (p_1, p_2, p_3), 2 components (c_1, c_2), and 3 time periods (t_0, t_1, t_2). The following variables are defined :

$$
\begin{aligned}
x_{c,p,t} &= \text{amount of component } c \text{ allocated to product } p \text{ at period } t \\
s_{p,t} &= \text{stock of product } p \text{ at end of period } t \\
q_{p,l,t} &= \text{quality } l \text{ of product } p \text{ at period } t
\end{aligned}
$$

The objective of the problem is to maximize the total value at the end of the last time period. The terminal value of each product (v_p) is given. Also provided are lower and upper bounds on the qualities of the products, qualities of stocks at start of each time period ($s_{p,t}$), qualities in each component ($QU_{c,l}$), and the product lifting ($LF_{p,t}$) for every period. The data for this problem is provided in Table 4.

The complete mathematical formulation for this problem, consisting of 39 variables and 22 inequality constraints (of which 12 are nonconvex) is given below :

$$
\max \quad \sum_{p \in PR} v_p . s_{p,'t'_2}
$$

$$
\begin{aligned}
s.t. \quad & \\
\sum_{p \in PR} x_{c,p,t} &\leq AR_{c,t} \qquad t \in \{t_1, t_2\},\ c \in CO \\
s_{p,t} + \sum_{c \in CO} x_{c,p,t+1} - s_{p,t+1} &\geq LF_{p,t+1} \qquad t \in \{t_0, t_1\},\ p \in PR \\
s_{p,t} . q_{p,l,t} + \sum_{c \in CO} x_{c,p,t+1} . QU_{c,l} &\geq \\
(s_{p,t+1} + LF_{p,t+1}) &\cdot q_{p,l,t+1} \qquad t \in \{t_0, t_1\},\ p \in PR,\ l \in QL
\end{aligned}
$$

The sources of nonconvexities in this problem are the bilinear terms $s_{p,t} \cdot q_{p,l,t}$ in the last set of constraints. Thus, fixing either the set of s or q variables makes the problem linear in the remaining variables.

The GOP Algorithm: To apply the GOP algorithm to this problem, we can *project* on the qualities (q_1, q_2). Then, the stocks are the *connected* variables. Since there are six of them (corresponding to three products at two time periods), 64 relaxed dual problem problems need to be solved at every iteration. The results of solving this problem using the branch-and-bound GOP and GOP/MILP algorithms are shown in Table 5.

Component Arrivals and Qualities

Component	*Arrivals*			*Qualities*	
	t_0	t_1	t_2	q_1	q_2
c_1	0.20	0.25	0.15	40	80
c_2	0.20	0.15	0.25	100	50

Product Lifting and Limits on Stocks

Product	*Product Lifting*		*Stock Limits*		
	t_1	t_2	t_0	t_1	t_2
p_1	0.08	0.12	0.05	0.10	0.10
p_2	0.15	0.10	0.05	0.10	0.10
p_3	0.15	0.20	0.05	0.10	0.10

Bounds and Initial Values for Product Qualities

Products	*Lower Bounds*		*Upper Bounds*		*Initial Values*	
	q_1	q_2	q_1	q_2	q_1	q_2
p_1	70	50	100	100	70	50
p_2	80	70	100	100	90	70
p_3	60	40	100	100	60	40

Terminal Value of products : $v_p = (60, 90, 40)$.

Table 4 Data for the Multiperiod Tankage Quality Problem

Starting Point	*Original GOP*			*GOP/MILP*	
(y)	Iter.	Subproblems	CPU	Iter	CPU
Lower bound	8	18	3.66	7	14.7
Upper bound	9	19	3.68	9	13.1
$q_{t1} = 100, q_{t2} = 70$	11	18	3.95	13	22.4
$q_{t1} = 80, q_{t2} = 100$	9	19	3.23	13	16.5

Table 5 Multiperiod Tankage Quality Problem

2.4 Problems in Separation Sequences

As in the case of heat exchanger networks, problems involving separations (sharp and nonsharp) can often be posed as a superstructure from which the best alternative is to be selected. The following example considers one such formulation.

Example 2.3 This problem involves the separation of a three component mixture into two multicomponent products using separators, splitters, blenders and pools. The superstructure for the problem (Floudas and Aggarwal, 1990) is given in Figure 7. The NLP formulation for the problem is given below:

$$\min \quad 0.9979 + 0.00432F_5 + 0.01517F_{13}$$

subject to

(Overall Mass Balances)

$$\begin{aligned}
F_1 + F_2 + F_3 + F_4 &= 300 \\
F_6 - F_7 - F_8 &= 0 \\
F_9 - F_{10} - F_{11} - F_{12} &= 0 \\
F_{14} - F_{15} - F_{16} - F_{17} &= 0 \\
F_{18} - F_{19} - F_{20} &= 0
\end{aligned}$$

(Splitter Component Balances)

$$\begin{aligned}
F_5 x_{j,5} - F_6 x_{j,6} - F_9 x_{j,9} &= 0 \quad j = A, B, C \\
F_{13} x_{j,13} - F_{14} x_{j,14} - F_{18} x_{j,18} &= 0 \quad j = A, B, C
\end{aligned}$$

(Inlet Mixer Balances)

$$
\begin{aligned}
0.333F_1 + F_{15}x_{j,14} - F_5x_{j,5} &= 0 \quad j = A, B, C \\
0.333F_2 + F_{10}x_{j,9} - F_{13}x_{j,13} &= 0 \quad j = A, B, C \\
0.333F_3 + F_7x_{A,6} + F_{11}x_{A,9} + F_{16}x_{A,14} + F_{19}x_{A,18} &= 30 \\
0.333F_3 + F_7x_{B,6} + F_{11}x_{B,9} + F_{16}x_{B,14} + F_{19}x_{B,18} &= 50 \\
0.333F_3 + F_7x_{C,6} + F_{11}x_{C,9} + F_{16}x_{C,14} + F_{19}x_{C,18} &= 30
\end{aligned}
$$

(Compositions)

$$x_{A,i} + x_{B,i} + x_{C,i} = 1 \quad i = 5, 16, 9, 13, 14, 16$$

(Sharp Split)

$$x_{B,6} = x_{C,6} = x_{A,9} = x_{C,14} = x_{A,18} = x_{B,18} = 0$$

By projecting on the compositions x_{A_i}, x_{B_i} and x_{C_i}, the primal and relaxed dual subproblems become linear. There are a total of 38 variables and 32 equality constraints. There are initially 20 *connected* variables (the flow rates.) However, considering Figure 7, it is obvious that the recycle streams cannot both be simultaneously active. This leads to solving two independent problems, with $F_{10} = 0$ in the first case and $F_{15} = 0$ in the second case. In each case, the resulting problem has 9 *connected* variables. Application of the GOP algorithm to the problem identifies the optimal solution (shown in Figure 8) in 17 iterations using the parallel configuration as a starting point. The total CPU time taken was 3.84 seconds on an HP730.

