Noname manuscript No.
(will be inserted by the editor)

A General Framework for Convexity Analysis and an Alternative to Branch and Bound in Deterministic Global Optimization

Ralph Baker Kearfott · Jessie Castille · Gaurav Tyagi

Received: date / Accepted: date

Abstract To date, complete search in deterministic global optimization has been based on branch and bound techniques, with the bounding often done with linear or convex relaxations of the original non-convex problem. Here, we present an alternative, inspired by talks of Ch. Floudas. In this alternative, a set of non-convex variables, chosen from the intermediate variables in the expressions for the objective and constraints, is first identified. The intervals corresponding to these variables are then subdivided a priori, and the total number of subregions to be examined is known beforehand. The algorithm is designed to provide bounds on the global optimum and at least one global optimizer, with an accuracy determined a posteriori. Advantages include simplicity (less overhead), as well as easy parallelization (since subproblems to be solved are known beforehand and are independent). Furthermore, the number of non-convex variables to be subdivided with the new techniques in this paper can be considerably less than the number identified with schemes from previous work. Identification of the set of non-convex variables can be considered to be a preprocessing step. This preprocessing, done in a much smaller amount of time, reveals beforehand the practicality of using this method to solve a particular problem.

Keywords automatic verification, non-convexity, automatic differentiation, branch and bound algorithms, interval analysis, complete search

Ralph Baker Kearfott
Department of Mathematics, University of Louisiana at Lafayette, U.L. Box 4-1010, Lafayette, LA 70504-1010 USA
Tel.: 337-482-5270
Fax: 337-482-5346
E-mail: rbk@louisiana.edu

Jessie Castille
Department of Mathematics, University of Louisiana at Lafayette,
E-mail: jmc4491@louisiana.edu

Gaurav Tyagi
Department of Mathematics, University of Louisiana at Lafayette,
E-mail: gxt3687@louisiana.edu

1 Introduction

We examine the general problem of finding the global optimum and one or more optimizing points of an objective function subject to both inequality and equality constraints. In our notation, we examine the problem

$$\boxed{\begin{array}{l}\text{minimize } \varphi(x)\\ \text{subject to } c_i(x) = 0, i = 1, \ldots, m_1,\\ \qquad\qquad g_i(x) \le 0, i = 1, \ldots, m_2,\\ \text{where } \varphi : \mathbb{R}^n \to \mathbb{R} \text{ and } c_i, g_i : \mathbb{R}^n \to \mathbb{R}.\end{array}} \tag{1}$$

Deterministic algorithms for solving this general non-convex problem consist primarily of branch and bound methods, to exhaustively search a region of $\mathbb{R}^n$. Usually, the search region is a hyper-rectangle given by bounds $\boldsymbol{x} = ([\underline{x}_1, \overline{x}_1], \ldots [\underline{x}_n, \overline{x}_n])$. These bounds may or may not correspond to actual bound constraints.

Such branch and bound algorithms have many variants, but the same overall structure, namely:

1. Establish an upper bound $\overline{\varphi}$ on the global optimum over the feasible set (defined by the constraints).
2. (Branching) Subdivide the initial region $\boldsymbol{x}$ into two or more subregions $\tilde{\boldsymbol{x}}$. Place all but one of these on a list $\mathcal{L}$ for further processing.
3. Use various methods to reduce the size of $\tilde{\boldsymbol{x}}$ (possibly producing the empty set).
4. (Bounding) Bound below the range of the objective function over each subregion $\tilde{\boldsymbol{x}}$, to obtain
$$\underline{\varphi}(\tilde{\boldsymbol{x}}) \le \{\varphi(x) \mid x \in \tilde{\boldsymbol{x}}, \quad c(x) = 0, \quad g(x) \le 0.\}\,.$$
5. If $\underline{\varphi} > \overline{\varphi}$, (1) is infeasible over $\tilde{\boldsymbol{x}}$, or if $\tilde{\boldsymbol{x}}$ cannot contain a global optimizer for some other reason,
   Then
      (Pruning) Discard $\tilde{\boldsymbol{x}}$,
   Else If the diameter of $\tilde{\boldsymbol{x}}$ is smaller than a specified tolerance
      Then
         Put $\tilde{\boldsymbol{x}}$ onto a list of boxes containing possible global optimizers.
      Else
         Insert $\tilde{\boldsymbol{x}}$ into the list $\mathcal{L}$ for further branching and bounding through steps 2 and 4.
      End If
   End If

Such branch and bound methods differ in various subtle but important ways. These differences include both items that affect the efficiency (and hence practicality) of the answer and items that affect the character or quality of the solution. Some items that affect the quality of the solution are:

(a) Whether or not heuristics are used in step 5 to remove a box $\tilde{\boldsymbol{x}}$ from further consideration, thus potentially not obtaining the actual global optimum or not obtaining all optimizing points.
(b) Whether or not boxes that contain additional global optimizing points are ignored once sufficiently accurate bounds on the global optimum are obtained.

(c) Whether or not regions are sought within which the equality constraints can be shown to hold exactly, or whether points are sought within which the equality (and inequality) constraints hold only to within a specified tolerance.
(d) Whether or not roundoff error is taken into account during the computation (so that bounds produced are mathematically rigorous).
(e) Assumptions made during any preprocessing, prior to execution of the branch and bound method[1].

Some items that affect efficiency are

(a) The types of preprocessing done prior to beginning the branch and bound process.
(a) How the upper bound on the global optimum is obtained (such as with use of a local optimization process).
(b) How the ranges of the objective and constraints are bounded (such as with interval arithmetic, or with linear or convex relaxations).
(c) What acceleration procedures are used (such as interval Newton methods or a wide variety of constraint propagation techniques).

Despite such differences, all such branch and bound software consists of adaptive search of a region. In particular, it is difficult to predict beforehand how much effort it will take to solve a particular problem, and execution of such algorithms in a multiprocessing environment, although reasonable in principle, may present load-balancing problems.

In contrast, Ch. Floudas has recently been espousing an alternative point of view: Many problems require only a preliminary analysis and subdividing with respect to one variable. Although this view can be considered theoretically to be equivalent to branch and bound with branching by subdividing with respect to a subset of the variables only (something investigated in [1] and later in our work [4]), taking this alternative point of view has implications in how the algorithm is structured and in the algorithm's efficiency. In particular, instead of resulting in an adaptive process, a mesh on the variables to be subdivided can first be determined, in a simple way, before any analysis of sub-domains $\tilde{\boldsymbol{x}}$. Analysis of any sub-domain $\tilde{\boldsymbol{x}}$ then requires, essentially, the same amount of computation as that of any other such sub-domain, so the total amount of computation required is known before most of the computation is actually done. This allows simple and efficient mapping of the algorithm to multiprocessor systems, as well as making the algorithm's execution more predictable.

