A Global Optimization Method, αBB, for General Twice-Differentiable Constrained NLPs: II – Implementation and Computational Results

C.S. Adjiman, I.P. Androulakis, and C.A. Floudas[1]
Department of Chemical Engineering
Princeton University
Princeton, NJ 08544-5263

Abstract

Part I of this paper (Adjiman *et al.*, 1997) described the theoretical foundations of a global optimization algorithm, the αBB algorithm, which can be used to solve problems belonging to the broad class of twice-differentiable NPLs. For any such problem, the ability to automatically generate progressively tighter convex lower bounding problems at each iteration guarantees the convergence of the branch-and-bound αBB algorithm to within ϵ of the global optimum solution. Several methods were presented for the construction of convex valid underestimators for general nonconvex functions. In this second part, the performance of the proposed algorithm and its alternative underestimators is studied through their application to a variety of problems. An implementation of the αBB is described and a number of rules for branching variable selection and variable bound updates are shown to enhance convergence rates. A user-friendly parser facilitates problem input and provides flexibility in the selection of an underestimating strategy. In addition, the package features both automatic differentiation and interval arithmetic capabilities. Making use of all the available options, the αBB algorithm successfully identifies the global optimum solution of small literature problems, of small and medium size chemical engineering problems in the areas of reactor network design, heat exchanger network design, reactor-separator network design, of generalized geometric programming problems for design and control, and of batch process design problems with uncertainty.

[1]Author to whom all correspondence should be addressed.

1 Introduction

In the first part of this paper, Adjiman *et al.* (1997) presented an optimization algorithm, the αBB algorithm, whose theoretical properties guarantee convergence to the global optimum solution of twice-differentiable NLPs of the form

$$\begin{array}{lrcl} \min\limits_{\boldsymbol{x}} & f(\boldsymbol{x}) & & \\ s.t. & \boldsymbol{g}(\boldsymbol{x}) & \leq & 0 \\ & \boldsymbol{h}(\boldsymbol{x}) & = & 0 \\ & \boldsymbol{x} & \in & X \subseteq \Re^n \end{array} \tag{1}$$

where f, $\boldsymbol{g}$ and $\boldsymbol{h}$ belong to $\mathcal{C}^2$, the set of functions with continuous second-order derivatives, and $\boldsymbol{x}$ is a vector of size n.

The αBB algorithm is based on a branch-and-bound approach, where a lower bound on the optimal solution is obtained at each node through the automatic generation of a valid convex underestimating problem. If the functions contain any bilinear, trilinear, fractional, fractional trilinear or univariate concave terms, these may be underestimated through their convex envelope. A valid convex underestimator $\mathcal{L}_{nt}(\boldsymbol{x})$ for a general nonconvex term $nt(\boldsymbol{x})$ is constructed by subtracting a sufficiently large positive quadratic term from $nt(\boldsymbol{x})$:

$$\mathcal{L}_{nt}(\boldsymbol{x}) = nt(\boldsymbol{x}) - \sum_{i=1}^{n} \alpha_i (x_i^U - x_i)(x_i - x_i^L). \tag{2}$$

A necessary and sufficient condition for the convexity of $\mathcal{L}_{nt}(\boldsymbol{x})$ is the positive semidefiniteness of its Hessian matrix given by $H_{nt}(\boldsymbol{x}) + 2\mathrm{diag}(\alpha_i)$. $H_{nt}(\boldsymbol{x})$ is the Hessian matrix of $nt(\boldsymbol{x})$.

Several methods were presented for the rigorous computation of a valid diagonal shift matrix $\Delta = \mathrm{diag}(\alpha_i)$ satisfying the convexity condition. All procedures require the use of interval analysis to obtain an interval Hessian matrix or an interval characteristic polynomial valid at the current node of the branch-and-bound tree. The computational complexity and the accuracy of the α calculations differ from scheme to scheme and one of the aims of the present paper is to study the effect of these variations on the performance of the algorithm.

Before the αBB algorithm is applied to a set of example problems, a current implementation is presented. The main characteristics of the package, such as branching schemes, variable bound update strategies, automatic differentiation, interval arithmetic and a user-friendly parser, are outlined. An extensive computational study is then carried out. The test problems chosen vary in size and type. The first series of examples consists of small problems often encountered in the literature. These are followed by chemical engineering problems representing reactor network design, heat exchanger design and separation system design. Then, a number of generalized geometric programming problems are addressed. Finally, some large batch scheduling problems with uncertain parameters are solved.

2 Algorithmic Issues

Using the theoretical advances described in Part I of this paper (Adjiman *et al.*, 1997), a user-friendly implementation of the algorithm was developed. The main objectives were to create an efficient optimization environment for NLP problems involving twice-differentiable functions, while ensuring that the user interface remains simple and flexible. The structure of the algorithm, illustrated in Figure 1, requires the implementation to possess the following features:

- several alternatives for the construction of the lower bounding problem, depending on the mathematical characteristics of the terms appearing in the problem,
- an interface with MINOPT (Schweiger *et al.*, 1997), which contains a collection of local NLP and MINLP solvers.
- the capability to derive Hessian matrices (second-order derivatives) or characteristic polynomials via automatic differentiation,
- the ability to generate interval Hessian matrices and to perform interval arithmetic,
- α calculation functions for each of the methods presented in Part I,
- several branching strategies,
- several variable bounds update strategies.

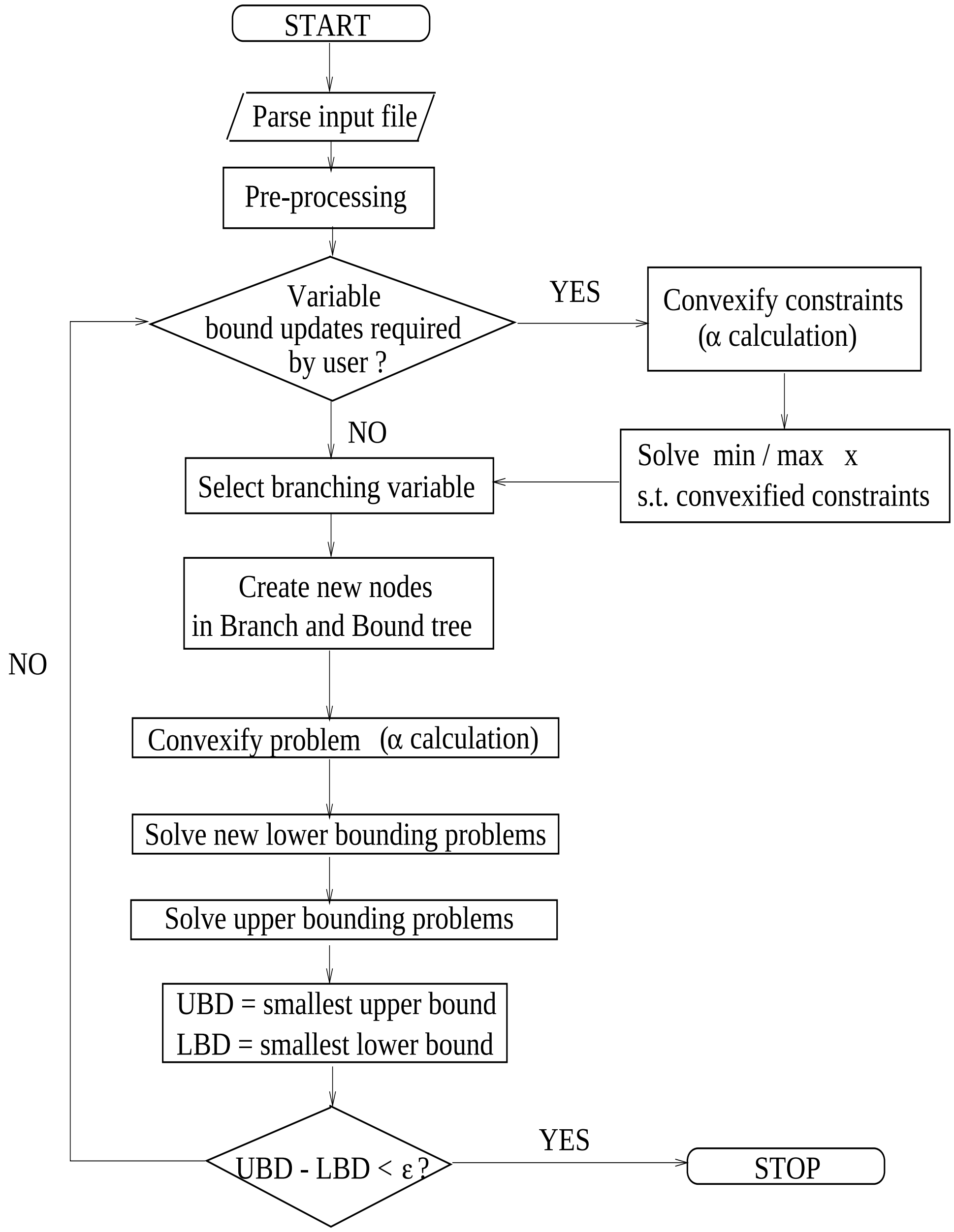

Figure 1: Flowchart for the αBB algorithm

The following discussion outlines the branching and variable bounding rules which are used in order to enhance the performance of the αBB algorithm. It also gives the rationale for the design of the different components of the current implementation of the αBB algorithm.

2.1 Branching Strategies

Although it does not present any theoretical difficulties, the branching step of any branch-and-bound algorithm often has a significant effect on the rate of convergence. This is especially true of the αBB algorithm as the quality of the underestimator depends on the variable bounds in a variety of ways. For instance, if a variable participates only linearly in the problem, branching on it will not have any effect on the accuracy of the convex lower bounding functions. On the other hand, reducing the range of a variable raised to a high power is likely to result in much tighter underestimating problems. To take advantage of these observations, the implementation of the αBB algorithm offers some choice in the branching strategies. Four alternatives are currently available:

1. Use k-section on all or some of the variables.
2. Use a measure of the quality of each term's underestimator, based on the maximum separation distance between term and underestimator.
3. Use a measure of the quality of each term's underestimator, based on the separation distance at the optimum point.
4. Use a measure of the overall influence of each variable on the quality of the lower bounding problem.

2.1.1 Strategy 1

By default, bisection on all the variables is used. The user can however specify a restricted list of variables on which to branch. The interval to be bisected is determined through the '*least reduced axis*' rule whose application involves the calculation of a 'current range to original range ratio', r_i, for each variable:

$$r_i = \frac{x_i^U - x_i^L}{x_{i,0}^U - x_{i,0}^L} \tag{3}$$

where $x_{i,0}^U$ and $x_{i,0}^L$ are respectively the upper and lower bound on variable x_i at the first node of the branch-and-bound tree, and x_i^U and x_i^L are respectively the upper and lower bound on variable x_i at the current node of the tree. The variable with the largest r_i is selected for branching.

2.1.2 Strategy 2

While the second branching option requires additional computational effort, it results in a significant improvement of the convergence time for difficult problems. A different measure μ is defined for each type of underestimator to facilitate the assessment of the quality of the lower bounding function. The maximum separation distance between the underestimator and the actual term at the optimal solution of the lower bound problem is one possible indicator of the degree of accuracy achieved in the construction of the convex approximation. For a bilinear term xy, the maximum separation distance was derived by Androulakis *et al.* (1995) so that μ_b is

$$\mu_b = \frac{(x^U - x^L)(y^U - y^L)}{4}. \tag{4}$$

For a fractional term, the maximum separation distance is derived in Adjiman *et al.* (1998).

For a univariate concave term $ut(x)$, the maximum separation distance can be expressed as an optimization problem so that μ_u is given by

$$\mu_u = - \min_{x^L \leq x \leq x^U} -ut(x) + ut^L(x) \tag{5}$$

where $ut^L(x)$ is the linearization of $ut(x)$ around x^L. Since $ut(x)$ is concave and $ut^L(x)$ is linear, the above optimization problem is convex and can therefore be solved to global optimality.

For a general nonconvex term, the maximum separation distance was derived by Maranas and Floudas (1994) and the term measure is

$$\mu_\alpha = \frac{1}{4} \sum_i \alpha_i (x_i^U - x_i^L)^2 \tag{6}$$

where the α_i's are defined by Equation (2).