2.5 Phase Equilibrium Problems

Phase and Chemical equilibrium problems are of crucial importance in several process separation applications. For conditions of constant pressure and temperature, a global minimum of the Gibbs free energy function describes the equilibrium state. Moreover, the Gibbs tangent plane criterion can be used to test the intrinsic thermodynamic stability of solutions obtained via the minimization of the Gibbs free energy. Simply stated, this criterion seeks the minimum of the distance between the Gibbs free energy function at a given point and the tangent plane constructed from any other point in the mole fraction space. If the minimum is positive, then the equilibrium solution is stable.

The tangent plane criterion for phase stability of an n-component mixture can be formulated as the following optimization problem (McDonald and Floudas, 1995):

$$\min_{y} \quad F(y) \;=\; \sum_{i \in C} y_i \{\mu_i(y) - \mu_i^0(z)\}$$

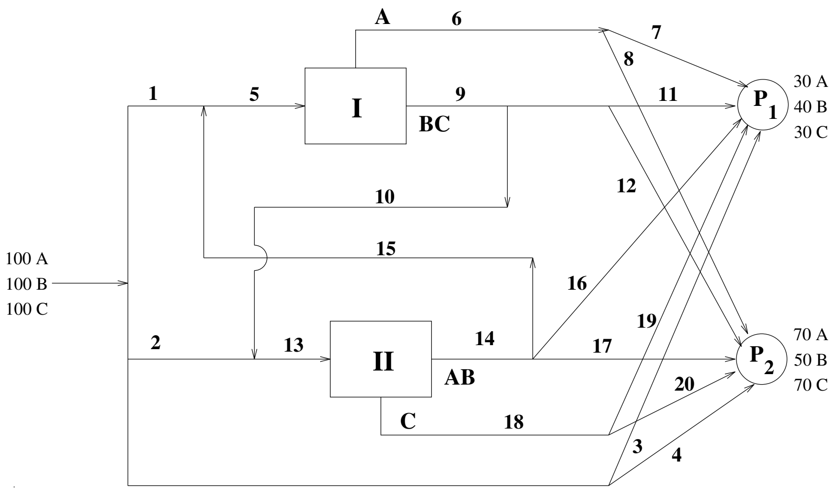

Figure 7 Superstructure for Example 2.3

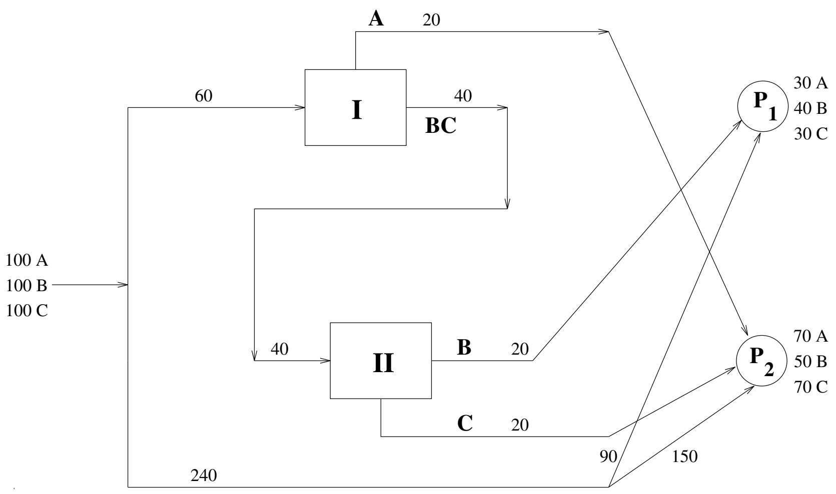

Figure 8 Optimal Configuration For Example 2.3

$$
\begin{aligned}
s.t. \quad \sum_{i \in C} y_i &= 1 \\
0 \leq y_i &\leq 1
\end{aligned}
$$

where y is the mole fraction vector for the various components, $\mu_i(y)$ is the chemical potential of component i, and $\mu_i^0(z)$ represents the tangent constructed to the Gibbs free energy surface at mole fraction z. The use of the NRTL equation for the chemical potential reduces the problem to the following formulation:

$$
\min \quad F(y) = \mathcal{C}(y) + \sum_{i \in C} y_i \cdot \sum_{j \in C} \mathcal{G}_{ij} \tau_{ij} \mathcal{X}_j
$$

$$
\begin{aligned}
s.t. \quad \mathcal{X}_i \cdot \sum_{j \in C} \mathcal{G}_{ji} y_j &= y_i \qquad \forall i \in C \\
\sum_{i \in C} y_i &= 1 \\
0 \leq y_i &\leq 1 \qquad \forall\, i \in C
\end{aligned}
$$

where τ_{ij} are non-symmetric binary interaction parameters, $\mathcal{G}_{ij}$ are parameters introduced for convenience, and the function $\mathcal{C}(y)$ is a convex function. By projecting on y_i, it can be seen that this problem satisfies *Conditions (A)*.

The GOP algorithm was applied to solve several problems in this class. These problems are taken from McDonald and Floudas (1995) and have been solved by them using the GLOPEQ package (McDonald and Floudas, 1994). The results are shown in Table 6. It can be seen that for most of the problems, the GOP algorithm performs very well when compared to the specialized code in GLOPEQ, which is a package specifically designed for phase equilibrium problems.

2.6 An Example In Robust Stability Analysis

The following example was first studied by de Gaston and Sofonov (1988). It concerns the exact computation of the stability margin for a system with real parameter uncertainty. This problem (shown in Figure 9) involves a single-input single-output feedback system with a lead-lag element controller. The model for the problem is given below:

$$
\min \ k_m = x_6
$$

$$
x_5 y_1 - (x_4 + 10x_2 + 10x_3) y_1 + 2x_1 = 0
$$

Problem Name	*Problem Size*			*GOP*		*GLOPEQ**	
	N_X	N_Y	N_C	Iterations	CPU	Iterations	CPU
BAW2L	2	2	3	27	0.68	32	0.15
BAW2G	2	2	3	30	0.75	36	0.16
TWA3T	6	3	4	13	0.86	16	0.22
TWA3G	6	3	4	121	9.00	85	0.96
PBW3T1	6	3	4	82	6.33	53	0.63
PBW3G1	6	3	4	393	35.21	213	2.37
PBW3T6	6	3	4	1366	134.99	549	4.98
PBW3G6	6	3	4	1886	207.19	757	7.09