In this work, we present an algorithm for such a priori subdivision of selected variables. The algorithm subdivides with respect to certain intermediate variables in the computational graph for the objective and constraints, chosen using techniques we have explained in [4, 2]. The approximation to the optimum and to optimizing points is then obtained by computing solutions to linear relaxations over each pregenerated subregion. The fineness of the non-uniform mesh is determined in the a-priori subdivision phase using a combination of ideas in Theorem 5.2 and its proof from [3], along with interval arithmetic. A user-specified tolerance on the optimum and located optimizing points is achieved with this mesh-determination scheme. There is little sacrifice in efficiency in taking account of roundoff error, to assure that bounds on the objective and optimizers are mathematically rigorous.

[1] For example, the problem may be identified as linear, and a particular linear programming solver may be used.

Although our goal is to develop such "discretization" algorithms for global optimization, the main focus of this paper is determining the set of variables to subdivide. The underlying techniques for the preprocessing to determine the set of variables to be subdivided begin with our previous work [4] and [3], but go beyond it in generality and flexibility. We will not exhaustively review the techniques from [4]. However, we illustrate the new ideas with examples in §2, while we formalize some of these in §3. We present algorithms in §4, We present computational results for the preprocessing and preliminary results using a prototypical implementation of the overall discretization algorithm in §5, while we mention theoretical limitations in §6. A synopsis appears in §7.

2 Simple Illustrative Examples

We illustrate our underlying ideas with the following simple example[2].

Example 1

$$\begin{aligned} &\text{minimize } -2x_1^2 - x_2^2 \\ &\text{subject to } x_1^2 - x_2^2 = 0, \\ &\qquad\qquad x_1^2 + x_2^2 - 1 \le 0, \\ &\qquad\qquad x_1^2 - x_2 \le 0, \\ &(x_1, x_2) \in ([0,1],[0,1]). \end{aligned} \tag{2}$$

Following [4], we add constraints corresponding to each operation involved in evaluating the original objective and constraints in (2), analyze which of the resulting equality constraints can be replaced by inequality constraints to result in an equivalent problem, then determine which of the corresponding operations are convex, resulting in what we call the *equivalent relaxed expanded NLP*. The equivalent relaxed expanded NLP corresponding to Problem 2 is depicted in Table 1 and Figure 1. As part of the analysis, those intermediate variables are identified for which sufficiently sharp linear relaxations can be obtained only by subdividing the interval corresponding to those intermediate variables and solving a different relaxation over each subinterval.

The corresponding computational graph occurs in Figure 1. There, the node numbers correspond to variable numbers, and the links are labeled with the corresponding operations. Variables that are the result of a non-convex operation are labeled with a "T", while the leaves are labeled with "L". We see that only node 3 and node 6 correspond to non-convex operations, while the leaves correspond to node 9 (corresponding to the constraint $x_1^2 + x_2^2 - 1 = v_9 \le 0$), node 7 (corresponding to the objective $-2x_1^2 - x_2^2 = v_7 \le \overline{\varphi}$, where $\overline{\varphi}$ is a known upper bound on the global optimum), node 11 (corresponding to the constraint $x_1^2 - x_2^2 = v_{11} \le 0$), and node 10 (corresponding to the constraint $x_1^2 - x_2^2 = v_{10} = 0$).

In [4] a subset of the set of independent variables was identified such that, if the bounds on the variables in that subset are sufficiently narrow, the problem is approximately convex. One alternative way of identifying variables to subdivide is to start at the top of the computational graph, corresponding to the independent variables x_1 and x_2, and to search down the computational graph to find the first node labeled "T", that is the first node that corresponds to the result of a non-convex operation.

[2] This example problem can easily be solved with a variety of methods, but is simple enough to present complete details.

Table 1 Operations and convexity analysis corresponding to Example 1.

♯	Operation	Enclosure	Constraint	Needs Subdivision?
1	$v_3 \leftarrow x_1^2$	$[0,1]$	$x_1^2 - v_3 = 0$	yes
2	$v_4 \leftarrow 2v_3$	$[0,2]$	$2v_3 - v_4 \geq 0$	no
3	$v_5 \leftarrow -v_4$	$[-2,0]$	$-v_4 - v_5 \leq 0$	no
4	$v_6 \leftarrow x_2^2$	$[0,1]$	$v_2^2 - v_6 = 0$	yes
5	$v_7 \leftarrow v_5 - v_6$	$[-3,0]$	$v_5 - v_6 - v_7 \leq 0$	no
6	$v_8 \leftarrow v_3 + v_6$	$[0,2]$	$v_3 + v_6 - v_8 \leq 0$	no
7	$v_9 \leftarrow v_8 - 1$	$[-1,1]$	$v_8 - 1 - v_9 \leq 0$	no
8	$v_{10} \leftarrow v_3 - v_2$	$[-1,1]$	$-1 - v_{10} \leq 0$	no
9	$v_{11} \leftarrow v_3 - v_6$	$[-1,1]$	$v_3 - v_6 - v_{11} = 0$	no

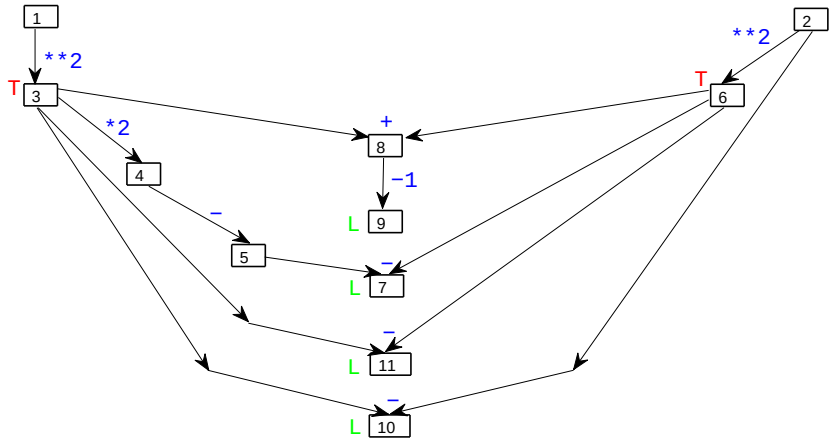

Fig. 1 A computational graph corresponding to Example 1

For Example 1, this is node 3. Assuming variable v_3 will be subdivided, the operations whose input arguments include node 3 are then marked: unary operations are marked as not needing subdivision, while binary operations are marked that subdivision is not required for the first argument, or marked as subdivision not needed if the second argument either occurs in a convex way or has already been marked. In the case of Example 1, there are no non-convex operations whose input arguments include v_3, so no nodes are marked.