Given a node to be partitioned, the values of μ_b, μ_u and μ_α are calculated for each term in accordance with its type. The term which appears to have the worst underestimator or, in other words, the largest μ, is then used as a basis for the selection of the branching variable. Out of the set of variables that participate in that term, the one with the '*least reduced axis*', as defined by Equation (3), is chosen for k-section. With this strategy, the influence of variable bounds on the quality of the underestimators is taken into account, and hence this adaptive branching scheme ensures the effective tightening of the lower bounding problem from iteration to iteration.

2.1.3 Strategy 3

This strategy is a variant of Strategy 2. Instead of computing the maximum separation distance between a term and its underestimator, their separation distance at the *optimum solution* of the lower bounding problem is used. Thus, the bilinear term measure is now

$$\mu_b = |x^* y^* - w^*| \tag{7}$$

where w is the variable which has been substituted for the bilinear term xy in order to construct its convex envelope (Adjiman *et al.*, 1997) and the * superscript denotes the value of the variable at the solution of the current lower bounding problem. The measures μ_t, μ_f and μ_{ft}, for trilinear, fractional and fractional trilinear terms, are obtained in a similar fashion.
The measure for univariate term $ut(x)$ is

$$\mu_u = ut(x^*) - ut^L(x^*). \tag{8}$$

Finally, the general nonconvex term measure is

$$\mu_\alpha = \sum_{i=1}^{n} \alpha_i (x_i^U - x_i^*)(x_i^* - x_i^L). \tag{9}$$

2.1.4 Strategy 4

The fourth branching procedure takes the approach of Strategy 3 one step further by considering the overall influence of each variable on the convex problem. After the relevant measures μ_b, μ_t, μ_f, μ_{ft}, μ_u and μ_α have been calculated for every term, a measure μ_v of each variable's contribution may be obtained as follows:

$$\mu_v^i = \sum_{j \in B_i} \mu_b^j + \sum_{j \in T_i} \mu_t^j + \sum_{j \in F_i} \mu_f^j + \sum_{j \in FT_i} \mu_{ft}^j + \sum_{j \in U_i} \mu_u^j + \sum_{j \in N_i} \mu_\alpha^j \tag{10}$$

where μ_v^i is the measure for the ith variable; B_i is the index set of the bilinear terms in which the ith variable participates, μ_b^j is the measure of the jth bilinear term; T_i is the index set of the trilinear terms in which the ith variable participates, μ_t^j is the measure of the jth trilinear term; F_i is the index set of the fractional terms in which the ith variable participates, μ_f^j is the measure of the jth fractional term; FT_i is the index set of the fractional trilinear terms in which the ith variable participates, μ_{ft}^j is the measure of the jth fractional trilinear term; U_i is the index set of the univariate concave terms in which the ith variable participates, μ_u^j is the measure of the jth univariate concave term; N_i is the index set of the general nonconvex terms in which the ith variable participates, μ_α^j is the measure of the jth general nonconvex term.

The variable with the largest measure μ_v is selected as the branching variable, and k-section can be performed on it. If two or more variables have the same μ_v, the '*least reduced axis*' test is performed to distinguish them.

As will become apparent in the computational studies, branching strategies 2, 3 and 4 are particularly effective since they take into account the sensitivity of the underestimators to the bounds used for each variable.

2.2 Variable Bound Updates

The quality of the convex lower bounding problem can also be improved by ensuring that the variable bounds are as tight as possible. In the current implementation of the αBB algorithm, variable bound updates can either be performed at the onset of an αBB run or at each iteration.

In both cases, the same procedure is followed in order to construct the bound update problem. Given a solution domain, the convex underestimator for every constraint in the original problem is formulated. The *bound problem* for variable x_i is then expressed as

$$x_i^{L,NEW}/x_i^{U,NEW} = \left\{ \begin{array}{ll} \min\limits_{\boldsymbol{x}} / \max\limits_{\boldsymbol{x}} & x_i \\ s.t. & \mathcal{G}(\boldsymbol{x}) \leq 0 \\ & \boldsymbol{x}^L \leq \boldsymbol{x} \leq \boldsymbol{x}^U \end{array} \right. \tag{11}$$

where $\mathcal{G}(\boldsymbol{x})$ are the convex underestimators of the constraints, and the bounds on the variables, $\boldsymbol{x}^L$ and $\boldsymbol{x}^U$ are the best calculated bounds. Thus, once a new lower bound $x_i^{L,NEW}$ on x_i has been computed via a minimization, this value is used in the formulation of the maximization problem for the generation of an upper bound $x_i^{U,NEW}$.

Because of the computational expense incurred by an update of the bounds on all variables, it is often desirable to define a smaller subset of the variables on which this operation is to be performed. The criterion devised for the selection of the branching variables can be used in this instance, since it provides a measure of the sensitivity of the problem to each variable. An option was therefore set up, in which bound updates are carried out only for a fraction of the variables with a non-zero μ_v, as calculated in Equation (10).

2.3 Implementation Issues

The goal of the implementation of the αBB algorithm is to produce a user-friendly global optimization environment which is reliable and flexible. This section describes the choices made in order to meet these objectives.

All the strategies for branching variable selection and variable bound updates presented here, as well as the underestimation techniques proposed in Part I of this paper (Adjiman *et al.*, 1997) have been implemented as part of the αBB package. In addition, an intuitive interface and a user's manual (Adjiman *et al.*, 1997) have been designed.

2.3.1 Problem Input and Pre-Processing

To a large extent, the practical value of a general optimization package is measured by its ability to solve a variety of problems. However, it must also be easily adapted to different problem types so that no substantial transformations are required from the user. The need for the second-order derivatives of the nonconvex functions in the problem raises practical difficulties in terms of the implementation of the algorithm and the format of the input file. Because the problems solved belong to a very broad class for which no structural assumptions may be made *a priori*, these derivatives must either be supplied by the user or generated automatically. The first option would be very cumbersome and would render the use of the αBB algorithm fastidious. In order to generate the derivatives automatically, the functions must be available in a code list format (Rall, 1981) which allows systematic differentiation based on elementary rules. Although the functions could be provided by the user in this codified notation, such an approach would result in counter-intuitive problem input. By designing an interface which accepts input in standard mathematical notation, transforms it into the desired form and automatically generates sparse Jacobian and Hessian matrices, both ease of use and flexibility can be achieved. The front-end parser developed as part of the αBB implementation creates this code list, while giving much freedom of notation to the user. This parser can also be used to input user-provided expressions for the second order derivatives when these cannot be obtained automatically. During the parsing phase of an αBB run, the following tasks are carried out:

- Identify variable, function, and parameter names.
- Define linear, bilinear, trilinear, fractional, fractional trilinear, convex, univariate concave, and general nonconvex terms. This determines what types of underestimators are to be used.
- Build code lists for the functions.
- Define bounds on the solution space.
- Gather optional information such as branching strategy, user-defined α values if any, etc.

The choice of all names (variable, function, term, parameter) is left up to the user, with the possibility to use index notation. Many index operations are supported, such as summation, multiplication, enumeration and index arithmetic. The input can therefore be made as explicit or as compact as desired. A sample input file for the CSTR sequence design problem discussed in Section 3.3.1 is listed in Appendix A. The formulation of this problem is given in Appendix B.

Once the parsing phase is completed, a processing stage is initiated in which the code list for each of the lower bounding functions is built using the term information provided in the input file. Sparse Jacobians and Hessian matrices are also generated in the code list format. The sparsity issue is critical not only from the point of view of memory requirements, but also with respect to the diagonal shift matrix computation: the speed of this process is strongly dependent on the number of participating variables for all the methods described in Part I of this paper (Adjiman *et al.*, 1997).

Once all the necessary information has been gathered, the main iteration loop can be started.

2.3.2 User-Specified Underestimators

In some instances, the problem to be solved may involve functions for which no explicit formulation is available. The potential energy function for proteins, for example, is typically calculated using force-field models such as CHARMM (Brooks *et al.*, 1983), AMBER (Weiner *et al.*, 1986) or the ECEPP/3 model (Némethy *et al.*, 1992). The user can require that these be used for function evaluations, as long as the Jacobian and a convex underestimator are also provided.

Some functions whose analytical form is entered in the input file may exhibit a special structure for which a tight underestimator is known. This tailored lower bounding function can then be entered in the input file and replace the generic nonconvex term underestimator of Equation (2).

Similarly, if values or formulae have been derived for the α parameters, this information can be used by the algorithm, thereby improving its performance.

2.3.3 α Calculations

Two main options are available to the user for the determination of the α values to be used in the lower bounding problem.

Option 1: User-specified expressions supplied in the input file for α computations.

Option 2: Calculation using one of the automatic computation methods described in Part I (Adjiman *et al.*, 1997).

The successful implementation of the second option requires the availability of a rigorous procedure for interval function evaluations. The αBB algorithm is therefore connected to a C++ interval arithmetic library, PROFIL/BIAS, developed by Knüppel (1993). All the interval calculations are carried out using the natural extension form of the functions (Ratschek and Rokne, 1988). The main feature of this form is its simplicity but more accurate results can be obtained using the centered form or the remainder form (Cornelius and Lohner, 1984), at increased computational expense. The nested form is particularly suited for the special case of polynomial functions because of the quality of its results and the ease of computation.

Most of the $\mathcal{O}(n^3)$ methods for the computation of a uniform diagonal shift matrix require the calculation of the minimum and/or maximum eigenvalue of real symmetric matrices. These matrices are usually dense and of small size since only the variables that participate in the nonconvex term to be underestimated are taken into consideration when building the Hessian matrix. Some Netlib routines for the transformation of real symmetric matrices into symmetric tridiagonal matrices using orthogonal similarity transformations and the rational QR method with Newton corrections were translated into C for this purpose. The code was also customized based on the specific requirements of the αBB algorithm. For Method I.5, based on the Kharitonov theorem (Kharitonov, 1979), a symbolic expansion of the determinant must be carried out and Leverrier's method (Wayland, 1945) was therefore implemented. Although it is more computationally expensive than Krylov's method or Danielewsky's method (Wayland, 1945), it is the only approach which does not involve divisions by the Hessian elements. This eliminates the risk of encountering singularities when interval arithmetic is subsequently used.

Finally, the error in the α parameter calculations is closely monitored to guarantee the global optimality of the final solution.

3 Computational Case Studies

The purpose of this section is to demonstrate the performance of the αBB algorithm in identifying the global minimum of nonconvex optimization problems and to study the effects of the different underestimating methods of Part I (Adjiman *et al.*, 1997) and the various convergence-enhancing schemes presented in the previous sections. First, some literature problems are tackled, giving insights into the suggested branching and variable range reduction strategies. Larger examples are then employed to test the algorithm. All computational results were obtained on a HP9000/730 with a convergence tolerance of 0.001, unless specified otherwise.

3.1 Literature Problems

The first example is a bilinearly constrained problem which allows to analyze the effects of the different branching strategies. In the second example, a family of concave problems with 20 variables shows that the proposed strategies successfully address both the issue of branching variable selection and variable bound quality. Finally, the third example illustrates the efficiency of the methods presented in Part I of this paper for the calculation of diagonal shift matrices in the case of a nonlinear problem.

3.1.1 Haverly's Pooling Problem

This problem is taken from Floudas and Pardalos (1990).

$$
\begin{aligned}
\max \quad f &= 9x + 15y - 6A - c_1 B - 10(C_x + C_y) \\
s.t. \quad & P_x + P_y - A - B = 0 \\
& x - P_x - C_x = 0 \\
& y - P_y - C_y = 0 \\
& pP_x + 2C_x - 2.5x \leq 0 \\
& pP_y + 2C_y - 1.5y \leq 0 \\
& pP_x + pP_y - 3A - B = 0 \\
& 0 \leq x \leq c_2 \\
& 0 \leq y \leq 200 \\
& 0 \leq A, B, C_x, C_y, p, P_x, P_y \leq 500
\end{aligned}
$$

Three instances of the problem, denoted HP_1, HP_2, and HP_3, have been defined based on the values of the parameters (c_1, c_2). These are (16, 100), (16, 600), and (13, 100) respectively. The nonconvexities are entirely due to the presence of bilinear terms, which can be underestimated using their convex envelope as specified in Part I, or as general nonconvex terms, using calculated α values.