Table 6 Results for the Phase Stability Problem

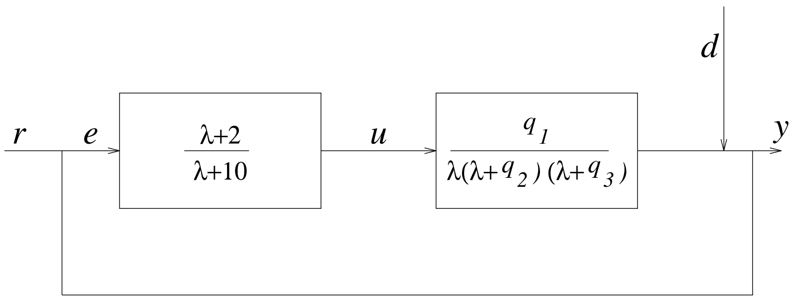

Figure 9 Feedback Structure For Robust Stability Analysis Example

$$
\begin{aligned}
(x_2 + x_3 + 10)y_1 - 10x_4 - x_1 &= 0 \\
x_4 - y_2 x_3 &= 0 \\
x_5 - y_1 &= 0 \\
x_2 - y_2 &= 0 \\
800 - 800x_6 \leq x_1 &\leq 800 + 800x_6 \\
4 - 2x_6 \leq x_2 &\leq 4 + 2x_6 \\
6 - 3x_6 \leq x_3 &\leq 6 + 3x_6
\end{aligned}
$$

Details of the development of the model can be found in Psarris and Floudas (1993). The optimal solution for this problem is $k_m = 0.3417$. Application of the GOP algorithm to this problem converges to the optimal solution in 45 iterations, requiring 1.5 seconds on an HP730.

2.7 Concave and Indefinite Quadratic Problems

The conditions under which the GOP algorithm can be applied make it highly attractive for problems with quadratic functions in the objective and/or constraints. Of particular interest are quadratic problems with linear constraints, which occur as subproblems in successive quadratic programming (SQP) and other optimization techniques, as well as being interesting global optimization problems in their own right. In this section, the results of applying the GOP and GOP/MILP algorithms to various problems of this type is discussed.

2.8 Problems from the literature

Eleven small-size concave quadratic problems from Phillips and Rosen (1988) have been solved using the GOP algorithm. The problems have the following form:

Problem	*Problem Size*			*GOP Algorithm*		P&R
	m	n	k	Iterations	CPU (HP730)	CPU (CRAY2)
1	5	2	0	3	0.09	0.026
2	5	6	0	2	0.07	0.022
3	5	6	0	2	0.06	0.020
4	5	6	0	2	0.03	0.026
5	4	2	0	4	0.12	0.017
6	4	3	0	4	0.11	0.015
7	4	3	0	4	0.14	0.014
8	10	3	0	17	0.50	0.022
9	10	3	0	8	0.20	0.020
10	4	4	0	3	0.18	0.029
11	9	2	1	3	0.08	0.023

Table 7 Test Problems from Phillips and Rosen (1988) ($\epsilon = 0.001$)

$$
\begin{array}{rcl}
\min\limits_{x,y\in\Omega} & & \psi(x,y) = \theta_1\varphi(x) + \theta_2 d^T y \\
s.t. \quad \varphi & = & 0.5\sum\limits_{i=1}^{n}\lambda_i(x_i - \overline{\omega_i})^2, \\
\Omega & = & \{(x,y) : A_1 x + A_2 y \le b, x \ge 0, y \ge 0\}, \\
x,\ \lambda,\ \overline{\omega} & \in & \Re^n, \quad y,\ d \in \Re^k \\
A_1 & \in & \Re^{m\times n}, \quad A_2 \in \Re^{m\times k} \\
\theta_1,\ \theta_2 & \in & \Re.
\end{array}
\tag{4.1}
$$

Here, m is the number of linear constraints, n is the number of concave variables (x), and k is the number of linear variables (y). The parameters θ_1 and θ_2 are -1 and 1 respectively, and the relative tolerance for convergence between the upper and lower bounds (ϵ) is 0.001.

The results of the application of the algorithm to these problems are given in Table 7. The CPU times for the GOP algorithm and the Phillips and Rosen algorithm (denoted by P&R) are given in seconds. It should be noted that the P&R algorithm was run on a CRAY2. As can be seen, the algorithm solves problems of this size very fast, taking about 5 iterations to identify and converge to the optimal solution.

Problem Name	*Problem Size*			*GOP*		*Sherali & Tuncbilek*	
	N_x	N_y	N_c	Iterations	CPU	Iterations	CPU
CQP1	10	10	11	27	0.68	32	0.15
CQP3	20	20	10	11	10.84	3	3.29
CQP4	20	20	10	4	3.57	1	2.61
CQP5	20	20	10	11	10.91	1	2.55
CQP6	20	20	10	5	5.07	1	2.61
CQP7	20	20	10	229	177.04	11	15.94
IQP1	20	20	10	3	0.65	3	2.73

Table 8 Quadratic Problems from Sherali and Tuncbilek (1994).

Results from application of the GOP algorithm to another set of concave and indefinite quadratic test problems taken from Floudas and Pardalos (1990) are given in table 8. These problems have also been solved recently by Sherali and Tuncbilek (1994) whose results are listed in the same table. Here, N_x, N_y and N_c refer to the number of x and y variables and the number of linear constraints respectively.

Run	Problem size			Iterations	CPU (sec)	
	m	n	k		GOP	GOP/MILP
CLR1	50	50	50	2.3	0.510	0.317
CLR2	50	50	100	3.0	5.736	2.254
CLR3	50	50	200	4.33	27.620	8.293
CLR4	50	50	300	5.0	----	8.977
CLR5	50	100	50	3.5	32.07	5.665
CLR6	50	100	150	6.8	----	38.892
CLR7	100	100	100	2.2	3.485	31.147
CLR8	100	200	100	3.8	----	100.370
CLR9	100	250	100	3.6	----	267.124

Table 9 Concave Quadratic Problems from Phillips and Rosen (1988), $\epsilon = 0.01$

Randomly Generated Quadratic Problems

This section describes the application of the GOP and GOP/MILP algorithms to randomly generated problems of the form (4.1). Such problems have earlier been

Run	Problem size			Iterations	CPU (sec)	
	m	n	k		GOP	GOP/MILP
CLR1	50	50	50	2.0	0.120	0.116
CLR2	50	50	100	2.0	0.145	0.141
CLR3	50	50	200	2.2	6.047	1.574
CLR4	50	50	500	3.0	----	14.125
CLR5	50	100	100	2.0	0.217	1.373
CLR6	50	100	200	2.0	0.360	11.982
CLR7	100	100	100	2.0	0.305	0.306
CLR8	100	100	200	2.0	0.374	0.369
CLR9	100	100	200	2.0	0.374	0.369
CLR10	100	100	500	3.0	----	80.028
CLR11	100	150	400	1.7	----	182.208