After node 3 has been identified and any possible non-convex nodes below it in the computational graph have been appropriately marked, additional non-marked non-convex operations are sought below node 3 in the table of operations. The next node to be encountered is node 6. Search of the table of operations below node 6 reveals no additional non-convex operations. This, the space in which subdivision occurs is with respect to v_3 and v_6.

The proposed algorithm would proceed as follows:

1. Form a priori a 2-dimensional mesh in (v_3, v_6) space, that is subdivide both the interval $[0,1]$, $v_3 \in [0,1]$, into M subintervals I_k, $1 \leq k \leq M$, and subdivide the interval $[0,1]$, $v_6 \in [0,1]$ into N subintervals J_ℓ, $1 \leq \ell \leq N$..
2. For a linear relaxation for each of the MN subinterval pairs $v_3 \in I_k$, $v_6 \in J_\ell$, after narrowing the values of the other variables as much as possible with constraint propagation.
3. We see that node 10 (i.e. v_{10}) depends directly on the independent variable x_2, and its interval will not be narrowed by substituting the I_k and J_ℓ into the expressions containing v_3 and v_6. However, since the operation corresponding to node 10 is linear, an accurate linear relaxation can be obtained without subdivision of x_2 (provided different relaxations are solved for each of the intervals I_k and J_ℓ).
4. Solve each of the linear relaxations, obtaining a posteriori bounds on the global optimum and nearness to feasibility of the original problem using interval analysis, as well as a set of optimizing points corresponding to the ϵ-approximate solutions so obtained.

We see that, in the case of Example 1, the number of variables to be subdivided is the same as the original number of variables. However, in general, there may be far fewer such variables, and even fewer than the number of variables given through the analysis in [4]. Furthermore, by directly subdividing the range of the top (highest in the computational graph) non-convex operations, we avoid some of the complication of the top-level non-convex operations.

Example 2 Minimize

$$\varphi(x) = (x_1 + x_2 - 1)^2 - \left(x_1^2 + x_2^2 - 1\right)^2$$

for $x_1 \in [-1,1]$ and $x_2 \in [-1,1]$.

This example first appeared in [4] to illustrate the convexity analysis explained there. Table 2 gives the resulting equivalent expanded NLP, while Figure 2 gives the corresponding computational graph.

Table 2 Operations and convexity analysis corresponding to Example 2.

♯	Operation	Enclosures	Constraints	Needs Subdivision?
1	$v_3 \leftarrow x_1 + x_2$	$[-2,2]$	$x_1 + x_2 - v_3 = 0$	no
2	$v_4 \leftarrow v_3 - 1$	$[-3,1]$	$v_3 - 1 - v_4 = 0$	no
3	$v_5 \leftarrow v_4^2$	$[0,9]$	$v_4^2 - v_5 \leq 0$	no
4	$v_6 \leftarrow x_1^2$	$[0,1]$	$v_1^2 - v_6 = 0$	yes
5	$v_7 \leftarrow x_2^2$	$[0,1]$	$v_2^2 - v_7 = 0$	yes
6	$v_8 \leftarrow v_6 + v_7$	$[0,2]$	$v_6 + v_7 - v_8 = 0$	no
7	$v_9 \leftarrow v_8 - 1$	$[-1,1]$	$v_8 - 1 - v_9 = 0$	no
8	$v_{10} \leftarrow v_9^2$	$[-1,0]$	$-v_9^2 + v_{10} \leq 0$	yes
9	$v_{11} \leftarrow v_5 - v_{10}$	$[-1,9]$	$v_5 - v_{10} - v_{11} \leq 0$	no

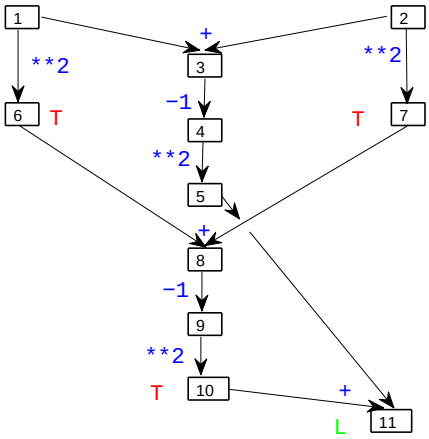

Fig. 2 A computational graph corresponding to Example 2

For Example 2, v_6 is first identified as needing subdivision, and the next variable in the operations table corresponding to a non-convex operation is v_7; since v_7 is independent of v_6, it is marked as needing subdivision. All variables below v_6 and v_7 that either depend on v_6 and v_7 or in a convex way on variables above v_6 and v_7 are marked as not needing subdivision. In this way, v_{10} is marked as not needing subdivision. The variables for the a priori mesh will therefore be v_6 and v_7.

Note that, for Example 2, it is sufficient to only subdivide v_{10}, posing the relaxations only in terms of v_{10} and v_5; once the solutions for v_5 and v_{10} are found. This reduces the mesh to a one-dimensional one, and greatly simplifies the corresponding linear relaxations, but results in more work in the process to subsequently determine the values of the original variables. Moreover, the level of the nodes to use for subdivision in the computational graph for a minimal amount of work is probably highly problem-dependent.

Example 3 Minimize

$$\varphi(x) = (x_1 x_2 - 1)^2$$

for $x_1 \in [-2, 2]$ and $x_2 \in [-2, 2]$.

Table 3 gives the resulting equivalent expanded NLP, while Figure 3 gives the corresponding computational graph.

Table 3 Operations and convexity analysis corresponding to Example 3.