The global optimum solution and the values of the variables that participate in bilinear terms are shown in Table 1 for each of the three cases. The problems were first solved using different options for the variable bound updates and the branching strategy and using the convex envelope for the bilinear terms. The results are presented in Table 2. In addition, results obtained using the GOP algorithm, a global optimization algorithm developed by Visweswaran and Floudas (1996a, 1996b) and designed specifically to handle problems containing bilinear, quadratic and polynomial terms, are listed.

Case	c_1	c_2	p	P_x	P_y	f^*
HP_1	16	100	1.0	0	100	-400
HP_2	16	600	3.0	300	0	-600
HP_3	13	100	1.5	0	200	-750

Table 1: Optimal solutions of the three instances of the Haverly Pooling Problem.

	HP_1			HP_2			HP_3		
Run	N_i	N_n	CPU	N_i	N_n	CPU	N_i	N_n	CPU
A	73	147	3.16	79	159	4.35	161	323	6.33
B	11	23	2.73	9	19	2.42	15	31	3.17
C	5	11	1.22	2	5	0.62	8	17	1.80
D	5	11	1.19	2	5	0.59	8	17	1.84
E	2	5	0.61	2	5	0.61	3	7	0.81
GOP	12	—	0.22	12	—	0.21	14	—	0.26

Table 2: Comparison of computational results using various strategies.
Run A : No variables bounds update, branching strategy 1,
Run B : Variable bounds updates at each iteration, branching strategy 1,
Run C : Variable bounds updates at each iteration, branching strategy 2,
Run D : Variable bounds updates at each iteration, branching strategy 3,
Run E : Variable bounds updates at each iteration, branching strategy 4.
Note N_i is the number of iterations and N_n is the number of expanded nodes.

As far as the αBB algorithm is concerned, it is clear that Run E is the most promising overall strategy. The addition of variable bounds updates results in a significant improvement of the convergence rate as is revealed by a comparison of Runs A and B. Important insight into the workings of the algorithm can be gained by recording the variables on which branching is performed when different strategies are used. In branching strategy 1 (Runs A and B), each variable is branched on equally and the lower bounds remain constant for several iterations as the variables participating linearly have little effect on the tightness of the underestimating problem. In other runs, branching depends on the term measure μ_b, either directly or through the variable measure μ_v. Since the bilinear terms include three variables, p, P_x and P_y, they are the only branching candidates. This leads to another considerable improvement in performance. With branching strategies 2 and 3, each one of the three variables is branched on equally. With branching strategy 4, however, p is branched on exclusively as it participates in all bilinear terms and its μ_v is always greater than that of P_x or P_y. The performance of the branch-and-bound search is therefore enhanced substantially by selecting appropriately the branching variables.

Using the same branching and bounding options as in Run E, the problems were then solved by treating the bilinear terms as general nonconvex terms. For bilinear terms, the Hessian matrix is independent of the variables and no interval calculations are required. The minimum eigenvalue required by all uniform diagonal shift matrix methods is therefore easily obtained. Table 3 compares the performance of the algorithm using the convex envelope ("Linear") and α-based underestimation. The row labeled "Convex I" reports results obtained using a uniform diagonal shift matrix (a single α value per term), and "Convex II" was obtained using the scaled Gerschgorin theorem method which generates one α per variable in a nonconvex term. The use of α leads to looser lower bounding functions than the convex envelope. Moreover, it requires the solution of a convex NLP for the generation of a lower bound, whereas a linear program is constructed when the convex envelope is used. As a result, both computation time and number of iterations increase significantly. The exploitation of the special structure of bilinear terms is thus expected to provide the best results in most cases. For very large problems, however, the introduction of additional variables and constraints may become prohibitive and a convex underestimator becomes more appropriate (Harding and Floudas, 1997).

	HP_1			HP_2			HP_3		
Underestimator	N_i	N_n	CPU	N_i	N_n	CPU	N_i	N_n	CPU
Linear	2	5	0.61	2	5	0.61	3	7	0.81
Convex I	24	49	5.66	27	55	7.23	18	37	4.53
Convex II	31	63	6.72	23	47	6.48	14	29	3.60

Table 3: Computational results using different underestimating schemes.

3.1.2 Linearly Constrained Concave Optimization Problems

This set of problems taken from Floudas and Pardalos (1990) will be used to show the increased efficiency of the algorithm when appropriate strategies are being employed to select the variables whose bounds will be updated. The general form of the problems is as follows :

$$\begin{array}{lrcl} \min & f & = & -0.5 \sum_{i=1}^{20} \lambda_i (x_i - \beta_i)^2 \\ s.t. & \multicolumn{3}{l}{x \in P = \{x : Ax \leq b, x \geq 0\} \subset \Re^{20}} \end{array}$$

Five cases are defined for different values of (λ, β) as shown in Table 4. The global minima are also given in Table 4, while the values of the variables at the optimal solutions are given in Table 5.

	C_1	C_2	C_3	C_4	C_5
λ_i	1	1	20	1	i
β_i	2	-5	0	8	2
f^*	-394.7506	-884.7508	-8695.011	-754.7506	-4150.410

Table 4: Parameter and optimal objective function values for the five instances of the linearly constrained concave problems ($i = 1, \cdots, 20$).

Case	x_3	x_6	x_{11}	x_{13}	x_{15}	x_{16}	x_{18}	x_{20}
C_1	28.80	4.179	0	0	0.619	4.093	2.306	0
C_2	28.80	4.179	0	0	0.619	4.093	2.306	0
C_3	28.80	4.179	0	0	0.619	4.093	2.306	0
C_4	28.80	4.179	0	0	0.619	4.093	2.306	0
C_5	1.043	0	1.747	0.431	0	4.433	15.858	16.487

Table 5: Optimal solutions of the five instances of the linearly constrained concave problems – Variables not shown here are equal to zero.

Since this problem involves only linear and univariate concave terms, linear underestimators may be used to construct the lower bounding problem.

The computational requirements for the different strategies are summarized in Table 6. Results obtained with the GOP algorithm are also provided.

A significant reduction of the number of iterations as well as the CPU time requirements is achieved when an appropriate branching procedure is used (Runs R3 and R4). Such a scheme repeatedly selects a few key variables for branching from the entire pool of 20 variables. When variable bounds updates are performed at every iteration (Runs R2 and R4), only a few variables bounds are actually optimized. This results in a significant decrease in the required number of iterations. However the overall CPU time increases by as much as a factor of 2. It was found that the construction of the lower bounding problem in each domain represented on the order of 0.1% of the overall computational expense when bound updates were performed at each iteration, and 1% when bound updates took place at the first iteration only.

	Iterations					CPU				
Case	R1	R2	R3	R4	GOP	R1	R2	R3	R4	GOP
C_1	137	72	42	26	27	13.2	23.2	4.8	9.4	0.68
C_2	116	71	38	23	4	11.6	22.9	4.5	8.2	3.57
C_3	123	77	41	26	11	11.3	24.2	4.6	8.5	10.91
C_4	115	79	38	24	5	11.9	25.2	4.4	8.3	5.07
C_5	544	221	134	63	229	51.0	67.0	12.0	19.9	177.04

Table 6: Computational requirements for the five instances of the linearly constrained concave problems.

R1 : Branching strategy 1; Bounds updates at first iteration,
R2 : Branching strategy 1; Bounds updates at every iteration,
R3 : Branching strategy 3; Bounds updates at first iteration,
R4 : Branching strategy 3; Bounds updates at every iteration,
GOP : GOP algorithm.

When the univariate concave terms are treated as general nonconvex terms and α values are calculated for each one of them, the computational requirements for convergence to the global optimum solution are almost unchanged, as can be seen in Table 7. Once again, the generation of the convex underestimators makes up about 1% of the computational cost.

	C1		C2		C3		C4		C5	
Underes.	N_i	CPU	N_i	CPU	N_i	CPU	N_i	CPU	N_i	CPU
Linear	42	4.8	38	4.5	41	4.6	38	4.4	134	12.0
Convex	42	4.2	38	3.8	41	3.8	38	3.9	134	9.3

Table 7: Computational results using different underestimators for the linearly constrained concave problems. Strategy R3 was used for all runs.

3.1.3 Constrained Nonlinear Optimization Example

This is a small but nevertheless difficult nonconvex problem from Murtagh and Saunders (1983). The αBB algorithm is very well suited for this example since it combines bilinear, univariate concave and general nonconvex terms.

$$\min \quad (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^3 + (x_3 - x_4)^4 + (x_4 - x_5)^4$$

s.t.

$$\begin{aligned} x_1 + x_2^2 + x_3^3 &= 3\sqrt{2} + 2 \\ x_2 - x_3^2 + x_4 &= 2\sqrt{2} - 2 \\ x_1 x_5 &= 2 \\ x_i &\in [-5, 5], \; i = 1, \cdots, 5 \end{aligned}$$

The global minimum as well as some local minima are presented in Table 8.

	obj	x_1	x_2	x_3	x_4	x_5
Global	0.0293	1.1166	1.2204	1.5378	1.9728	1.7911
Local 1	27.8719	-1.2731	2.4104	1.1949	-0.1542	-1.5710
Local 2	44.0221	-0.7034	2.6357	-0.0963	-1.7980	-2.8434
Local 3	52.9026	0.7280	-2.2452	0.7795	3.6813	2.7472
Local 4	64.8740	4.5695	-1.2522	0.4718	2.3032	4.3770

Table 8: Global and local minima of example 3.1.3.

All the α calculation methods proposed in Part I of this paper have been used to solve this problem to global optimality. Strategies 2, 3 or 4 were used for branching variable selection and bound updates were performed on a subset of the variables at every iteration for all runs. All branching strategies were tried for this problem and the most successful run is reported in Table 9.

The percentage of total time spent generating convex underestimators, t_U, varies from method to method. For Method I.5, the derivation of intervals for the coefficients of the characteristic polynomial and the solution of the real polynomials is quite time-consuming. In the case of Method II.3, the increase in computational expense is due to the time spent solving semi-definite programs. For all other methods, a low percentage of the overall CPU time (about 2.5%) is needed to derive the lower bounding problem.

The best results are given by the E-matrix approach, the Hertz method and the scaled Gerschgorin approach with $d_i = (x_i^U - x_i^L)$. The minimization of the maximum separation distance approach converges with the smallest number of iterations but performs poorly in terms of CPU requirements. Branching strategy 3 is especially well-suited for this problem. If the variable bounds are updated only once at the beginning of the αBB run, the scaled Gerschgorin method requires 2026 iterations and 224.2 CPU seconds. The relatively high degree of nonlinearity of this example problem increases the dependence of the quality of the lower bounding problem on the variable bounds: while the α values were constant for the Haverly pooling problems and the linearly constrained concave problems, they are now computed through interval arithmetic. Variable bound updates should therefore be carried out preferentially on the most nonlinear variables.

For all the methods used, the quality of the lower bound increases sharply from iteration to iteration at the beginning of a run. After a few levels of the branch-and-bound tree have been explored, the rate of improvement tapers off. Figure 2 shows the progress of the lower bound for the first 100 iterations using Method II.2 and one can clearly see the dramatic amelioration in the quality of the underestimator.

The shape of the branch-and-bound tree also reflects the efficiency of the underestimating techniques used in the αBB algorithm: when using the Hertz method, for example, all nodes on the first five levels are visited. Fathoming becomes significant at subsequent levels as the lower bound approaches the optimum solution. Level 6 is 95% full, level 7 is 69% full, level 8 is 36% full and level 9 is only 25% full. Thereafter, the absolute number of visited nodes on each level actually decreases as the algorithm zeroes in on the region containing the solution and time is spent closing the small gap between the lower and upper bounds.