Table 10 Concave Quadratic Problems from Phillips and Rosen (1988), $\epsilon = 0.1$

Run	Problem size			$\epsilon = 0.1$		$\epsilon = 0.01$	
	m	n	k	Iter	CPU	Iter	CPU
ILR1	25	25	25	2.0	0.232	2.200	0.312
ILR2	25	25	50	2.0	0.416	2.600	0.606
ILR3	25	25	100	2.2	1.522	3.000	2.030
ILR4	25	50	100	4.0	13.19	11.50	37.56
ILR5	50	50	50	2.0	0.864	2.400	1.504
ILR6	50	50	100	2.0	1.264	2.800	3.018
ILR7	25	75	100	3.0	68.86	30.00	294.3
ILR8	50	75	100	2.0	1.564	3.600	9.724
ILR9	75	75	100	2.0	2.120	2.800	6.304
ILR10	25	75	150	4.0	115.80	----	----
ILR11	50	75	150	2.2	9.5380	----	----
ILR12	75	75	150	2.0	2.9560	----	----
ILR13	25	100	50	3.6	23.21	23.50	118.6
ILR14	50	100	50	2.2	2.130	3.800	6.510
ILR15	75	100	50	2.2	3.544	2.800	5.244

Table 11 Indefinite Quadratic Problems from Phillips and Rosen (1988), $\epsilon = 0.1$ and 0.01

studied by Phillips and Rosen (1988), and we generated the data for the constants $\lambda, \overline{\omega}, d, A_1, A_2$ and b as they have used. The parameters θ_1 and θ_2 have been set to values of -0.001 and 0.1 respectively. Depending on the values of λ_i, the problems generated are either concave quadratic or indefinite quadratic problems. For the case of indefinite quadratic problems, roughly as many postive λ_i as negative λ_i are generated. For each problem size, 5-10 different problems (using various seeds) have been generated and solved.

Tables 9 and 10 present the results for concave quadratic problems using tolerances of 0.01 and 0.1 respectively, while Table 11 presents the results for indefinite quadratic problems using tolerances of 0.01 and 0.1 with the GOP algorithm. In all the cases, it can be seen that the algorithm generally requires very few iterations for the upper and lower bounds to be within 10% of the optimal solution; generally, the convergence to within 1% is achived in a few more iterations. Moreover, certain trends are noticeable in all cases. For example, as the number of constraints (m) grows, the problems generally become easier to solve. Conversely, as the size of the linear variables (k) increases, the algorithm requires more time for the solution of the dual problems, leading to larger overall CPU times. In general, these results indicate that the GOP and GOP/MILP algorithms can be very effective in solving medium sized quadratic problems with several hundred variables and constraints.

It should be noted that several sizes of these problems have also been solved on a supercomputer using a specially parallelized version of the GOP algorithm. The results can be found in Androulakis *et al.* (1995).

3 CONCLUSIONS

Visweswaran and Floudas (1995) proposed new formulations and branching strategies for the GOP algorithm for solving nonconvex optimization problems. In this paper, a complete implementations of various versions of the algorithm has been discusssed. The new formulation as a branch and bound algorithm permits a simplified implementation. The resulting package **cGOP** has been applied to a large number of engineering design and control problems as well as quadratic problems. It can be seen from the results that the implementation permits very efficient solutions of problems of medium size.

Acknowledgments

Financial support from the National Science Foundation under grant CTS-9221411 is gratefully acknowledged.

REFERENCES

[1] I. P. Androulakis, V. Visweswaran, and C. A. Floudas. Distributed Decomposition-Based Approaches in Global Optimization. In *Proceedings of State of the Art in Global Optimization: Computational Methods and Applications (Eds. C.A. Floudas and P.M. Pardalos)*, Kluwer Academic Series on Nonconvex Optimization and Its Applications, 1995. To Appear.

[2] T.E. Baker and L.S. Lasdon. Successive linear programming at Exxon. *Mgmt. Sci.*, 31(3):264, 1985.

[3] A. Ben-Tal and V. Gershovitz. Computational Methods for the Solution of the Pooling/Blending Problem. Technical report, Technion-Israel Institute of Technology, Haifa, Israel, 1992.

[4] R. R. E. de Gaston and M. G. Sofonov. Exact calculation of the multiloop stability margin. *IEEE Transactions on Automatic Control*, 2:156, 1988.

[5] C. A. Floudas and A. Aggarwal. A decomposition strategy for global optimum search in the pooling problem. *ORSA Journal on Computing*, 2(3):225, 1990.

[6] C. A. Floudas, A. Aggarwal, and A. R. Ciric. Global optimum search for nonconvex NLP and MINLP problems. *C&ChE* , 13(10):1117, 1989.

[7] C. A. Floudas and A. R. Ciric. Strategies for overcoming uncertainties in heat exchanger network synthesis. *Comp. & Chem. Eng.*, 13(10):1133, 1989.

[8] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1990.

[9] C. A. Floudas and V. Visweswaran. A global optimization algorithm (GOP) for certain classes of nonconvex NLPs: I. theory. *C&ChE*, 14:1397, 1990.

[10] C. A. Floudas and V. Visweswaran. A primal-relaxed dual global optimization approach. *J. Optim. Theory and Appl.*, 78(2):187, 1993.

[11] R. E. Griffith and R. A. Stewart. A nonlinear programming technique for the optimization of continuous processesing systems. *Manag. Sci.*, 7:379, 1961.

[12] Studies of the Behaviour of Recursion for the Pooling Problem. *ACM SIGMAP Bulletin*, 25:19, 1978.

[13] Behaviour of Recursion Model - More Studies. *SIGMAP Bulletin*, 26:22, 1979.

[14] L.S. Lasdon, A.D. Waren, S. Sarkar, and F. Palacios-Gomez. Solving the Pooling Problem Using Generalized Reduced Gradient and Successive Linear Programming Algorithms. *ACM SIGMAP Bulletin*, 27:9, 1979.

[15] W. B. Liu and C. A. Floudas. A Remark on the GOP Algorithm for Global Optimization. *J. Global Optim.*, 3:519, 1993.

[16] C.D. Maranas and C.A. Floudas. A Global Optimization Approach for Lennard--Jones Microclusters. *J. Chem. Phys.*, 97(10):7667, 1992.

[17] C.M. McDonald and C.A. Floudas. *A user guide to GLOPEQ*. Computer Aided Systems Laboratory, Chemical Engineering Department, Princeton University, NJ, 1994.

[18] C.M. McDonald and C.A. Floudas. Global Optimization for the Phase Stability Problem. *AICHE Journal*, 41:1798, 1995.

[19] F. Palacios-Gomez, L.S. Lasdon, and M. Engquist. Nonlinear Optimization by Successive Linear Programming. *Mgmt. Sci.*, 28(10):1106, 1982.