♯	Operation	Enclosures	Constraints	Needs Subdivision?
1	$v_3 \leftarrow x_1 x_2$	$[-4, 4]$	$x_1 x_2 - v_3 = 0$	yes
2	$v_4 \leftarrow v_3 - 1$	$[-5, 3]$	$v_3 - 1 - v_4 = 0$	no
3	$v_5 \leftarrow v_4^2$	$[0, 25]$	$v_4^2 - v_5 \leq 0$	no

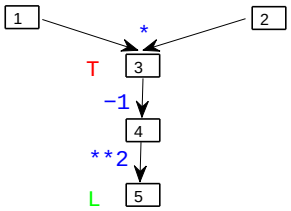

Fig. 3 A computational graph corresponding to Example 3

Here, there is only one node corresponding to a variable, v_3 whose interval needs to be subdivided. Now suppose, for example, we subdivide $[-4,4]$, $v_3 \in [-4,4]$ into 8 subintervals $I_1 = [-4,-3]$, $I_2 = [-3,-2]$, ..., $I_8 = [3,4]$, and furthermore approximate the convex operation $v_5 \leftarrow v_4^2$ with a sufficient number of tangent lines for each of the 8 linear relaxations to get a good approximate value for the optimum of v_5 over each subinterval. Let $v_{3,k}^*$ be the optimizer over I_k and $v_{5,k}^* = \varphi_k^* = \tilde{\varphi}(v_{3,k}^*)$. We then have the following table.

k	$v_{3,k}^*$	$\tilde{\varphi}(v_{3,k}^*)$
1	≈ -3	≈ 16
2	≈ -2	≈ 9
3	≈ -1	≈ 4
4	≈ 0	≈ 1
5	≈ 1	≈ 0
6	≈ 2	≈ 1
7	≈ 3	≈ 4
8	≈ 4	≈ 9

Thus, $\tilde{\varphi}(v_{3,k}^*) \approx 0$ would correspond to the global optimum of the original problem, and $v_{3,k}^* \approx 1$ would correspond to the actual unique optimizing point when the original problem is reparametrized in terms of v_3. However, when we attempt to use constraint propagation to then determine the values of x_1 and x_2, we have only $x_1 x_2 \approx 1$, along with $x_1 \in [-2,2]$ and $x_2 \in [-2,2]$. Nonetheless, this is not a failure of the method, since the entire portion of the curve $x_1 x_2 = 1$ within the box $([-2,2],[-2,2])$ *actually is* the solution set for this problem, no more and no less. Such singular solution techniques sometimes are identified incidentally with this technique. Once such singularities have been identified, they may be presented in terms of the defining relations, or they may be followed using techniques in [9]. A complete search branch and bound algorithm, such as GlobSol [5], designed to find all optimizing points, will work inefficiently, since it must cover the entire solution set with small boxes, and since the Hessian, Kuhn–Tucker, or Fritz John matrix is singular at points on such solution sets[3].

[3] In fact, the current version of GlobSol, running on a 2.4GHz machine and compiled with the Gnu compiler suite, fails to complete after processing 50,000 boxes, using 560 CPU seconds, but a graph of the uncompleted boxes reveals a close approximation of the intersection of the two branches of $xy = 1$ with the box $([-2,2],[-2,2])$.

3 Some Definitions

It will help to formally clarify the concepts illustrated in the preceding examples.

Definition 1 A *non-convex* variable is an intermediate variable in the computational graph of the function that has been identified to be the result of a non-convex operation, according to the analysis in [4, §5][4]. Variables that are not non-convex are called *convex variables*.

In the tables in the preceding section, the non-convex variables are those marked as "needs subdivision." For instance, in Table 1, variables 3 and 6, corresponding to rows 1 and 4, are non-convex variables. Non-convex variables correspond to operations and constraint senses ($\geq$, $\leq$ or $=$) for which tight relaxations can be obtained only by using different relaxations on multiple sub-intervals.

Definition 2 A *complete set of non-convex variables* is a set of non-convex variables such that every backwards (i.e. upwards) trace from leaves (corresponding to the objective and constraint values) in the computational graph either only encounters convex variables or eventually encounters an element of the set.

Thus, if we subdivide the elements of a complete set, forward substitution in the computational graph (using interval computation) will result in narrow bounds on every other non-convex variable.

Definition 3 A *minimal complete set* of non-convex variables is a complete set such that no set formed by removing any element is also a complete set.

We desire complete sets to be minimal to keep the dimension of the space in which we must form the mesh small. The idea is that we will subdivide intervals corresponding to the complete set.

The examples and discussion in the preceding section illustrate advantages and disadvantages of choosing a complete set from elements high or low in the computational graph. The following two definitions single out sets that are high in the computational graph and low in the computational graph.

Definition 4 A *dependent minimal complete set* is a minimal complete set such that every forward (i.e. downward) trace towards leaves (corresponding to the objective and constraint values) in the computational graph encounters only elements of the set or convex variables.

Dependent minimal complete sets lead to particularly simple linear programs. In fact, the problem becomes convex when the elements of the dependent minimal complete set are viewed as independent variables, and variables preceding the elements of the minimal complete set are ignored. For example, $\{v_6, v_7\}$ is a minimal complete set in Example 2, whereas $\{v_{10}\}$ is a dependent minimal complete set for that problem. If we reparametrize the problem as

$$\min v_{10} + v_5,$$

[4] This analysis is done in the GlobSol routine `subspace_analysis/identify_subproblems.f90`.

its linear relaxation is much simpler (and, indeed, is equal to the problem itself, in this case). One procedure is to subdivide v_{10} only, then solve linear relaxations associated with each of the subintervals of v_{10} to find a lower bound on the global optimum. After the approximate global optimum is found, constraint propagation or branch and bound can be used to find the solution in terms of the original variables. Note that bounds on the global optimum can possibly[5] be obtained inexpensively in this manner, before bounds on optimal solutions in terms of the original variables are obtained.

Proposition 1 *The dependent minimal complete set is unique. Furthermore, if the dependent minimal complete set is empty, the problem is convex.*

Proof If we start any backward trace in the computational graph, the first variable resulting from a non-convex operation must be in the set. Since the first such variable along a particular trace is well defined and since each separate backwards trace is well defined, the set of all first such variables is unique. Furthermore, by the definitions, this set is a complete set, and must be minimal. This proves that the set so constructed is the unique dependent minimal complete set.

If the dependent minimal complete set is empty, that means that each leaf, that is, the objective and each constraint, are computed only with convex operations. Therefore, the entire optimization problem is convex. □

In some contexts, it may be important to obtain optimal solutions in terms of the original parameters, but the problem of solving for the original parameters from the solution in terms of a dependent minimal complete set may be difficult. In such cases, it may be advantageous to choose for subdivision non-convex variables higher in the computational graph. The following defines a minimal complete set that is as high as possible without including the original independent variables.