Method		Iter.	CPU sec.	t_U (%)	Branch
Gerschgorin (I.1)		353	76.3	2.1	4
E-Matrix (I.2)	$E = 0$	337	70.7	2.8	3
E-Matrix (I.2)	$E = \mathrm{diag}(\Delta H)$	339	72.9	2.3	2
Mori-Kokame (I.3)		611	137.5	2.3	3
Lower bounding Hessian (I.4)		346	72.8	2.4	3
Kharitonov (I.5)		412	115.9	11.1	4
Hertz (I.6)		334	72.2	2.9	3
Scaled Ger. (II.1)	$d_i = 1$	352	76.1	2.0	4
Scaled Ger. (II.1)	$d_i = (x_i^U - x_i^L)$	330	69.7	2.0	3
H-Matrix (II.2)		347	79.6	3.1	3
Min. Max. distance (II.3)		318	100.4	30.3	4

Table 9: Results for example 3.1.3. t_U denotes the percentage of total CPU time spent generating convex underestimators.

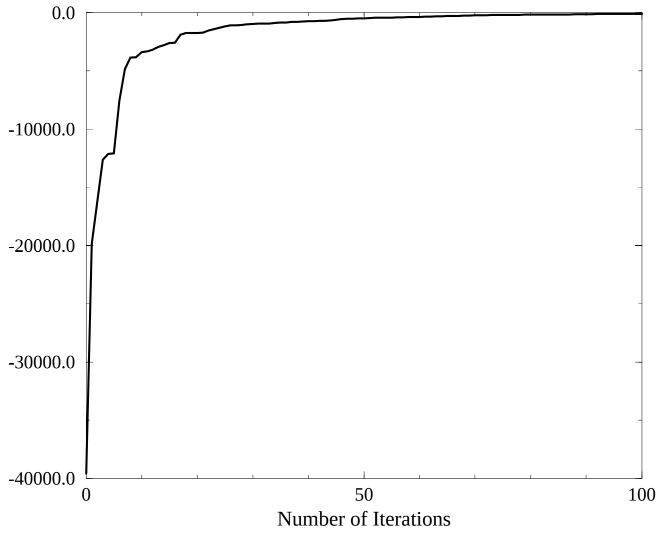

Figure 2: Progress of the lower bound for Example 3.1.3.

3.2 Chemical Engineering Design Problems

Having established the importance of branching variable selection and variable bound updates, some results will now be presented for a set of optimization formulations for chemical process design problems. The first problem is a small reactor network design problem, the second example is a heat exchanger network design problem and the third is a separation network design problem.

3.2.1 Reactor Network Design

The following example, taken from Ryoo and Sahinidis (1995), is a reactor network design problem, describing the system shown in Figure 3.

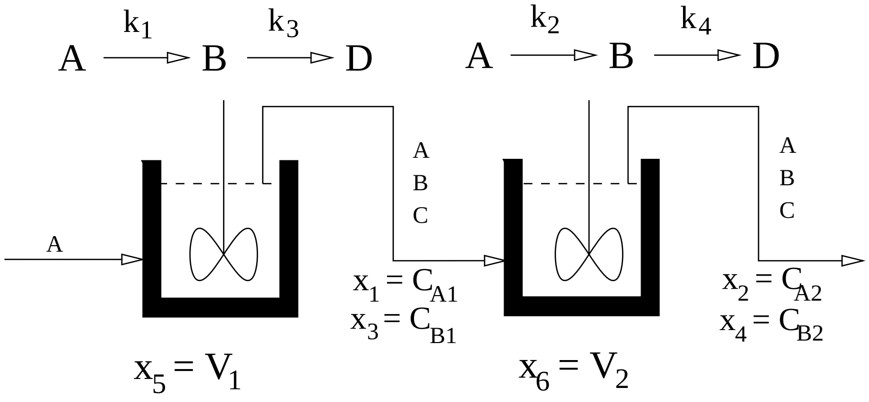

Figure 3: Reactor Network Design Problem.

This problem is known to have caused difficulties for other global optimization methods.

$$\min -x_4$$

s.t.

$$\begin{aligned} x_1 + k_1 x_1 x_5 &= 1 \\ x_2 - x_1 + k_2 x_2 x_6 &= 0 \\ x_3 + x_1 + k_3 x_3 x_5 &= 1 \\ x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 &= 0 \end{aligned}$$

$$
\begin{aligned}
x_5^{0.5} + x_6^{0.5} &\leq 4 \\
k_1 &= 0.09755988 \\
k_2 &= 0.99 k_1 \\
k_3 &= 0.0391908 \\
k_4 &= 0.9 \\
(0, 0, 0, 10^{-5}, 10^{-5}) \leq (x_1, x_2, x_3, x_4, x_5) &\leq (1, 1, 1, 1, 16, 16)
\end{aligned}
$$

A number of close local minima exist, as shown in Table 10.

	obj	x_1	x_2	x_3	x_4	x_5	x_6
Global	-0.38881	0.772	0.517	0.204	0.388	3.036	5.097
Local 1	-0.38808	1.0	0.393	0.0	0.388	10^{-5}	15.975
Local 2	-0.37461	0.391	0.391	0.375	0.375	15.975	10^{-5}

Table 10: Global and local minima of the small reactor network design problem.

If the univariate concave and bilinear terms are underestimated linearly, the αBB algorithm identifies the global minimum after 33 iterations and 5.5 CPU seconds. When the bilinear terms are treated as general nonconvex terms, the computational requirements increase to 157 iterations and 43.4 CPU seconds. In both situations, the generation of convex underestimators requires about 0.4% of the overall CPU time.

3.2.2 Heat Exchanger Network Design

The following problem, from Floudas and Pardalos (1990), addresses the design of a heat exchanger network as shown in Figure 4.

$$\min \quad f = x_1 + x_2 + x_3$$

s.t.

$$
\begin{aligned}
0.0025(x_4 + x_6) - 1 &\leq 0 \\
0.0025(-x_4 + x_5 + x_7) - 1 &\leq 0 \\
0.01(-x_5 + x_8) - 1 &\leq 0 \\
100x_1 - x_1 x_6 + 833.33252 x_4 - 83333.333 &\leq 0
\end{aligned}
$$

$$
\begin{aligned}
x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 &\leq 0 \\
x_3x_5 - x_3x_8 - 2500x_5 + 1250000 &\leq 0 \\
100 \leq x_1 &\leq 10000 \\
1000 \leq x_2, x_3 &\leq 10000 \\
10 \leq x_4, x_5, x_6, x_7, x_8 &\leq 1000
\end{aligned}
$$

The optimal value of the objective function is $f^* = 7049.25$, while the optimal variable vector is

$$\boldsymbol{x}^* = (579.19, 1360.13, 5109.92, 182.01, 295.60, 217.99, 286.40, 395.60)^T.$$

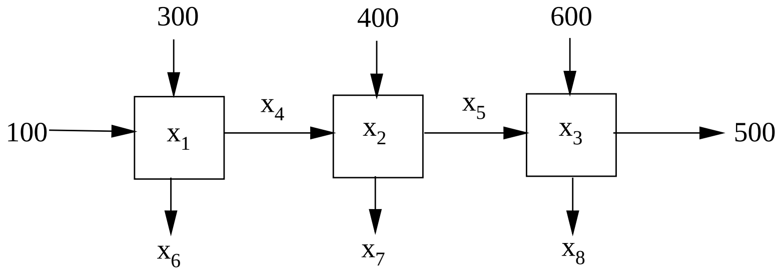

Figure 4: Heat Exchanger Network Design Problem.

It takes the αBB algorithm 244 iterations and 54.4 seconds to identify the global minimum when linear underestimators are used. With convex underestimators, 558 iterations and 141.4 CPU seconds are required.

3.2.3 Separation network synthesis

In this problem taken from Floudas and Aggarwal (1990), the superstructure for the separation of a three-component mixture into two products of different compositions is considered. Two separators and a number of splitters and mixers are available to complete the task, as shown in Figure 5. The formulation of the cost minimization problem given in Floudas and Aggarwal (1990) has been slightly modified to reduce the number of variables. It now involves 22 variables and 16 equality constraints. It is expressed as

$$\min 0.9979 + 0.00432F_1 + 0.00432F_{13} + 0.01517F_2 + 0.01517F_9$$

s.t

$$\begin{aligned}
F_1 + F_2 + F_3 + F_4 &= 300 \\
F_5 - F_6 - F_7 &= 0 \\
F_8 - F_9 - F_{10} - F_{11} &= 0 \\
F_{12} - F_{13} - F_{14} - F_{15} &= 0 \\
F_{16} - F_{17} - F_{18} &= 0 \\
F_{13}x_{A,12} - F_5 + 0.333 * F_1 &= 0 \\
F_{13}x_{B,12} - F_8 x_{B,8} + 0.333 * F_1 &= 0 \\
-F_8 x_{C,8} + 0.333F_1 &= 0 \\
-F_{12}x_{A,12} - 0.333F_2 &= 0 \\
F_9 x_{B,8} - F_1 2xB, 12 + 0.333F_2 &= 0 \\
F_9 x_{C,8} - F_{16} + 0.333F_2 &= 0 \\
F_{14}x_{A,12} + 0.333F_3 + F_6 &= 30 \\
F_{10}x_{B,8} + F_{14}x_{B,12} + 0.333F_3 &= 50 \\
F_{10}x_{C,8} + 0.333F_3 + F_{17} &= 30 \\
x_{B,8} + x_{C,8} &= 1 \\
x_{A,12} + x_{B,12} &= 1 \\
0 \leq F_i &\leq 150 \\
0 \leq x_{i,j} &\leq 1
\end{aligned}$$

where F_i denotes the total flowrate of the ith stream and $x_{j,i}$ denotes the fraction of component j in the ith stream.

The global optimal configuration is identified in Figure 6. The optimal concentrations are $x_{B,8} = x_{C,8} = 0.5$, $x_{A,12} = 0$ and $x_{B,12} = 1$. The optimal non-zero flowrates are $F_1 = 60$, $F_3 = 90$, $F_4 = 150$, $F_5 = F_7 = 20$, $F_8 = F_9 = 40$, $F_{12} = F_{14} = 20$ and $F_{16} = F_{18} = 20$.

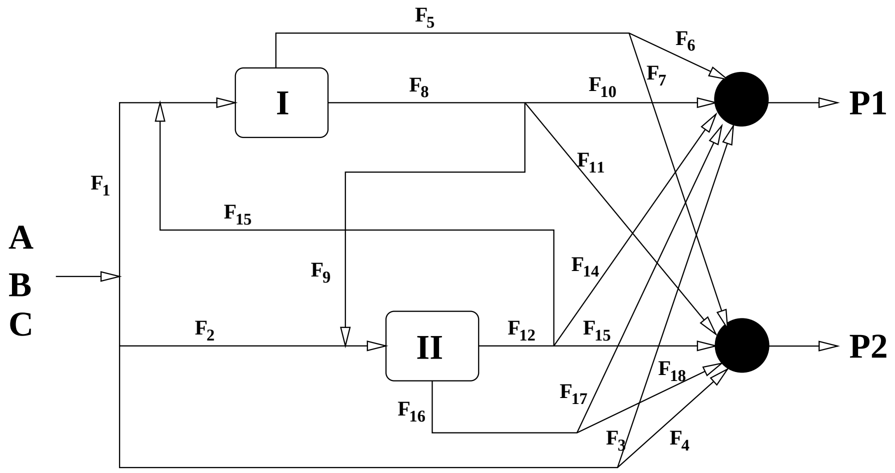

Figure 5: Superstructure for example 3.2.3.

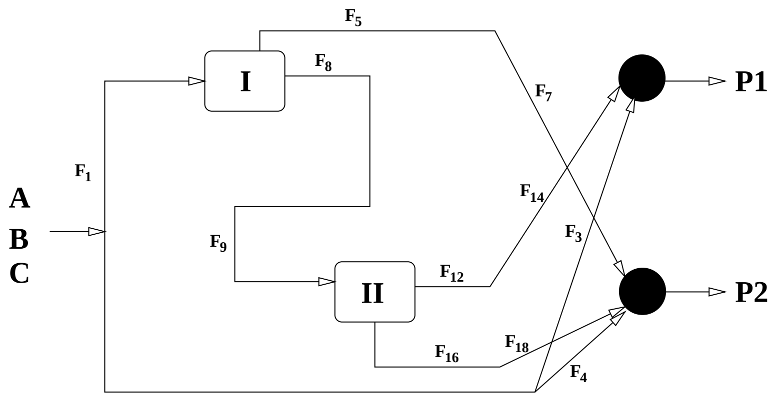

Figure 6: Optimal configuration for example 3.2.3.