[20] A parallel algorithm for constrained concave quadratic global minimization. *Mathematical Programming*, 42:421, 1988.

[21] Polycarpos Psarris and C. A. Floudas. Robust Stability Analysis of Linear and Nonlinear Systems with Real Parameter Uncertainty. *Journal of Robust and Nonlinear Control*, 1994. Accepted for publication.

[22] I. Quesada and I. E. Grossmann. Global Optimization Algorithm for Heat Exchanger Networks. *I&EC Res.*, 32:487, 1993.

[23] H. Sherali and C. H. Tuncbilek. Tight Reformulation-Linearization Technique Representations for Solving Nonconvex Quadratic Programming Problems. *Submitted for Publication*, 1994.

[24] V. Visweswaran and C. A. Floudas. New Formulations and Branching Strategies for the GOP Algorithm. In *Global Optimization in Engineering Design*, (Ed.) I. E. Grossmann, Kluwer Book Series in Nonconvex Optimization and Its Applications, Chapter 3, 1995a.

[25] V. Visweswaran and C. A. Floudas. *cGOP: A User's Guide*. Princeton University, Princeton, New Jersey, 1995b.

Appendix A: Implementation of the GOP and GOP/MILP Algorithms

This section describes the key features of the implementation of the GOP and GOP/MILP algorithms. In particular, the interaction of the various subroutines and the storage and transfer of relevant data between these routines are crucial to the efficiency of the algorithm, and are therefore discussed in some detail. The implementation has been written so as to be a useful framework in the development of any generic branch and bound algorithms for global optimization.

Overview of the cGOP package

The **cGOP** package is written entirely in the C programming language, and consists of approximately 8000 lines of source code, of which around 30% are comments. The algorithms can be called either in standalone mode or as subroutines from within another program. The primal and relaxed dual subproblems are solved either using CPLEX 2.1 (for linear or mixed integer linear) problems or MINOS 5.4 for nonlinear problems. Various options are available to change the routines that are used, such as obtaining tighter bounds on the x variables and $g_i^k(y)$ (the gradients of the Lagrange function), as well as solving the full problem as a local optimization problem at each node.

Data Structures

Since the **cGOP** package is written in C, it is highly convenient to aggregate the data transfer from one routine to another using structures (equivalent to COMMON blocks in Fortran). The primary data structures used in the package describe the problem data, the solutions of the various primal problems, the data for the various Lagrange functions, and the solutions of the relaxed dual subproblems at each iteration.

The most important group of data is obviously the problem data itself. In order to facilitate easy and general use of this data, the implementation was written assuming that the following types of problems would be solved:

$$\begin{array}{lll}
\min\limits_{x,y} & c_0^T x + d_0^T y + x^T Q_0 y + F_0(x) + G_0(y) & \\
s.t. & l_j \leq c_j^T x + d_j^T y + x^T Q_j y \leq u_j, & i = 1, \ldots, M_1 \\
& F_j(x) + G_j(y) \leq u_j & i = M_1 + 1, \ldots, M_2 \\
& \quad L \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq U &
\end{array} \tag{4.2}$$

where $j = 1, \ldots, M_1$ are the set of bilinear constraints, and $j = M_1 + 1, \ldots, M_2$ are the set of general nonlinear constraints. It is assumed that the functions $F_j(x)$ and $G_j(y)$ are convex in x and y respectively. Under this assumption, it can easily be shown that (4.2) satisfies *Conditions (A)*. Note also that while the bilinear constraints can be equalities or inequalities, the other nonlinear terms in the constraints are assumed to lie in convex inequalities.

Given the formulation (4.2), the data for the problem can be separated into one part containing the linear and bilinear terms, and another part containing the nonlinear terms $F_i(x)$ and $G_i(y)$. The first part can be specified through a data file or as arguments during the subroutine call that runs the algorithm. The nonlinear terms, which in general cannot be specified using data files, can be given through user defined subroutines that compute the contribution to the objective function and constraints from these terms, as well as their contribution to the Hessian of the objective function and the Jacobian of the constraints. The problem data is therefore carried in one data structure (called *pdat* from here on, and shown in Figure 10) that describes the following items:

Control Data This refers to the type of the problem (bilinear, quadratic, nonlinear, etc), number of x and y variables, the number of constraints, type and value of the starting point for the y variables, as well as tolerances for convergence.

Bilinear Data For reasons of convenience, the linear and bilinear terms in the objective function and constraints are treated together. The data is stored in sparse form, with only the nonzero terms being stored. For each term, the value of the term as well as the indices of its x and/or y terms are stored.

Bounds The global bounds on the variables (which can be changed before the start of the algorithm, but thereafter remain constant) are stored in arrays.

Nonlinear Data The pointers to the functions that compute the nonlinear terms and their gradients are stored in the data structure.

Iteration Data Various counters and loop variables that control and aid in the progress of the iterations are stored in the main data structure. In addition, the best solution obtained by the algorithm so far is also stored.

It is important to note that almost all of the main data structure, once it has been read in from the data file or passed to the main subroutine in the algorithm, remains constant throughout the progress of the algorithm. The only exceptions are the iteration variables and the best solution obtained by the algorithm so far.

The solution of the primal problem is stored together as another data structure, *psol* (shown in Figure 11). This contains the value of y^K for which the primal problem was solved, solution for the x variables, the marginals for all the constraints and variables at their bounds, as well as an indicator of whether the primal was feasible or not.

Because of the form (4.2), the Lagrange function (for iterations with feasible primal problems) can be written (after linearization of the terms with respect to x and substitution of the KKT optimality conditions for the primal problem) as

$$L(x, y, \lambda^K)\big|_{x^K}^{lin} = L_C + L_L^T y + \sum_{i=1}^{NI_c^K} x_i g_i^T (y - y^K) + G'(y)$$

where $G'(y)$ represents all the nonlinear terms weighted by the marginals, and can be written as

$$G'(y) = G_0(y) + \sum_{j=M_1+1}^{M_2} \lambda_j^K G_j(y)$$

By introducing new variables to represent the nonlinear constraints, the Lagrange function can be rewritten as

$$\begin{aligned} L(x, y, \lambda^K)\big|_{x^K}^{lin} &= L_C + L_L^T y + \sum_{i=1}^{NI_c^K} x_i g_i^T (y - y^K) + \sum_{j=M_1+1}^{M_2} \lambda_j^K z_j && (4.3) \\ z_j &\geq G_0(y) + G_j(y) && (4.4) \end{aligned}$$