Definition 5 An *independent minimal complete set* of non-convex variables is a minimal complete set of non-convex variables such that no non-convex variable can be encountered by tracing backward in the computational graph from elements of the set.

In fact, an independent minimal complete set may not exist. To illustrate, consider the following variant of Example 3

Example 4 (non-existence of an independent minimal complete set)

$$\min(x_1x_2x_3x_4x_5 - 1)^2.$$

The computational graph for Example 4 is in Figure 4. This example does not have an independent minimal complete set, while the only minimal complete set is a dependent minimal complete set. In fact, for this illustrative example, the solution to the original problem cannot be given more precisely than through the relation

$$x_1x_2x_3x_4x_5 = 1,$$

a relation that is found by solving the problem constructed in terms of the dependent minimal complete set, namely, "$\min(x_9 - 1)^2$."

Finally, we give formal notation for the tables of operations.

[5] depending on the cardinality of the dependent minimal complete set

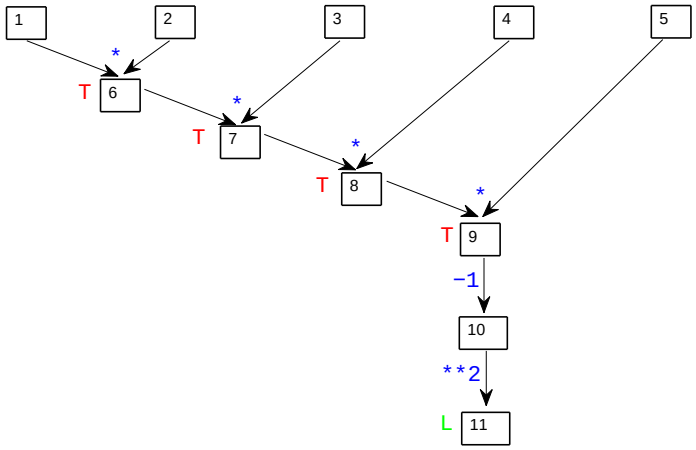

Fig. 4 A computational graph corresponding to Example 4

Definition 6 A *non-convexity code list* corresponding to an instance of an optimization problem of the form (1) is an array of row numbers ℓ, $1 \leq \ell \leq N_{\text{ops}}$ (where N_{ops} is the total number of operations), corresponding result variable index p_ℓ, argument indices q_ℓ and r_ℓ, operation code ω_ℓ, the binary variable ν_ℓ, where ν_ℓ is "true" if and only if p_ℓ is a non-convex variable, and a set $\mathcal{F}$ of indices ℓ corresponding to leaves (i.e. corresponding to objective and constraints) of the computational graph.

The variable indices in a non-convexity code list that do not appear as operands q_ℓ or r_ℓ correspond to leaves in the corresponding computational graph, and also correspond to the objective and constraints.

We use the terms "table" and "code list," while closely related terms in the literature are "tape" and "directed acyclic graph," or "DAG". Also, our tables contain only unary and binary operations[6]. Much work has been done concerning relaxations based on operations with larger numbers of arguments (such as triple products $w = xyz$); the methods introduced here can generalize to such schemes.

Our final definition will help us to make precise what we mean when we say we have rigorously verified solutions.

Definition 7 An ϵ-approximate solution to (1) is a point $\check{x}$ such that:

1. $|c_i(\check{x})| \leq \epsilon$, $1 \leq i \leq m_1$,
2. $g_i(\check{x}) \leq \epsilon$, $1 \leq i \leq m_2$,
3. $\varphi(\check{x}) \leq \overline{\varphi}$, where $\overline{\varphi}$ is an upper bound on φ from among all points satisfying 1 and 2, and $\overline{\varphi} - \varphi^* \leq \epsilon$, where φ^* is the global optimum of φ.

Bounds on ϵ-approximate solutions are easier to mathematically rigorously verify than bounds on solutions that exactly satisfy the original constraints, particularly equality constraints or active inequality constraints[7].

[6] consistent with the current implementation in GlobSol

[7] GlobSol works with exact solutions, rather than ϵ-approximate solutions.

4 Algorithms

The following algorithm finds a complete set high in the computational graph that is a candidate for an independent minimal complete set.

Algorithm 1 (Finding a complete set high in the computational graph)
INPUT: The non-convexity code list corresponding to the problem.
OUTPUT: A list $\mathcal{I}$ of indices corresponding a complete set high in the computational graph that is possibly an independent minimal complete set.

1. *(Initialization)*
   (a) *Set* $\mu_{\ell,i} \leftarrow$ *"false"*, $\ell = 1, 2$, $1 \le i \le N_{ops}$. (μ will be used to identify those non-convex variables that do not need subdivision because all variables above them are subject to subdivision.)
   (b) $\mathcal{I} \leftarrow \emptyset$.
   (c) $\mathcal{S} \leftarrow \emptyset$ *and* $\tilde{\mathcal{S}} \leftarrow \emptyset$. ($\mathcal{S}$ and $\tilde{\mathcal{S}}$ are temporary lists of nodes used to determine nodes ℓ whose variable ranges will be narrow when variables whose node indices are already stored in $\mathcal{I}$ are narrow.)
2. FOR $\ell = 1$ *to* N_{ops}
   IF $\{\nu_\ell$ *and not* $[\mu_{1,\ell}$ *and* $(\mu_{2,\ell}$ *or* ω_ℓ *is univariate*$)]\}$ THEN
   (a) *Append* ℓ *to* $\mathcal{I}$.
   (b) FOR $j = \ell + 1$ *to* N_{ops}
      i. IF $q_j = p_\ell$ THEN $\mu_{1,j} \leftarrow$ *"true"*.
      ii. IF $r_j = p_\ell$ *or* ω_j *is univariate* THEN $\mu_{2,j} \leftarrow$ *"true"*.
      iii. IF *both* $\mu_{1,j}$ *and* ($\mu_{2,j}$ *or* ω_j *is univariate*) THEN
         A. *Insert* j *into* $\mathcal{S}$.
         B. DO WHILE ($\mathcal{S} \neq \emptyset$)
         (1) FOR $k \in \mathcal{S}$
            FOR $i = k + 1$ *to* N_{ops}.
            (α) IF $q_i = p_k$ THEN $\mu_{1,i} \leftarrow$ *"true"*.
            (β) IF $r_i = p_k$ *or* ω_i *is univariate* THEN $\mu_{2,i} \leftarrow$ *"true"*.
            (γ) IF *both* $\mu_{1,i}$ *and* $\mu_{2,i}$ THEN *Insert* i *into* $\tilde{\mathcal{S}}$.
            END FOR
         END FOR
         (2) $\mathcal{S} \leftarrow \tilde{\mathcal{S}}$.
         (3) $\tilde{\mathcal{S}} \leftarrow \emptyset$.
         END DO
      END FOR
   END IF
END FOR

End Algorithm 1.