The algorithm converged to the global solution in 15.2 CPU seconds and after 11 iterations, when the bilinear terms were underestimated linearly. When they were treated as general nonconvex terms, 49 iterations and 220.1 CPU seconds were required.

3.3 Generalized Geometric Programming Problems

Many important design and control problems can be formulated as generalized geometric programming problems, a subclass of the twice-differentiable problems that the αBB algorithm can address. The main property of the functions involved in the formulation is that they are the algebraic sum of posynomials, terms of the form $c \prod_{i=1}^{n} x_i^{d_i}$ where c is a positive real number, the x_i's are positive real variable and the d_i's are scalars. No restrictions of integrality or positivity are imposed on the exponents. Maranas and Floudas (1997) proposed a new approach to tackle such problems, based on a difference of convex functions transformation embedded in a branch-and-bound framework. In this section, we show that the αBB can be successfully used on this class of problems and we report results for the example problems treated in Maranas and Floudas (1997).

Two design problems were studied in the area of chemical process engineering: an alkylation design problem and the design of a CSTR sequence subject to some capital cost constraints. The objective was to minimize cost or to maximize production. The aim of the six control problems was to carry out the stability analysis of some nonlinear systems with uncertain parameters. These examples were therefore formulated as the minimization of the stability margin, k, and the system was deemed unstable if the optimal solution was found to be less than unity. Two approaches can be used to treat such problems, the first relying on the identification of the smallest possible value of k, and the second testing the feasibility of the problem with $k \in [0, 1]$. The alkylation problem was discussed in detail in Part I of this paper. All other problem formulations are given in Appendix B.

3.3.1 CSTR Sequence Design

Following the reformulation proposed in Maranas and Floudas (1997) for example 3.2.1, the objective function is expressed as a ratio of polynomials and is subject to a single nonlinear constraint. The reactor volumes are the only two variables. The problem is solved for two sets of reaction constants.

Since there is only one constraint, which relates the two reactor volumes, variable bound updates are very fast. Yet, as is apparent in Table 11, the reduction in overall iteration number achieved when bound updates are performed at each iteration results in an increase in the computational requirements. This example and the alkylation problem (Adjiman *et al.*, 1997) seem to indicate that problems belonging to the class of generalized geometric programming can be treated using a single bound update before the first iteration.

For this problem the use of a single α per term or of one α per variable does not have much influence on the performance of the algorithm. Upon examining the functions in the problem, it appears that the variables participate in very similar ways, therefore contributing to the nonconvexity to the same degree.

Finally, the performance of the algorithm for this highly nonlinear formulation is better, in terms of CPU time, than for the simpler formulation of example 3.2.1 which involves only bilinear terms. It therefore seems worthwhile to reduce the number of variables, even at the expense of functional simplicity.

3.3.2 Stability Analysis of Nonlinear Systems

Example 1 This problem involves 4 variables and 4 constraints, one of which is nonlinear. The system is found to be unstable with $k = 0.3417$. The results shown in Table 12 were obtained with the third branching strategy, using a single bound update. They correspond to the solution of the global optimality problem, in which wide bounds on k are used to try and identify the smallest possible value. If the actual value of the stability margin is of no interest, the feasibility problem can be solved, in which the value of k is restricted to $[0, 1]$. In such a case, the algorithm can terminate when an upper bound on k has been found within this interval (the system is unstable), or when infeasibility has been asserted (the system is stable). For this example, an upper bound below 1 is obtained in less than 1 CPU sec.

Example 2 Contrary to the previous example, this 4 variable system is stable, with a stability margin of $k = 1.089$. In order to prove that the system is stable, one must show that the feasibility problem has no solution. Alternatively, the minimum stability margin can be identified using wider bounds on k. Clearly, global optimality runs are more time demanding as they span a larger solution space. The results are shown in Table 13.

Example 3 The global optimum solution of $k = 0.8175$ is identified using the fourth branching strategy and a single bound update. The fact that the

First set of reaction rate constants							
Method		Single Update			One Update/Iter		
		Iter.	CPU sec.	t_U (%)	Iter.	CPU sec.	t_U (%)
Gerschgorin (I.1)		101	4.5	28.9	67	7.1	15.8
E-Matrix (I.2)	$E = 0$	94	4.4	28.6	67	7.0	16.3
E-Matrix (I.2)	$E = \text{diag}(\Delta H)$	105	4.6	30.2	68	7.2	17.0
Mori-Kokame (I.3)		137	6.1	39.1	89	8.9	17.9
Lower bounding Hessian (I.4)		94	4.3	31.6	67	6.7	16.5
Kharitonov (I.5)		123	21.3	66.6	81	24.4	51.7
Hertz (I.6)		94	4.4	41.4	67	6.8	21.2
Scaled G. (II.1)	$d_i = 1$	101	4.5	27.8	66	6.6	20.2
Scaled G. (II.1)	$d_i = (x_i^U - x_i^L)$	92	4.2	21.9	63	6.5	15.3
H-Matrix (II.2)		94	4.2	16.9	65	6.7	17.7
Min. Max. distance (II.3)		93	6.3	40.1	62	8.0	29.1

Second set of reaction rate constants							
Method		Single Update			One Update/Iter		
		Iter.	CPU sec.	t_U (%)	Iter.	CPU sec.	t_U (%)
Gerschgorin (I.1)		111	4.9	24.1	75	7.7	18.1
E-Matrix (I.2)	$E = 0$	109	4.7	29.0	67	7.0	18.2
E-Matrix (I.2)	$E = \text{diag}(\Delta H)$	114	5.1	24.1	79	8.0	21.1
Mori-Kokame (I.3)		150	6.4	25.9	95	9.8	17.8
Lower bounding Hessian (I.4)		109	4.7	26.8	74	7.5	15.2
Kharitonov (I.5)		135	15.6	61.5	84	16.8	46.7
Hertz (I.6)		109	4.7	23.0	74	7.5	17.9
Scaled G. (II.1)	$d_i = 1$	109	4.6	26.2	74	7.5	18.9
Scaled G. (II.1)	$d_i = (x_i^U - x_i^L)$	105	4.6	24.2	72	7.3	17.8
H-Matrix (II.2)		109	4.6	17.7	74	7.6	15.1
Min. Max. distance (II.3)		107	6.8	43.4	73	9.2	30.0

Table 11: Results for Example 3.3.1. t_U denotes the percentage of total CPU time spent generating convex underestimators.

Method		Iter.	CPU sec.	t_U (%)
Gerschgorin (I.1)		24	1.4	26.5
E-Matrix (I.2)	$E = 0$	20	1.4	28.9
E-Matrix (I.2)	$E = \text{diag}(\Delta H)$	20	1.3	15.4
Mori-Kokame (I.3)		20	1.2	23.1
Lower bounding Hessian (I.4)		20	1.4	16.2
Kharitonov (I.5)		20	3.3	70.5
Hertz (I.6)		20	1.4	20.5
Scaled Ger. (II.1)	$d_i = 1$	22	1.6	11.1
Scaled Ger. (II.1)	$d_i = (x_i^U - x_i^L)$	21	1.3	15.2
H-Matrix (II.2)		21	1.4	19.3
Min. Max. distance (II.3)		18	8.6	85.0

Table 12: Results for Example 1 of Section 3.3.2. t_U denotes the percentage of total CPU time spent generating convex underestimators.

Method		Global Optimality			Feasibility		
		Iter.	CPU sec.	t_U (%)	Iter.	CPU sec.	t_U (%)
Gerschgorin (I.1)		82	3.0	3.5	31	1.1	7.9
E-Matrix (I.2)	$E = 0$	77	2.8	5.5	29	1.8	2.5
E-Matrix (I.2)	$E = \text{diag}(\Delta H)$	74	2.6	4.5	29	1.2	2.1
Mori-Kokame (I.3)		98	1.3	3.6	34	1.3	7.3
Lower bounding Hessian (I.4)		77	2.8	5.5	29	1.2	2.7
Kharitonov (I.5)		89	5.4	57.6	32	2.1	5.5
Hertz (I.6)		77	2.3	4.7	29	1.2	11.1
Scaled G. (II.1)	$d_i = 1$	43	1.5	1.7	13	0.8	5.6
Scaled G. (II.1)	$d_i = (x_i^U - x_i^L)$	35	1.2	2.0	13	0.7	5.6
H-Matrix (II.2)		76	3.3	5.2	29	1.6	2.8
Min. Max. distance (II.3)		34	5.1	86.1	11	2.9	75.1

Table 13: Results for Example 2 of Section 3.3.2. t_U denotes the percentage of total CPU time spent generating convex underestimators.

system is unstable is identified in under 1 CPU sec. The results of the global optimality problem are shown in Table 14.

Method		Iter.	CPU sec.	t_U (%)
Gerschgorin (I.1)		20	1.0	4.3
E-Matrix (I.2)	$E = 0$	18	0.9	4.2
E-Matrix (I.2)	$E = \mathrm{diag}(\Delta H)$	17	0.9	3.7
Mori-Kokame (I.3)		17	0.8	4.2
Lower bounding Hessian (I.4)		20	1.0	4.0
Kharitonov (I.5)		18	1.1	31.0
Hertz (I.6)		18	0.8	4.2
Scaled Ger. (II.1)	$d_i = 1$	17	0.8	4.2
Scaled Ger. (II.1)	$d_i = (x_i^U - x_i^L)$	12	0.6	5.3
H-Matrix (II.2)		17	0.9	4.4
Min. Max. distance (II.3)		26	2.1	37.6

Table 14: Results for Example 3 of Section 3.3.2. t_U denotes the percentage of total CPU time spent generating convex underestimators.

Example 4 With a minimum stability margin of 6.27, this system is stable. The perturbation frequency ω, which was eliminated from previous formulations, is kept in this example. At the global optimum solution, the value of ω is 0.986. The bounds used for this variable have a significant effect on the convergence rate of the algorithm. To illustrate this point, the feasibility problem was solved with $\omega \in [0, 1]$ and $\omega \in [0, 10]$. The fourth branching strategy was used and a single variable bound update was performed. The results are shown in Table 15.

Example 5 This formulation is used to determine the stability of Daimler Benz bus. It involves 4 variables and the exponent values range from 1 to 8. The minimum stability margin is $k = 1.2069$ and the problem is treated as a feasibility problem. The results are shown in Table 16.

Example 6 This final problem was developed to study the stability of the Fiat Dedra spark ignition engine. It involves 9 variables and is highly nonlinear. The solution of the stability problem shows that this system is

Method		$\omega \in [0,1]$			$\omega \in [0,10]$		
		Iter.	CPU sec.	t_U (%)	Iter.	CPU sec.	t_U (%)
Gerschgorin (I.1)		11	1.7	22.6	32	4.1	18.4
E-Matrix (I.2)	$E = 0$	9	1.2	22.7	29	3.2	13.8
E-Matrix (I.2)	$E = \mathrm{diag}(\Delta H)$	9	1.3	20.5	29	3.2	23.9
Mori-Kokame (I.3)		12	1.7	24.5	34	4.2	23.7
Lower bounding Hessian (I.4)		5	1.0	30.0	25	2.9	21.0
Kharitonov (I.5)		4	20.7	98.1	23	63.1	98.3
Hertz (I.6)		8	1.7	41.8	29	4.6	46.7
Scaled G. (II.1)	$d_i = 1$	1	0.6	18.8	8	1.3	21.2
Scaled G. (II.1)	$d_i = (x_i^U - x_i^L)$	1	0.4	14.2	5	0.8	21.2
H-Matrix (II.2)		9	1.3	16.1	28	3.4	15.6
Min. Max. distance (II.3)		1	2.6	91.2	8	16.5	58.4

Table 15: Results for Example 4 of Section 3.3.2. t_U denotes the percentage of total CPU time spent generating convex underestimators.