Note that a simplistic implementation of the algorithm for the general nonlinear problem in (4.2) leads to a problem with nonlinear terms in each Lagrange function, making it much more computationally intensive. Given the fact that the nonlinear terms are the same in each Lagrange function except for a factor due to the marginals λ_j^K, it is far more efficient to group the terms together, and therefore to compute their gradients only once. Moreover, the regrouping of the terms means that as far as

```
struct pdat {
  /* Control section */
  char     *probname;     /* Name of original problem */
  char     objtype;       /* Type of objective function */
  char     contype;       /* Type of constraints */
  char     primaltype;    /* Type of primal problems */
  char     rdualtype;     /* Type of relaxed dual problems */
  int      nxvar;         /* Number of x variables */
  int      nyvar;         /* Number of y variables */
  int      ncon;          /* Number of constraints */
  int      nzcnt;         /* Total number of non-zeros */

  /* Data */
  char     *ctype;        /* Type of X and Y variables */
  int      *sense;        /* Sense of row:  <=, ==, >= */
  double   *rhs;          /* Right hand sides of the rows */
  int      *count;        /* Number of entries in each row */
  int      *begin;        /* Start of entries for each row */
  TERMS    terms;         /* Bilinear terms in problem */
  double   *xlbd, *xubd; /* Bounds on X variables */
  double   *ylbd, *yubd; /* Bounds on Y variables */
  double   objconst;      /* Constants in the objective */
  double   epsa;          /* Absolute tolerance specified */
  double   epsr;          /* Relative tolerance specified */
  int      maxiter;       /* Maximum number of iterations */

  /* Various functions */
  void     userobj();     /* Nonlinear terms in objective */
  void     usercon();     /* Nonlinear terms in constraints */

  /* Solution */
  int      niter;         /* Number of iterations so far */
  double   primalubd;     /* Current upper bound from primals */
  double   rdlbd;         /* Current lower bound from duals */
  double   *x;            /* Starting point, solution for X */
  double   *y;            /* Starting point, solution for Y */
  double   abserror;      /* Absolute error between bounds */
  double   relerror;      /* Relative error between bounds */
};
```

Figure 10 Main data structure for the GOP and GOP/MILP algorithms

```
struct psol {
  int      modstat;      /* Feasible or infeasible */
  int      nxvar;        /* Number of x variables  */
  int      nyvar;        /* Number of y variables  */
  int      ncon;         /* Number of constraints */
  double   *yval;        /* Fixed values for Y variables */
  double   objval;       /* Objective value for primal */
  double   *varval;      /* Solution for X variables */
  double   *cmargval;    /* Marginals for constraints */
  double   *bmargval;    /* Marginals for bounds */
  char     *varstat;     /* Status for each variable */
  char     *solver;      /* Which solver was used */
};
```

Figure 11 Solution of the Primal Problem

```
/* Structure to hold the data for the Lagrange function */
typedef struct lagdata {
  int      NIc;          /* Number of connected X */
  int      nyvar;        /* Number of Y variables */
  double   *xlbd;        /* Lower bounds for connected X */
  double   *xubd;        /* Lower bounds for connected X */
  int      *xindex;      /* Indices of connected X */
  double   *ylbd, *yubd; /* Bounds on Y variables */
  double   *glbd, *gubd; /* Bounds on qualifying constraints */
  double   **glin;       /* Terms in qualifying constraints */
  double   *gconst;      /* Constants in qualifying const. */
  double   *llin;        /* Terms in Lagrange function */
  double   lconst;       /* Constants in Lagrange function */
};
```

Figure 12 Lagrange function data structure each individual Lagrange function is concerned, only the data regarding (4.3) need to be stored, .e. the coefficients of the linear terms L_L^T, the bilinear terms g_i^T and the multipliers λ_j^K. Its structure is shown in Figure 12.

The solutions of the relaxed dual subproblems comprise the last major data structure. Apart from the actual objective value for the solution and the values of the y variables, this data includes information about which iteration and parent node generated each child node in the branch and bound tree. Thus, the entire information about the tree is stored in the array of relaxed dual solution structures, *rdsol*.

Based upon these various data units, the overall scheme of the implementation is now presented. A pictorial view of the algorithm is given in Figure (13).

Initialization of parameters

At the start of the algorithm, the list of relaxed dual solutions *rdsol* is initialized to contain the starting point for the y variables, indicating the root node for the whole branch and bound tree. An initial local optimization problem can be solved to find a good upper bound and starting point for the y variables, if desired. Various counters and bookkeeping variables are initialized before the start of the iterations.

Selection Of Previous Lagrange Functions and Current Region

At any given iteration, the relaxed dual subproblems will contain a Lagrange function from the current iteration, and one from each of the parent nodes of the current node in the branch and bound tree. In order to select these functions, a backward search is done through the list of solutions to the relaxed dual problems starting from the current node (i.e. the node that has been chosen at the end of the previous iteration). The following steps are repeated:

Step 0. Initialize lagsel[MAXITER], the array of parent nodes for the current node.

Step 1. Add the current node C to lagsel. Set lagsel[1] = C, and set the number of Lagrange functions $numlag = 1$.

Step 2. Find the iteration P that generated the current node.

Step 3. Go to the node corresponding to iteration P (say node D) and add this node to the list, i.e. set $numlag = numlag + 1$, lagsel[$numlag$] = D.

Step 4. Repeat Steps 2 and 3 until the root node has been reached.