One can think of Algorithm 1 as identifying a reduced problem associated with the portion of the computational graph below the identified independent minimal complete set. The reduced problem corresponding to Example 2 is

Example 5

$$\begin{aligned} &\text{minimize } v_{11}(v_6, v_7) \\ &(v_6, v_7) \in (I_k, J_\ell, [0, 9]), \quad \cup I_k = [0, 2], \quad \cup J_\ell = [-1, 1]. \end{aligned} \tag{3}$$

Note that v_{11}, the objective, also depends on variable 5, not descended only from variables 6 and 7. However, necessarily, variables such as v_5 are convex and must be descended from variables that are descended only from convex variables and variables in the complete set identified by the algorithm.

The following algorithm finds the dependent minimal complete set, that is, a minimal complete set low in the computational graph.

Algorithm 2 (Finding the dependent minimal complete set)
INPUT: The non-convexity code list corresponding to the problem.
OUTPUT: A list $\mathcal{D}$ of indices corresponding to the dependent minimal complete set.

1. *Mark every operation as not yet fathomed:* $\texttt{fathomed}_i \leftarrow \texttt{false}$, $i = 1$ *to the number of operations in the code list.*
2. FOR $\ell = 1$ *to the number of leaves in the computational graph.*
   (a) *Push the index of leaf* ℓ *onto the stack* $\mathcal{R}$ *of indices to be analyzed.*
   (b) (This is a complete depth-first traversal of the tree that is the portion of the computational graph above leaf ℓ, with edges of reverse sense to those of the original computational graph.)
   DO WHILE *the stack* $\mathcal{R}$ *is non-empty.*
      i. *Pop an index* ρ *from* $\mathcal{R}$.
      ii. IF ρ *does not correspond to an independent variable or a constant*[8] *and* $\texttt{fathomed}_\rho = \texttt{false}$
      THEN
         IF ρ *is non-convex* THEN
         A. $\texttt{fathomed}_\rho \leftarrow \texttt{true}$.
         B. *Store the variable index* ρ *in* $\mathcal{D}$.
         ELSE
            Push the variable index of the variable (for univariate operations) of both variables (for binary operations) that are arguments to the operation leading to the ρ *variable onto* $\mathcal{R}$.
         END IF(operation is convex or not)
      END IF(not independent or constant and not fathomed)
   END DO (traversal of tree corresponding to leaf ℓ)
END FOR (loop on all leaves ℓ)

End Algorithm 2.

Unfortunately, it can occur that there are more variables in an identified dependent minimal complete set than in the set of non-convex variables identified with the techniques of [4], or even more than the number of independent variables. This can happen if there are many constraints with non-convex terms that aren't related, as in the following.

Example 6

$$\begin{aligned} &\text{minimize } x_1^2 + x_2^2 \\ &\text{subject to } -2x_1^3 + 2 - x_2 \le 0, \\ &\qquad\qquad -x_1^2 + 1 - x_2 \le 0, \\ &\qquad\qquad \sqrt{x_1} - x_2 \le 0, \\ &(x_1, x_2) \in ([0,1],[0,1]). \end{aligned} \tag{4}$$

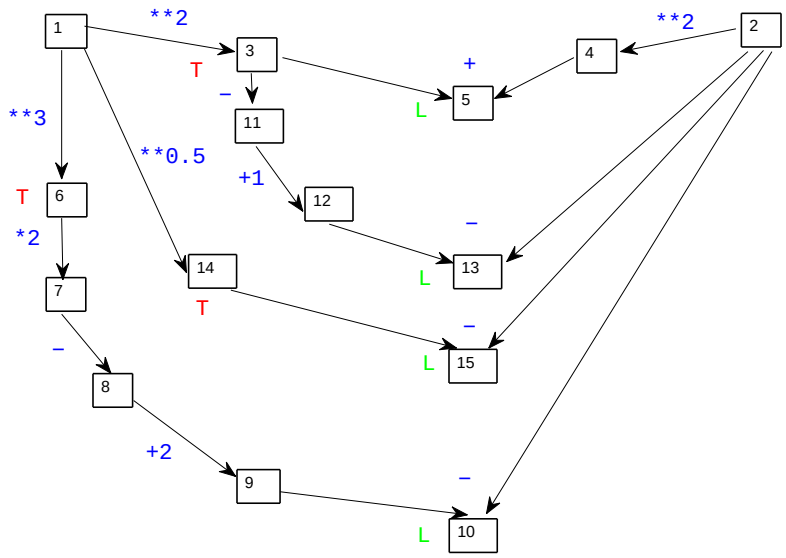

Fig. 5 A computational graph corresponding to Example 6

The computational graph for Example 6 is in Figure 5. We see that the function in each of the three inequality constraints is non-convex, and there necessarily are three non-convex variables in the only minimal complete set. However, the analysis from [4] shows that only one independent variable, x_1, need be subdivided to make all non-convex variables small. In such instances, we might favor the independent variables or the subset of independent variables some of whose progeny are non-convex.

In any case, problems reformulated in terms of a dependent minimal complete set are necessarily convex over each subregion. Therefore, any convex solver can be used to find an approximate solution, and once the approximate solution is found, it can be verified using local methods. The approximate solutions from the separate subregions can then be combined to obtain the overall global solution.

Once a set of variables has been selected and the problem has possibly been reformulated in terms of the identified set, the following algorithm will find bounds on the optimum and ϵ-approximate solutions.

Algorithm 3 (Creating a mesh, computing an ϵ, and finding ϵ-approximate solutions)
INPUT: The code list and search box $\boldsymbol{x}$ for the problem, as well as the number of subdivisions M for each non-convex variable.
OUTPUT: A number ϵ for ϵ-approximate solutions, as well as a list $\mathcal{E}$ of boxes guaranteed to contain ϵ-approximate solutions (but not necessarily all such solutions).