Method		$\omega \in [0,1]$			$\omega \in [0,10]$		
		Iter.	CPU sec.	t_U (%)	Iter.	CPU sec.	t_U (%)
Gerschgorin (I.1)		23	3.8	20.8	2997	784	18.9
E-Matrix (I.2)	$E = 0$	18	2.9	23.1	2559	392	22.6
E-Matrix (I.2)	$E = \mathrm{diag}(\Delta H)$	20	3.4	26.3	2563	351	20.4
Mori-Kokame (I.3)		35	5.5	21.7	5328	1874	14.4
Lower bounding Hessian (I.4)		18	2.7	21.1	2695	111	22.1
Kharitonov (I.5)		38	20.3	79.6	2872	1406	80.7
Hertz (I.6)		16	2.7	24.3	2403	328	22.8
Scaled G. (II.1)	$d_i = 1$	18	2.5	22.4	1013	255	22.4
Scaled G. (II.1)	$d_i = (x_i^U - x_i^L)$	7	1.1	26.7	821	107	13.9
H-Matrix (II.2)		18	3.1	14.0	2457	385	12.4
Min. Max. distance (II.3)		14	5.4	69.8	1011	514	72.5

Table 16: Results for Example 5 of Section 3.3.2. t_U denotes the percentage of total CPU time spent generating convex underestimators.

stable and computational results, obtained with $\omega \in [0, 10]$, are reported in Table 17. Method I.5 failed to solve this problem because the nonconvex terms are unusually complex expressions which depend on many variables. The storage requirements for the analytical expressions of the coefficients of the characteristic polynomials thus become excessive. The complexity of this problem also results in an increase in the time requirements for the underestimator generation, regardless of the method used.

Method		$\omega \in [0, 10]$		
		Iter.	CPU sec.	t_U (%)
Gerschgorin (I.1)		148	42.3	63.4
E-Matrix (I.2)	$E = 0$	140	44.4	64.4
E-Matrix (I.2)	$E = \text{diag}(\Delta H)$	186	52.0	62.9
Mori-Kokame (I.3)		261	74.5	65.7
Lower bounding Hessian (I.4)		130	38.6	60.9
Kharitonov (I.5)		—	—	—
Hertz (I.6)		135	76.9	77.1
Scaled Ger. (II.1)	$d_i = 1$	163	48.9	63.7
Scaled Ger. (II.1)	$d_i = (x_i^U - x_i^L)$	3944	1129.2	64.0
H-Matrix (II.2)		143	45.1	71.3
Min. Max. distance (II.3)		246	1528.4	96.5

Table 17: Results for Example 6 of Section 3.3.2.

3.3.3 Summary of Generalized Geometric Programming Results

The results for the set of generalized geometric programming problems show that no single method for the generation of a diagonal shift matrix systematically outperforms the others. A relative performance index can be computed for each method in terms of CPU time or number of iterations. For a given problem, the worst performance is denoted t^* and N^*_{iter} when considering CPU time and iteration number respectively. The performance indices of method M are then $p_{M,t} = t_M/t^*$ and $p_{M,iter} = N_{M,iter}/N^*_{iter}$. An overall performance index for any method is then obtained by averaging its performance index over all example problems. According to this definition, the performance index is always less than or equal to one, and small performance indices indicate relatively successful methods. The histograms in Figure 7 show the average performance indices for each method. They do not account for the failure of Method I.5 to solve Example 6.

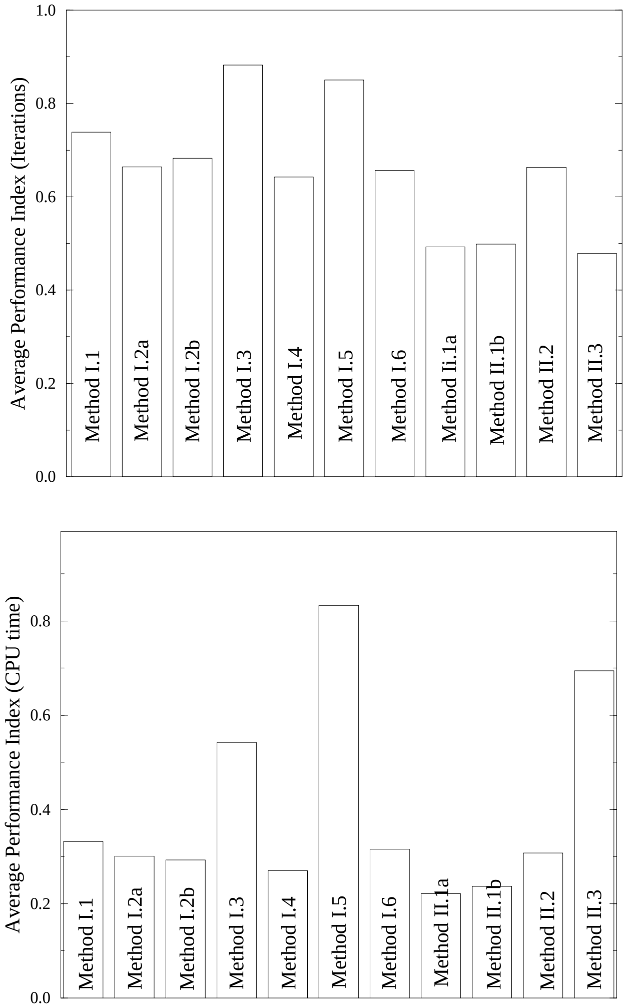

Figure 7: Comparison of α calculation methods for Section 3.3.2 examples

These graphs reveal that the scaled Gerschgorin theorem approach (Method II.1) is on average significantly better than others. At the other end of the spectrum, the Kharitonov theorem approach (Method I.5) and the Mori and Kokame approach (Method I.3) give comparatively poor results. The minimization of maximum separation distance approach (Method II.3) is unique in that it performs well in terms of number of iterations but does poorly in terms of CPU time. The additional time requirement arises from the need to solve a semi-definite programming problem for each nonconvex term at each node. Thus t_U, the percentage of time spent generating convex underestimators, is much larger for Method II.3 than for all other approaches, with the exception of Method I.5.

3.4 Batch Process Design and Scheduling under Uncertainty

This class of problems addresses the design of multiproduct batch plants given uncertain demands and processing parameters. The aim of the problem is the identification of the equipment sizes, batch sizes and production rates which maximize profit, given a fixed number of stages. The uncertainties are taken into account in two ways: the demands are assigned probability distributions, while the size factors and the processing times are assigned a set of discrete values. Each of the P potential combinations of these values gives rise to a different scenario and optimization must be carried out over all P possibilities. The derivation of the NLP formulation used to represent batch design problems is presented in detail by Harding and Floudas (1997). Its final form is given by

$$
\begin{aligned}
\min_{\boldsymbol{v},\boldsymbol{b},\boldsymbol{Q}} \quad & \delta \sum_{j=1}^{M} \alpha_j N_j \exp(\beta_j v_j) \\
& - \sum_{p=1}^{P} \frac{1}{\omega^p} \sum_{q=1}^{Q} \omega^q J^q \sum_{i=1}^{N} p_i Q_i^{qp} \\
\text{s.t.} \quad & v_j \; - \; b_i \; \geq \; \ln(S_{ij}^p) \qquad \forall\, i, \forall\, j, \forall\, p \qquad\qquad (12) \\
& \sum_{i=1}^{N} Q_i^{qp} \cdot \exp(t_{Li}^p - b_i) \; \leq \; H \qquad \forall\, q, \forall\, p \\
& \theta_i^L \; \leq \; Q_i^{qp} \; \leq \; \theta_i^q \qquad \forall\, i, \forall\, q, \forall\, p \\
& \ln(V_j^L) \; \leq \; v_j \; \leq \; \ln(V_j^U)
\end{aligned}
$$

where the parameters are defined as follows: δ is the coefficient used for capital cost annualization, M denotes the number of stages, N is the number of products, N_j is the number of identical pieces of equipment in stage j, α_j and β_j are the fixed charge cost coefficients for the equipment in stage j, p_i is the market price of product i, ω^p is the weighting factor for scenario p, S_{ij}^p is the equipment volume used to produce one mass unit of product i in stage j and for scenario p, t_{Li}^p is the natural logarithm of the longest processing time for product i over all stages in scenario p, H is the time horizon for the campaign, θ_i^L is the lower bound for the uncertain demand for product i, V_j^L and V_j^U are the lower and upper bounds on the equipment size for stage j. The second term in the objective function is an approximation of the expected revenue based on a Gaussian quadrature formula. Q is the total number of quadrature points and J^q is the probability of quadrature point q, ω^q is its weighting factor, θ_i^q is the upper bound on the uncertain demand of product i associated with quadrature point q in scenario p. The variables are Q_i^{pq}, the amount of product i produced in scenario p and associated with quadrature point q, v_j, the natural logarithm of the equipment size for stage j, and b_i, the natural logarithm of the size of the batch for product i.

The size of the problem increases dramatically with the number of scenarios, stages, products and quadrature points. The presence in the constraints of nonconvex terms involving many of the variables renders these problems difficult.

3.4.1 Example 1

This example consists of a two-product plant with one period and uncertainty in the demands, for which five quadrature points were used. There are therefore 55 variables and 31 constraints. The results reported in Table 18 were obtained with a relative tolerance of 0.003.

In all instances, each term in the summation $\sum_{i=1}^{N} Q_i^{qp} \cdot \exp(t_{Li}^p - b_i)$ was underestimated independently. When the summation was considered as a single nonconvex term, only two methods (I.6 and II.1) converged to the global solution after a reasonable number of iterations. Even when each term was treated separately, some of the α computation techniques generated underestimators that were too loose for fast convergence. Significant variations in performance were observed for the successful approaches. In particular, Method II.1 with $d_i = x_i^U - x_i^L$ enabled very fast convergence, while many of the uniform diagonal shift techniques required CPU times larger by two order of magnitudes. Finally, very similar procedures such as the two instances of Method I.2 or Method II.1 led to a drastically different outcome.

Method		Iter.	CPU sec.	Relative error after 1000 iter.
Gerschgorin (I.1)		—	—	0.0650
E-Matrix (I.2)	$E = 0$	393	1018	—
E-Matrix (I.2)	$E = \mathrm{diag}(\Delta H)$	—	—	0.0284
Mori-Kokame (I.3)		—	—	0.2441
Lower bounding Hessian (I.4)		393	1041	—
Kharitonov (I.5)		—	—	0.2456
Hertz (I.6)		8	17	—
Scaled Ger. (II.1)	$d_i = 1$	—	—	0.0331
Scaled Ger. (II.1)	$d_i = (x_i^U - x_i^L)$	4	10	—
H-Matrix (II.2)		393	906	—
Min. Max. distance (II.3)		652	1164	—

Table 18: Results for Example 3.4.1.

3.4.2 Example 2

If uncertainty in the size factors and processing times is introduced in Example 1, resulting in three possible scenarios, the size of the problem increases to 155 variables and 93 constraints. For this problem, only two of the α computation techniques result in successful runs without excessive computational expense. The Hertz method (I.6) identifies the global optimum solution after 197 iterations and 7711 CPU seconds. Using the scaled Gerschgorin theorem (Method II.1) with $d_i = (x_i^U - x_i^L)$, the αBB algorithm converges after only 6 iterations and 255 CPU seconds.

3.4.3 Example 3

This larger example is a four-product plant with six stages and a single unit per stage. There is a single scenario since only the demands are considered uncertain. Once again, five quadrature points are used for each demand with a normal distribution. As a result, the problem involves 2510 variables and 649 constraints. There are 1250 nonconvex terms to be underestimated.

The algorithm converges to the optimum solution after 3 iterations and 3055 CPU seconds with the scaled Gerschgorin theorem (Method II.1) and $d_i = (x_i^U - x_i^L)$.