Start Of The Algorithm

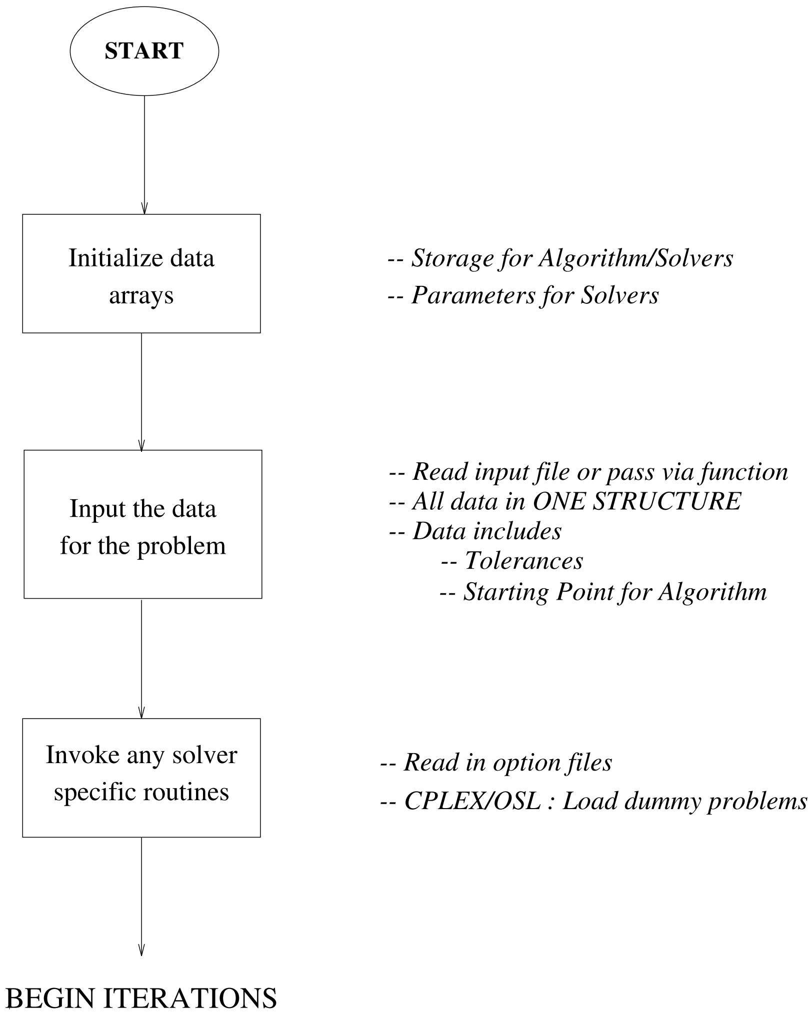

Figure 13 Implementation of the GOP Algorithm in C

Primal Problem

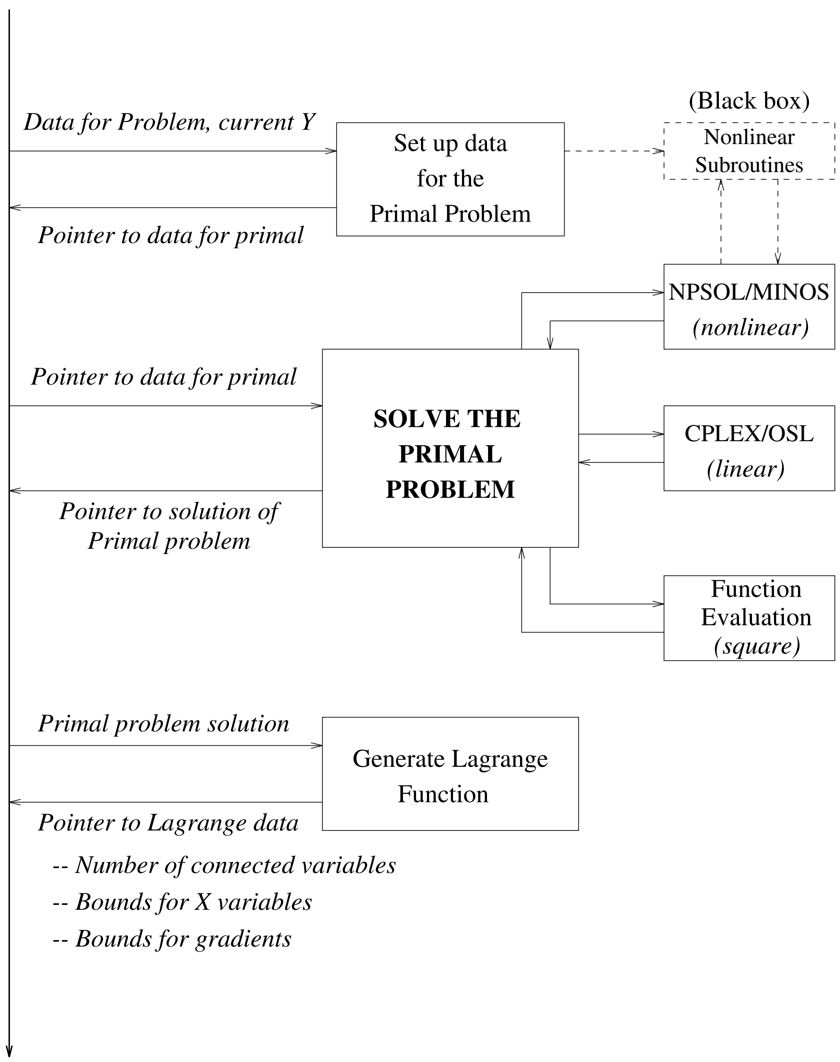

Figure 13 *(continued)* Implementation of the GOP Algorithm in C

Relaxed Dual Problem

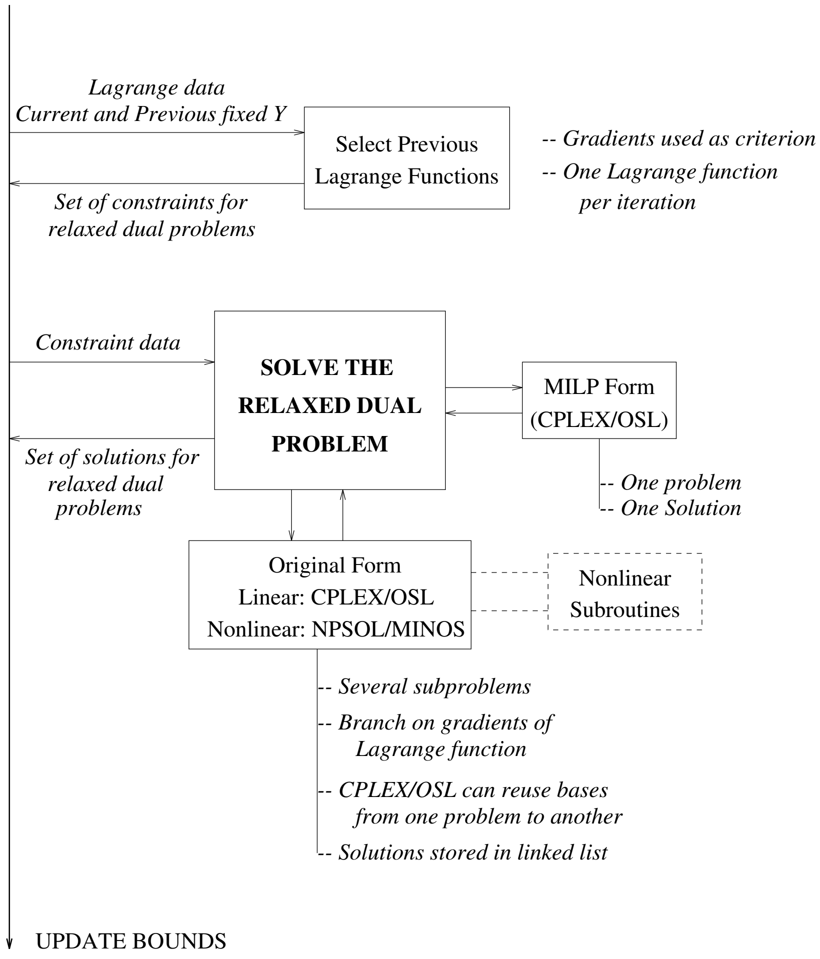

Figure 13 *(continued)* Implementation of the GOP Algorithm in C

Selecting The Best Solution and Lower Bound

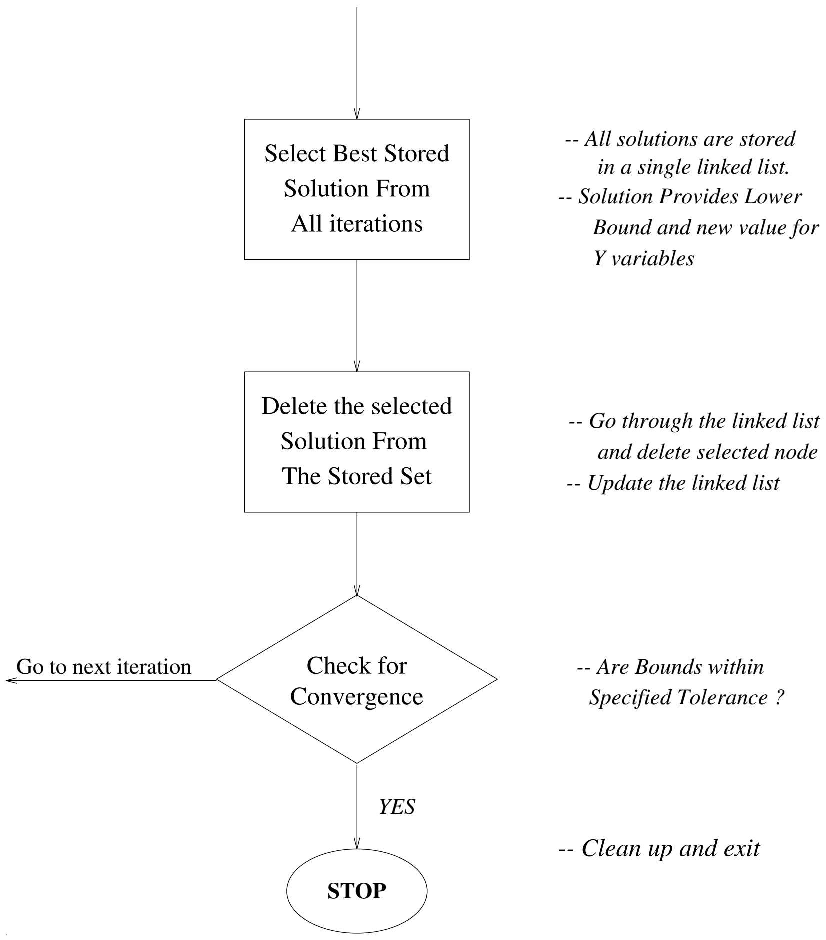

Figure 13 *(continued)* Implementation of the GOP Algorithm in C

The list of nodes generated in the above steps provides a set of *qualifying* constraints (one set per node) that define the region of operation for the current node.

Obtaining Tighter Bounds For The x Variables

If desired, a set of bounds problems are solved that try to find the tightest bounds on the x and y variables given any linear and convex constraints in the original problem, and the current region for the y variables as defined by the *qualifying* constraints for the parent nodes of the current node. This is a very important step, because the tightness of the bounds on the x variables is crucial to obtaining tight underestimators for the relaxed dual problems.

Primal problem

The primal problem takes as data the *pdat* structure, along with the current vector for y^K. It is also given the current region for the problem as defined by the selected *qualifying* constraints. There are several schemes that can be followed to solve the primal problem, all of which involve various combinations of the primal, relaxed primal or a local optimization problem solved in the current region. One possible scheme is as follows:

1. Solve the primal problem at the current y^K.
2. If the primal problem is feasible, update the upper bound.
   (a) Solve the full NLP as a local optimization problem in the current region.
   (b) If the NLP solution is lower than the upper bound, replace y^K with the NLP solution and go to Step 1. Otherwise go to Step 4.
3. If the primal problem is infeasible
   (a) Solve the full NLP as a local optimization problem in the current region.
   (b) If the NLP provides a feasible solution, then replace y^K with the new solution from the NLP and go to Step 1. Otherwise, solve the relaxed primal problem go to Step 4.
4. Return the solution of the problem as a *psol* data structure.

Determination Of Connected Variables

The solution of the current primal (or relaxed primal) problem is used to determine the set of *connected* variables. Several reduction tests are used to determine the set. These include testing for the lower and upper bounds on the gradients of the Lagrange function and the tightness of the bounds on the x variables. If the lower and upper bounds on an x variable are within a certain tolerance, that variable can be fixed at its bound. Provision is also made for user defined tests for reducing the number of *connected* variables.

Generation of Lagrange Function Data