1. (Do initial preprocessing)
   (a) *Compute interval bounds $\boldsymbol{v}$ on all of the intermediate variables in the code list by evaluating them over the box $\boldsymbol{x}$.*

[8] that is, if the corresponding operation is not setting the variable to a constant

(b) *Using Algorithm 1, Algorithm 2, the scheme in [4], a combination of these, or related alternative scheme compute a set $\mathcal{V}$ to be used as the new independent variables in the re-parametrized problem.*
(c) *Based on the cardinality N_v of the set $\mathcal{V}$ from step 1b, determine whether or not it is practical to proceed with this algorithm. (Otherwise stop here.)*
(d) *If $N_v = 0$, mark the entire problem as convex, and set M to 1.*
(e) *Subdivide each interval $\boldsymbol{v}_i$, $i \in \mathcal{V}$ into M subintervals, in preparation for generating M_v^N subproblems to be solved approximately.*

2. (Bound solutions to the subproblem for each individual box)
FOR $i = 1$ *to* M^{N_v}
(a) *Create the i-th box $\boldsymbol{v}_i$ in the space defined by $\mathcal{V}$, $\boldsymbol{v}$, and the number of subdivisions.*
(b) *Approximately solve the created subproblem*[9].
(c) *Compute verified lower and upper bounds on the global optimum over the subbox, or else prove that the box is infeasible with respect to the original constraints*[10].
(d) IF *the subproblem corresponding to $\boldsymbol{v}_i$ is feasible*
THEN
i. *Mark feasibility, and store lower and upper bounds.*
ii. *Compute (e.g. with interval arithmetic) bounds on the ranges of the objective and constraints at the approximate solution, to compute and store an ϵ_i corresponding to the computed approximate solution.*
iii. *Store the approximate solution.*
END IF
END FOR

3. (Postprocessing)
(a) *Compute a smallest upper bound on the global optimum to the ϵ-approximate problem by taking the minimum of the upper bounds for each of the subproblems marked as feasible.*
(b) *Based on the results of step 3a, store in $\mathcal{E}$ those approximately feasible boxes whose objective lower bounds are sufficiently small.*
(c) *Compute an ϵ for the entire problem by taking the maximum of the ϵ_i over each of the M_v^N sub-problems corresponding to $\mathcal{E}$.*

End Algorithm 3.

Many important implementation details are left out of our description of Algorithm 3. For instance, constraint propagation may be used at many points in this algorithm. Constraint propagation may cause additional unpredictability in the amount of time each individual subproblem will take, but could be important if sets other than dependent minimal complete sets are used, or if expressions in terms of the original variables are used during the process[11].

Important aspects of Algorithm 3 are:

[9] e.g. by forming linear relaxations, or, if starting with a dependent minimal complete set, by using a solver for convex problems

[10] If using linear relaxations, this may be done with the simple and inexpensive technique in [6]. If using a solver for convex problems, this can possibly be done with an astutely implemented interval Newton method applied over a small box constructed about the approximate solution.

[11] not recommended, but easier to implement

1. In contrast to a branch and bound algorithm, the total number M^{N_v} of nodes to be traversed is determined a priori, while this number of nodes determines the ϵ in the ϵ-approximations to the solution. This ϵ is known only after completion of the algorithm. Thus, a much better estimate for the total computational effort is known beforehand, in exchange for an approximate solution whose accuracy is not pre-determined.
2. Each iteration of the main loop (Step 2) requires much more nearly the same amount of computational effort (in comparison with the computational efforts of two branches in a branch and bound process), and each such iteration is independent of the other iterations. Thus, load balancing should not be needed, and mapping the algorithm to systems with a large number of processors should be easier.
3. The algorithm produces mathematically rigorous lower and upper bounds to the optimum of original problem, but possibly doesn't output enclosures to all optimizing points.
4. The algorithm is limited by the dimension N_v, the number of subdivisions M, and the total number of processors available. However, it is known beforehand whether or not a run of the algorithm will be practical, and the result of a run always gives *some* information.

5 Numerical Results

To get a preliminary idea of the usefulness of the analysis, we tried the analysis on the 175 problems from the COCONUT Lib-1 test set [10] with the smallest code lists[12]. In considering the cardinality of the selected set of variables, we compared Algorithm 1 and Algorithm 2 to the selection process from [4], where the selection process from [4] gives a subset of the set of independent variables such that, when those variable ranges are sufficiently small, the problem is approximately convex. Table 4 summarizes the results. In Table 4, the columns and rows are as follows:

Table 4 Summary of four parametrization schemes on 175 standard test problems.

Scheme	# better than original	# best	# ≤ 4
Original independent variables	—	—	41
Independent complete set	70	0	46
Dependent minimal complete set	135	97	124
Scheme from [4]	131	26	83

"# better than original" is the number of problems for which the number of identified variables was smaller than the original number of independent variables.

[12] We limited these because of a temporary, surmountable efficiency issue in programming removal of redundant operations in GlobSol; our selected code lists are those that consume less than roughly 2MB of disk space.

"# best" is the number of problems for which the number of identified variables is strictly less than the number of identified variables for any of the other three schemes.

"# ≤ 4" is the number of problems for which the scheme produced a set with 4 or less variables[13].

"Independent complete set" refers to the scheme in Algorithm 1.

"Dependent minimal complete set" refers to the scheme in Algorithm 2.

From this summary, we conclude that no one scheme shows a definite superiority, except that the independent complete set defined by Algorithm 1 does not generally give a smaller number of variables than the other schemes. A practical algorithm might combine all of these schemes and other schemes based on the general techniques in this work.

We also tried the solution-by-discretization algorithm, Algorithm 3, on the 84 of the 175 analyzed problems for which either the scheme from [4] or Algorithm 1 gave a dimension of 4 or less. In these, the number of independent variables in the original problems ranges from 1 to 108. We used 20 equally spaced subdivisions in each variable, although 10 problems were identified as convex, and therefore required solution of only one relaxation. In our algorithm, we chose for parametrization the variable set from the original set of independent variables, the scheme from [4], and the dependent minimal complete set which gave the smallest number of variables. The purpose of this initial check was to get an idea of the practicality of the overall procedure, and the implementation was somewhat crude. In particular, absent up-to-date licenses and installation, we used the outdated SLATEC routine DSPLP[14]. We also used the full linear program generated by the GlobSol module `LP_operations`, without trying to formulate the simpler problems such as those defined by the dependent minimal complete set: We input a particular sub-box into the values in the code list and used constraint propagation in various places to narrow the other values. This easy implementation should give good bounds on the objective and constraints, especially when the dependent minimal set is used, but may not give good values on the independent variables or make the corresponding linear relaxations optimally easy to solve efficiently. Nonetheless, we should get a preliminary idea of how practical or applicable the scheme is.