4 Conclusions

The theoretical developments presented by Adjiman *et al.* (1997) were used to implement a user-friendly version of the αBB global optimization algorithm. The current version incorporates the different α calculation methods, some special underestimators as well as some branching and variable bounding rules which greatly enhance the rate of convergence. These rely on an analysis of the quality of the lower bounding problem at each node. A wide variety of problems have been studied to gain a better understanding of the many available options and to test the performance of the algorithm. The examples used in this paper were taken from various categories, such as pooling problems, concave problems, chemical engineering design, generalized geometric programming and batch process design under uncertainty. In all instances, the algorithm identified the global optimum solution in satisfactory time and the scaled Gerschgorin theorem approach (Method II.1) was found especially successful at generating tight underestimators.

Acknowledgments

The authors gratefully acknowledge financial support from the National Science Foundation, the Air Force Office of Scientific Research, the National Institutes of Health, Exxon Foundation and Mobil Technology Company.

References

C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. *An Implementation of the αBB Global Optimization Algorithm: User's Guide.* Computer–Aided Systems Laboratory, Dept. of Chemical Engineering, Princeton University, NJ, 1997.

C. S. Adjiman, I.P. Androulakis, and C. A. Floudas. Global Optimization of Mixed-Integer Nonlinear Problems. in preparation, 1998.

C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. A Global Optimization Method, αBB, for General Twice–Differentiable NLPs – I. Theoretical Advances. accepted for publication, 1997.

I.P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB : A Global Optimization Method for General Constrained Nonconvex Problems. *J. of Glob. Opt.*, 7:337–363, 1995.

B. Brooks, R. Bruccoleri, B. Olafson, D. States, S. Swaminathan, and M. Karplus. CHARMM: A Program for Macromolecular Energy, Minimization and Dynamics Calculations. *J. Comput. Chem.*, 4(2):187–217, 1983.

H. Cornelius and R. Lohner. Computing the Range of Values of Real Functions with Accuracy Higher Than Second Order. *Computing*, 33:331–347, 1984.

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.

C.A. Floudas and A. Aggarwal. A Decomposition Strategy for Global Optimum Search in the Pooling Problem. *OSRA Journal on Computing*, 2(3), 1990.

S. T. Harding and C. A. Floudas. Global Optimization in Multiproduct and Multipurpose Batch Design under Uncertainty. *I&EC Res.*, 36(5):1644–1664, 1997.

V.L. Kharitonov. Asymptotic Stability of an Equilibrium Position of a Family of Systems of Linear Differential Equations. *Differential Equations*, 78:1483–1485, 1979.

O. Knüppel. *PROFIL – Programmer's Runtime Optimized Fast Interval Library.* Technische Informatik III, Technische Universität Hamburg–Harburg, 1993.

C. D. Maranas and C.A. Floudas. Global Minimum Potential Energy Conformations of Small Molecules. *J. of Glob. Opt.*, 4:135–170, 1994.

C. D. Maranas and C.A. Floudas. Global Optimization in Generalized Geometric Programming. *Computers chem. Engng.*, 21(4):351–370, 1997.

B. A. Murtagh and M. A. Saunders. *MINOS 5.4 User's Guide*. Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, CA., 1983.

G. Némethy, K.D. Gibson, K.A. Palmer, C.N. Yoon, G. Paterlini, A. Zagari, S. Rumsey, and H.A. Scheraga. Engergy Parameters in Polypeptides. 10. Improved Geometrical Parameters and Nonbonded Interactions for use in the ECEPP/3 Algorithm, with Application to Proline–containing Peptides. *J. Phys. Chem.*, 96:6472–6484, 1992.

L. B. Rall. *Automatic Differentiation : Techniques and Applications*. Lecture Notes in Computer Science. Springer-Verlag, 1981.

H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Ellis Horwood Series in Mathematics and its Applications. Halsted Press, 1988.

H. S. Ryoo and N. V. Sahinidis. Global Optimization of Nonconvex NLPs and MINLPs with Applications in Process Design. *Computers chem. Engng*, 19(5):551–566, 1995.

C. A. Schweiger, A. Rojnuckarin, and C. A. Floudas. *MINOPT : A Software Package for Mixed–Integer Nonlinear Optimization, User's Guide*. Computer–Aided Systems Laboratory, Dept. of Chemical Engineering, Princeton University, NJ, 1997.

V. Visweswaran and C. A. Floudas. New Formulations and Branching Strategies for the GOP Algorithm. In I. E. Grossmann, editor, *Global Optimization in Engineering Design*, Kluwer Book Series in Nonconvex Optimization and its Applications, 1996a. Chapter 3.

V. Visweswaran and C. A. Floudas. Computational Results for an Efficient Implementation of the GOP Algorithm and its Variants. In I. E. Grossmann, editor, *Global Optimization in Engineering Design*, Kluwer Book Series in Nonconvex Optimization and its Applications, 1996b. Chapter 4.

H. Wayland. Expansion of Determinantal Equations into Polynomial Form. *Quarterly of Applied Mathematics*, 2:277–306, 1945.

S.J. Weiner, P.A. Kollman, D.T. Nguyen, and D.A. Case. An All Atom Force Field for Simulations of Proteins and Nucleic Acids. *J. Comput. Chem.*, 7(2):230–252, 1986.

A Sample Input File

```
#
# Geometric programming example
# CSTR sequence design
#

##############################################################
                            Data
##############################################################

nxvar  2       # Number of variables
nfun   2       # Number of functions
nuterm 2       # Number of univariate concave terms
nnterm 1       # Number of general nonconvex terms
epsr   1e-3    # Relative tolerance

##############################################################
                        Name declaration
##############################################################

# Parameters : First set of rate constants and
# initial concentration
param   ka1 = 9.6540e-2, kb1 = 3.5272e-2, \
        ka2 = 9.7515e-2, kb2 = 3.9191e-2, \
        ca0 = 1

# Variable names
xvar    V1, V2

# Function names
fun     f, g
```

```
# Term names
uterm   uta, utb        # Univariate concave terms
nterm   nt              # General nonconvex terms

#################################################################
                          Options
#################################################################
funcalc standard       # Function evaluations using code list
acalc   standard       # alpha calculations using one of the
                       # methods presented in Part I

# Note that other options are specified on the command line.

################################################################
                           Terms
################################################################
uta = V1^0.5
utb = V2^0.5
nt  = -ca0 * ( ka2*V2*(1+kb1*V1) + ka1*V1*(1+ka2*V2) ) /    \
      ( (1+ka1*V1)*(1+kb1*V1)*(1+ka2*V2)*(1+kb2*V2) )

################################################################
                          Functions
################################################################
f .. nt
g .. uta + utb <= 4

################################################################
                           Bounds
################################################################
V1  lbd = 1e-6
V1  ubd = 16
V2  lbd = 1e-6
V2  ubd = 16
```

B Generalized Programming Formulations

B.1 CSTR Sequence Design

$$\min \quad -c_{b2} = -c_{a0}\frac{k_{a2}V_2\left(1+k_{b1}V_1\right)+k_{a1}V_1\left(1+k_{a2}V_2\right)}{\left(1+k_{a1}V_1\right)\left(1+k_{b1}V_1\right)\left(1+k_{a2}V_2\right)\left(1+k_{b2}V_2\right)}$$

$$\begin{aligned} \text{subject to} \quad V_1^{0.5}+V_2^{0.5} &\leq 4 \\ 10^{-6} \leq V_1, V_2 &\leq 16 \end{aligned}$$

The two sets of reaction rate coefficients shown in Table 19 are used.

	Set 1	Set 2
k_{a1}	$9.6540\ 10^{-2}\ s^{-1}$	$9.755988\ 10^{-2}\ s^{-1}$
k_{b1}	$3.5272\ 10^{-2}\ s^{-1}$	$3.919080\ 10^{-2}\ s^{-1}$
k_{a2}	$9.7515\ 10^{-2}\ s^{-1}$	$9.658428\ 10^{-2}\ s^{-1}$
k_{b2}	$3.9191\ 10^{-2}\ s^{-1}$	$3.527172\ 10^{-2}\ s^{-1}$

Table 19: Reaction rate coefficients for the CSTR sequence design

Global optimum solution: The solution for the first set of rate constants is $f^* = -0.38802$, $V_1^* = 15.992$ and $V_2^* = 10^{-6}$. For the second set, it is $f^* = -0.38881$, $V_1^* = 3.037$, $V_2^* = 5.096$.

B.2 Stability Analysis of Nonlinear Systems

B.2.1 Example 1

$$\min \quad k$$

$$\begin{aligned} \text{subject to} \quad & 10q_2^2q_3^3+10q_2^3q_3^2+200q_2^2q_3^2+100q_2^3q_3 \\ & +100q_2q_3^3+q_1q_2q_3^2+q_1q_2^2q_3+1000q_2q_3^2 \\ & +8q_1q_3^2+1000q_2^2q_3+8q_1q_2^2+6q_1q_2q_3 \\ & +60q_1q_3+60q_1q_2-q_1^2-200q_1 \leq 0 \end{aligned}$$

$$\begin{array}{rcccl} 800 - 800k & \leq & q_1 & \leq & 800 + 800k \\ 4 - 2k & \leq & q_2 & \leq & 4 + 2k \\ 6 - 3k & \leq & q_3 & \leq & 6 + 3k \end{array}$$

Global optimum solution: $k = 0.3417$, $q_1^* = 1073.4$, $q_2^* = 3.318$, $q_3^* = 4.975$. The system is unstable.

B.2.2 Example 2

$$\min \quad k$$

$$\text{subject to} \qquad q_1^4 q_2^4 - q_1^4 - q_2^4 q_3 = 0$$

$$\begin{array}{rcccl} 1.4 - 0.25k & \leq & q_1 & \leq & 1.4 + 0.25k \\ 1.5 - 0.20k & \leq & q_2 & \leq & 1.5 + 0.20k \\ 0.8 - 0.20k & \leq & q_3 & \leq & 0.8 + 0.20k \end{array}$$

Global optimum solution: $k_m = 1.089$, $q_1^* = 1.1275$, $q_2^* = 1.2820$, $q_3^* = 1.0179$. The system is stable.

B.2.3 Example 3

$$\min \quad k$$

$$\begin{array}{lrcl} \text{subject to} & q_3 + 9.625 q_1 \omega + 16 q_2 \omega + 16\omega^2 + 12 - 4q_1 - q_2 - 78\omega & = & 0 \\ & 16 q_1 \omega + 44 - 19 q_1 - 8 q_2 - q_3 - 24\omega & = & 0 \end{array}$$

$$\begin{array}{rcccl} 2.25 - 0.25k & \leq & q_1 & \leq & 2.25 + 0.25k \\ 1.5 - 0.50k & \leq & q_2 & \leq & 1.5 + 0.50k \\ 1.5 - 1.50k & \leq & q_3 & \leq & 1.5 + 1.50k \end{array}$$

Global optimum solution: $k_m = 0.8175$, $q_1^* = 2.4544$, $q_2^* = 1.9085$, $q_3^* = 2.7263$, $w^* = 1.3510$. The system is unstable.

B.2.4 Example 4

$$\min \quad k$$

$$\begin{array}{lrcl} \text{subject to} & a_4(\boldsymbol{q})\omega^4 - a_2(\boldsymbol{q})\omega^2 + a_0(\boldsymbol{q}) & = & 0 \\ & a_3(\boldsymbol{q})\omega^2 - a_1(\boldsymbol{q}) & = & 0 \end{array}$$

$$\begin{array}{rcccl} 10.0 - 1.0k & \leq & q_1 & \leq & 10.0 + 1.0k \\ 1.0 - 0.1k & \leq & q_2 & \leq & 1.0 + 0.1k \\ 1.0 - 0.1k & \leq & q_3 & \leq & 1.0 + 0.1k \\ 0.2 - 0.01k & \leq & q_4 & \leq & 0.2 + 0.01k \\ 0.05 - 0.005k & \leq & q_5 & \leq & 0.05 + 0.005k \end{array}$$

$$\begin{array}{lrcl} \text{where} & a_4(\boldsymbol{q}) & = & q_3^2 q_2 \left(4q_2 + 7q_1\right) \\ & a_3(\boldsymbol{q}) & = & 7q_4 q_3^2 q_2 - 64.918 q_3^2 q_2 + 380.067 q_3 q_2 + 3q_5 q_2 + 3q_5 q_1 \\ & a_2(\boldsymbol{q}) & = & 3\Big(-9.81 q_3 q_2^2 - 9.81 q_3 q_1 q_2 - 4.312 q_3^2 q_2 \\ & & & +264.896 q_3 q_2 + q_4 q_5 - 9.274 q_5\Big) \\ & a_1(\boldsymbol{q}) & = & \frac{1}{5}\left(-147.15 q_4 q_3 q_2 + 1364.67 q_3 q_2 - 27.72 q_5\right) \\ & a_0(\boldsymbol{q}) & = & 54.387 q_3 q_2 \end{array}$$

Global optimum solution: $k = 6.2746$, $q_1^* = 16.2746$, $q_2^* = 1.6275$, $q_3^* = 1.6275$, $q_4^* = 0.1373$, $q_5^* = 0.0186$, $w^* = 0.9864$. The system is stable.