As mentioned earlier, only the data for the Lagrange functions (4.3) are stored. This data is generated from the current *psol* structure. Once the data is generated, it can be used again whenever the Lagrange functions from that iteration need to be generated.

Global Lagrange functions

If there are no connected variables in the Lagrange function generated at the current iteration, then this function contains only the y variables. Therefore, it is a valid underestimator for the entire y space, and can be included as a cut for all future relaxed dual subproblems. In such a case, the current Lagrange function is added to the list of ''global'' Lagrange functions.

Relaxed Dual Problem

Given the current region and a set of *connected* variables, the region is partitioned using the *qualifying* constraints of the current Lagrange function. Then, a relaxed dual subproblem is solved in each region, and the solutions are stored as part of *rdsol* if feasible. The nonlinear terms in the objective function and constraints are again incorporated through calls to the user defined functions. In the case of the GOP/MILP algorithm, only one MILP problem needs to be solved.

Selection of the Lower Bound

After the relaxed dual problem has been solved for every possible combination of the bounds of the connected variables (in the case of the GOP/MILP algorithm, after the MILP has been solved), a new lower bound needs to be determined for the global solution. Since the solutions are all stored as a linked list, this permits a simple search for the best solution. This solution is then removed by simply removing the corresponding node from the linked list. At the same time, the corresponding value of y is also extracted to use for the next iteration.

Resolving the MILP Formulation

In the case of the GOP/MILP formulation, after a solution has been selected from the list of candidate solutions, the MILP formulation corresponding to the iteration from which the solution was generated needs to be resolved. To accomplish this, a binary cut that excludes the selected solution is generated and added to the MILP formulation, which is then solved. Because of the likelihood that the formulation for any given iteration is likely to be solved again and again at least a few times, several such formulations are stored in memory, so that when they are resolved, it is merely a matter of restarting the problem with the additional binary cut. This saves valuable loading and startup time for the solution of these problems.

Convergence

Finally, the check for convergence is done. The algorithm is deemed to have converged if the relative difference between the upper bound from the primal problems and the lower bound from the relaxed dual problems is less than ϵ. Then, the algorithm terminates (in the case of the standalone version) or returns to the calling routine (in case of the subroutine version). Otherwise, the algorithm continues with the new fixed value of y for the primal problem found from the previous step.