We ran our problem set within the GlobSol environment on a core-2-duo laptop with clock speed 2.5GHz and 4GB of memory, running Ubuntu 10.10 and the latest supported version of the Gnu compiler suite. Of the 84 problems, DSPLP failed to compute a solution, detect infeasibility, or detect unboundedness in 26 problems[15]. However, the technique was shown to be practical, with only one problem taking 805 seconds, with a median of 0.312 seconds, and with an average of 46.8 seconds. The number of epsilon-approximate solution boxes returned averaged approximately 149, with a median of 1, and with only one problem returning 8000 boxes[16]. The bound on ϵ in the ϵ-approximate solutions was small in many cases, although this aspect of our algorithm needs to be refined.

[13] This is a measure of whether or not the scheme leads to a parametrization for which Algorithm 3 is practical.

[14] This routine is apparently not robust with respect to ill-conditioning in the constraint matrix, something that happens when convex constraints are approximated closely with numerous tangent lines.

[15] This will undoubtedly be improved with a better implementation and a better LP solver.

[16] Note that, for $n = 4$, $20^4 = 160,000$ boxes are considered.

The GlobSol environment, along with programs for replicating these results, is available upon request from the first author, by access to a version control system.

6 Limitations

The examples and computational tests have illustrated that there are problems for which each of the particular schemes presented is best suited, and problems for which any particular scheme will fail. Furthermore, the discretization procedure embodied in Algorithm 3 is applicable only when the number of variables identified for subdivision is sufficiently small, the number of subdivisions required to approximate a convex problem well is small, or the number of available processors is sufficiently large. However, this can be determined relatively easily before the main discretization algorithm is actually tried.

Situations such as in Example 4 illustrate that the set of optimizers for the original problem may be a continuum, and hence not easily enclosed with branch and bound algorithms. In such instances, techniques such as those in [9] can be used in some cases to trace solutions. Otherwise, such dependencies can sometimes be identified with the techniques we have explained, with various possibilities for further analysis and presentation.

It must be pointed out that, although rigorous bounds on the global optimum are obtained with the techniques we have explained, use of relaxations, even if implemented with rigorous bounding of roundoff error, does not necessarily lead to bounding of *all* optimizers.

In cases where an adequately small dimension cannot be found with the techniques in this paper and a general branch and bound algorithm does not give results in a predictable amount of time, the limited memory algorithm in [7, Chapter 4] or [8] has a good chance of giving useful bounds on the global optimum in a more predictable amount of time and memory than other general branch and bound algorithms.

7 Summary and Future Work

We have presented an alternative way of viewing solution of global optimization problems, in terms of discretization of selected variables, rather than in as an adaptive search. This leads to a simpler algorithm that is more easily parallelizable. An underlying idea in this scheme is redefining the problem in terms of a set of variables which, if of sufficiently small widths, guarantee that the problem is approximately convex. A methodology for determining a particular such set of variables, one that has a minimal cardinality over sets of nodes high in the computational graph, has been presented. Choosing the set high in the computational graph makes the constraint propagation to compute optimal solutions in terms of the original variables easier. However, more exploration into different choices of this set could yield better results for particular problems. For example, one might choose the set as low in the computational graph as possible: The resulting subproblems would then be convex, but the constraint propagation to determine optimal solutions in terms of the original variables would be more difficult. Alternatively, one might choose the set with nodes at varying levels in the computational graph, to try to minimize its cardinality; this may reduce the dimension of the space to be subdivided. The overall structure of the solution algorithm is

such that the total amount of work is more predictable than in traditional branch and bound algorithms, and issues of load balancing in a multiprocessing environment are less onerous.

Choosing parameters in the middle of the computational graph amounts to splitting solution of a particular subproblem into two parts: a part that is approximately convex and a part that is amenable to constraint propagation or that reveals singularity in the problem.

The implementation can be kept simple to facilitate parallelization. One use of this simple technique could be to generate starting boxes for a more sophisticated branch and bound algorithm. However, although the global minimum can be rigorously bounded with these techniques, the ϵ-approximate solution-containing boxes returned cannot be guaranteed to contain all possible optimizing points.

Acknowledgements

Above all, the first author wishes to thank Frédéric Messine at the Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et des Télécommunications for hospitality during his visit in December 2010. It was also a pleasure to interact face-to-face with Jordan Ninin, Pierre Hansen, Leo Liberti, Sonia Cafieri et al. Although the primary purpose of his visit was to serve on Jordan Ninin's Ph.D. jury, the entire week, full of talks, meetings, and discussions, was inspiring and intellectually stimulating.

References

1. T. G. W. Epperly and E. N. Pistikopoulos. A reduced space branch and bound algorithm for global optimization. *J. Global Optim.*, 11(3):287–311, October 1997.
2. R. Baker Kearfott. Erratum: Validated linear relaxations and preprocessing: Some experiments. *SIAM J. Optim.*, 2011.
3. R. Baker Kearfott. Interval computations, rigour and non-rigour in deterministic continuous global optimization. *Optim. Methods Softw.*, 2011. (to appear).
4. R. Baker Kearfott and Siriporn Hongthong. Validated linear relaxations and preprocessing: Some experiments. *SIAM J. Optim.*, 16(2):418–433, 2005.
5. Ralph Baker Kearfott. GlobSol user guide. *Optimization Methods and Software*, 24(4-5):687–708, August 2009.
6. Arnold Neumaier and Oleg Shcherbina. Safe bounds in linear and mixed-integer programming. *Math. Prog.*, 99(2):283–296, March 2004.
7. Jordan Ninin. *Optimisation Globale Basée sur l'Analyse d'Intervalles: Relaxations Affines et Techniques d'Accélération*. Ph.D. dissertation, Université de Toulouse, Toulouse, France, December 2010.
8. Jordan Ninin and Frédéric Messine. A metaheuristic methodology based on the limitation of the memory of interval branch and bound algorithms. *J. Glob. Optim.*, February 2010.
9. Julie Roy. *Singularities in Deterministic Global Optimization*. Ph.D. dissertation, Department of Mathematics, University of Louisiana, Lafayette, LA, USA, May 2010.
10. O. Shcherbina, A. Neumaier, D. Sam-Haroud, X.-H. Vu, and T.-V. Nguyen. Benchmarking global optimization and constraint satisfaction codes. In C. Bliek, C. Jermann, and A. Neumaier, editors, *COCOS*, volume 2861 of *Lecture Notes in Computer Science*, pages 211–222. Springer-Verlag, 2003.