B.2.5 Example 5

$$\min \quad k$$

$$\begin{array}{lrcl} \text{subject to} & a_8(\boldsymbol{q})\omega^8 - a_6(\boldsymbol{q})\omega^6 + a_4(\boldsymbol{q})\omega^4 - a_2(\boldsymbol{q})\omega^2 + a_0(\boldsymbol{q}) & = & 0 \\ & a_7(\boldsymbol{q})\omega^6 - a_5(\boldsymbol{q})\omega^4 + a_3(\boldsymbol{q})\omega^2 - a_1(\boldsymbol{q}) & = & 0 \end{array}$$

$$\begin{array}{rcccl} 17.5 - 14.5k & \leq & q_1 & \leq & 17.5 + 14.5k \\ 20.0 - 15.0k & \leq & q_2 & \leq & 20.0 + 15.0k \end{array}$$

$$
\begin{aligned}
\text{where } a_0(q_1,q_2) &= 453\,10^6 q_1^2\\
a_1(q_1,q_2) &= 528\,10^6 q_1^2 + 3640\,10^6 q_1\\
a_2(q_1,q_2) &= 5.72\,10^6 q_1^2 q_2 + 113\,10^6 q_1^2 + 4250\,10^6 q_1\\
a_3(q_1,q_2) &= 6.93\,10^6 q_1^2 q_2 + 911\,10^6 q_1 + 4220\,10^6\\
a_4(q_1,q_2) &= 1.45\,10^6 q_1^2 q_2 + 16.8\,10^6 q_1 q_2 + 338\,10^6\\
a_5(q_1,q_2) &= 15.6\,10^3 q_1^2 q_2^2 + 840 q_1^2 q_2 + 1.35\,10^6 q_1 q_2 + 13.5\,10^6\\
a_6(q_1,q_2) &= 1.25\,10^3 q_1^2 q_2^2 + 16.8 q_1^2 q_2 + 53.9\,10^3 q_1 q_2 + 270\,10^3\\
a_7(q_1,q_2) &= 50 q_1^2 q_2^2 + 1080 q_1 q_2\\
a_8(q_1,q_2) &= q_1^2 q_2^2
\end{aligned}
$$

By imposing an upper bound of 1 on k, the problem is determined to be infeasible, and the problem is therefore stable.

B.2.6 Example 6

$$
\min \quad k
$$

$$
\begin{aligned}
\text{subject to} \quad -a_6(\boldsymbol{q})\omega^6 + a_4(\boldsymbol{q})\omega^4 - a_2(\boldsymbol{q})\omega^2 + a_0(\boldsymbol{q}) &= 0\\
a_7(\boldsymbol{q})\omega^6 - a_5(\boldsymbol{q})\omega^4 + a_3(\boldsymbol{q})\omega^2 - a_1(\boldsymbol{q}) &= 0
\end{aligned}
$$

$$
\begin{array}{rcccl}
3.4329 - 1.2721k & \leq & q_1 & \leq & 3.4329\\
0.1627 - 0.06k & \leq & q_2 & \leq & 0.1627\\
0.1139 - 0.0782k & \leq & q_3 & \leq & 0.1139\\
0.2539 & \leq & q_4 & \leq & 0.2539 + 0.3068k\\
0.0208 - 0.0108k & \leq & q_5 & \leq & 0.0208\\
2.0247 & \leq & q_6 & \leq & 2.0247 + 2.4715k\\
1.0000 & \leq & q_7 & \leq & 1.0000 + 9.0000k
\end{array}
$$

$$
\begin{aligned}
\text{where } a_0(\boldsymbol{q}) &= 6.82079\ 10^{-5} q_1 q_3 q_4^2 + 6.82079\ 10^{-5} q_1 q_2 q_4 q_5\\
a_1(\boldsymbol{q}) &= 7.61760\ 10^{-4} q_2^2 q_5^2 + 7.61760\ 10^{-4} q_3^2 q_4^2\\
&\quad +4.02141\ 10^{-4} q_1 q_2 q_5^2 + 0.00336706 q_1 q_3 q_4^2
\end{aligned}
$$

$$
\begin{aligned}
&+6.82079\ 10^{-5}q_1q_4q_5 + 5.16120\ 10^{-4}q_2^2q_5q_6 \\
&+0.00336706q_1q_2q_4q_5 + 6.82079\ 10^{-5}q_1q_2q_4q_7 \\
&+6.28987\ 10^{-5}q_1q_2q_5q_6 + 4.02141\ 10^{-4}q_1q_3q_4q_5 \\
&+6.28987\ 10^{-5}q_1q_3q_4q_6 + 0.00152352q_2q_3q_4q_5 \\
&+5.16120\ 10^{-4}q_2q_3q_4q_6
\end{aligned}
$$

$$
\begin{aligned}
a_2(\boldsymbol{q}) \quad = \quad & 4.02141\ 10^{-4}q_1q_5^2 + 0.00152352q_2q_5^2 + 0.0552q_2^2q_5^2 \\
&+0.0552q_3^2q_4^2 + 0.0189477q_1q_2q_5^2 + 0.034862q_1q_3q_4^2 \\
&+0.00336706q_1q_4q_5 + 6.82079\ 10^{-5}q_1q_4q_7 \\
&+6.28987\ 10^{-5}q_1q_5q_6 + 0.00152352q_3q_4q_5 \\
&+5.16120\ 10^{-4}q_3q_4q_6 - 0.00234048q_3^2q_4q_6 \\
&+0.034862q_1q_2q_4q_5 + 0.0237398q_2^2q_5q_6 \\
&+0.00152352q_2^2q_5q_7 + 5.16120\ 10^{-4}q_2^2q_6q_7 \\
&+0.00336706q_1q_2q_4q_7 + 0.00287416q_1q_2q_5q_6 \\
&+8.04282\ 10^{-4}q_1q_2q_5q_7 + 6.28987\ 10^{-5}q_1q_2q_6q_7 \\
&+0.0189477q_1q_3q_4q_5 + 0.00287416q_1q_3q_4q_6 \\
&+4.02141\ 10^{-4}q_1q_3q_4q_7 + 0.1104q_2q_3q_4q_5 \\
&+0.0237398q_2q_3q_4q_6 + 0.00152352q_2q_3q_4q_7 \\
&-0.00234048q_2q_3q_5q_6 + 0.00103224q_2q_5q_6
\end{aligned}
$$

$$
\begin{aligned}
a_3(\boldsymbol{q}) \quad = \quad & 0.0189477q_1q_5^2 + 0.1104q_2q_5^2 + 5.16120\ 10^{-4}q_5q_6 + q_2^2q_5^2 \\
&+7.61760\ 10^{-4}q_2^2q_7^2 + q_3^2q_4^2 + 0.1586q_1q_2q_5^2 \\
&+4.02141\ 10^{-4}q_1q_2q_7^2 + 0.0872q_1q_3q_4^2 + 0.034862q_1q_4q_5 \\
&+0.00336706q_1q_4q_7 + 0.00287416q_1q_5q_6 \\
&+6.28987\ 10^{-5}q_1q_6q_7 + 0.00103224q_2q_6q_7 + 0.1104q_3q_4q_5 \\
&+0.0237398q_3q_4q_6 + 0.00152352q_3q_4q_7 - 0.00234048q_3q_5q_6 \\
&+0.1826q_2^2q_5q_6 + 0.1104q_2^2q_5q_7 + 0.0237398q_2^2q_6q_7 \\
&-0.0848q_3^2q_4q_6 + 0.0872q_1q_2q_4q_5 + 0.034862q_1q_2q_4q_7 \\
&+0.0215658q_1q_2q_5q_6 + 0.0378954q_1q_2q_5q_7 \\
&+0.00287416q_1q_2q_6q_7 + 0.1586q_1q_3q_4q_5 \\
&+0.0215658q_1q_3q_4q_6 + 0.0189477q_1q_3q_4q_7 + 2q_2q_3q_4q_5 \\
&+0.1826q_2q_3q_4q_6 + 0.1104q_2q_3q_4q_7 - 0.0848q_2q_3q_5q_6
\end{aligned}
$$

$$
\begin{aligned}
& -0.00234048 q_2 q_3 q_6 q_7 + 7.61760\ 10^{-4} q_5^2 + 0.0474795 q_2 q_5 q_6 \\
& +8.04282\ 10^{-4} q_1 q_5 q_7 + 0.00304704 q_2 q_5 q_7
\end{aligned}
$$

$$
\begin{aligned}
a_4(\boldsymbol{q}) \quad = \quad & 0.1586 q_1 q_5^2 + 4.02141\ 10^{-4} q_1 q_7^2 + 2 q_2 q_5^2 + 0.00152352 q_2 q_7^2 \\
& +0.0237398 q_5 q_6 + 0.00152352 q_5 q_7 + 5.16120\ 10^{-4} q_6 q_7 \\
& +0.0552 q_2^2 q_7^2 + 0.0189477 q_1 q_2 q_7^2 + 0.0872 q_1 q_4 q_5 \\
& +0.034862 q_1 q_4 q_7 + 0.0215658 q_1 q_5 q_6 + 0.00287416 q_1 q_6 q_7 \\
& +0.0474795 q_2 q_6 q_7 + 2 q_3 q_4 q_5 + 0.1826 q_3 q_4 q_6 + 0.1104 q_3 q_4 q_7 \\
& -0.0848 q_3 q_5 q_6 - 0.00234048 q_3 q_6 q_7 + 2 q_2^2 q_5 q_7 + 0.1826 q_2^2 q_6 q_7 \\
& +0.0872 q_1 q_2 q_4 q_7 + 0.3172 q_1 q_2 q_5 q_7 + 0.0215658 q_1 q_2 q_6 q_7 \\
& +0.1586 q_1 q_3 q_4 q_7 + 2 q_2 q_3 q_4 q_7 - 0.0848 q_2 q_3 q_6 q_7 + 0.0552 q_5^2 \\
& +0.3652 q_2 q_5 q_6 + 0.0378954 q_1 q_5 q_7 + 0.2208 q_2 q_5 q_7
\end{aligned}
$$

$$
\begin{aligned}
a_5(\boldsymbol{q}) \quad = \quad & 0.0189477 q_1 q_7^2 + 0.1104 q_2 q_7^2 + 0.1826 q_5 q_6 + 0.1104 q_5 q_7 \\
& +0.0237398 q_6 q_7 + q_2^2 q_7^2 + 0.1586 q_1 q_2 q_7^2 + 0.0872 q_1 q_4 q_7 \\
& +0.0215658 q_1 q_6 q_7 + 0.3652 q_2 q_6 q_7 + 2 q_3 q_4 q_7 - 0.0848 q_3 q_6 q_7 \\
& +q_5^2 + 7.61760\ 10^{-4} q_7^2 + 0.3172 q_1 q_5 q_7 \\
& +4 q_2 q_5 q_7
\end{aligned}
$$

$$
a_6(\boldsymbol{q}) \quad = \quad 0.1586 q_1 q_7^2 + 2 q_2 q_7^2 + 2 q_5 q_7 + 0.1826 q_6 q_7 + 0.0552 q_7^2
$$

$$
a_7(\boldsymbol{q}) \quad = \quad q_7^2
$$

By imposing an upper bound of 1 on k, the problem is determined to be infeasible, and the problem is therefore stable.