Advances Implementation of ooOPS with spatial Branch and Bound Algorithm

Rajeev Kishore[1], Puneet Sethi[2], Arif Nadeem[3],

[1]Deptt. of Mathematics, Galgotia College of Engineering &Technology ,Gr.Noida,U.P.(India)
[2]Deptt. of Mathematics, TMU Moradabad U.P. (India)
[3]Deptt. of Mathematics, Bareilly College, Bareilly U.P. (India)

Abstract

In this paper we describes spatial Branch-and-Bound algorithm with advances Implementation of ooOPS. Spatial Branch-and-Bound is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization. A spatial branch-and-bound algorithm consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidates are discarded *en masse*, by using upper and lower estimated bounds of the quantity being optimized. *we turn our attention to the sBB algorithm itself and its computer implementation in a form that can be used as a component within larger software systems such as process modeling tools of Smith's sBB algorithm which forms the basis for our work and propose some algorithmic improvements to improve its performance. Lastly, we describe ooOPS (object-oriented Optimization System), a general software framework for defining and solving optimization problems, and the implementation of our algorithm within this framework.*

***Keywords**: object oriented optimization system, spatial branch and bound algorithm.*

Introduction

Branch and bound is a systematic method for solving optimization problems**.** *Branch-and-bound* is an approach developed for solving discrete and combinatorial optimization problems. The discrete optimization problems are problems in which the decision variables assume discrete values from a specified set; when this set is set of integers, we have an integer programming problem. The combinatorial optimization problems, on the other hand, are problems of choosing the best combination out of all possible combinations. Most combinatorial problems can be formulated as integer programs. The spatial Branch-and-Bound algorithm described in this paper solves NLPs in the following form:

$$Min_x g(x)$$
$$a \le f(x) \le b$$
$$\alpha \le x \le \beta$$

Where $x \in R^n$ are the (continuous) problem variables, $g : R^n \to R$ is the objective function (which may be non-convex), $f \in R^n \to R^m$ are a vector of generally non-convex functions, a and b are the lower and upper bounds of the constraints[1], and $\alpha and \beta$ are the lower and upper bounds of the variables.

The Branch-and-Reduce method [1] is an sBB algorithm with strong emphasis on variable range reduction. The αBB algorithm [2,3,4,5] is an sBB whose main feature is that the convex under estimators for general twice-differentiable nonconvex terms can be constructed automatically. The reduced-space Branch-and- Bound algorithm [6] identifies a *priori* a reduced set of branching variables so that less branching is required. The generalized Branch-and-Cut framework proposed in [7] derived cuts from violated constraints in three sub problems related to the original problems. Most sBB algorithms for the global optimization of non-convex NLPs conform to a general framework of the following form

1. (Initialization) Initialize a list of regions to a single region comprising the entire set of variable ranges. Set the convergence tolerance ε and the best objective function value $U = \infty$ Optionally, perform optimization-based bounds tightening.

2. (Choice of Region) If the list of regions is empty, terminate the algorithm with solution *U*. Otherwise, choose a region (the "current region") from the list. Delete this region from the list. Optionally, perform feasibility-based bounds tightening on this region.

3. (Lower Bound) Generate a convex relaxation of the original problem in the selected region and solve it to obtain an underestimation *l* of the objective function. If $l > U$ or the relaxed problem is infeasible, go back to step 2.

4. (Upper Bound) Attempt to solve the original (generally nonconvex) problem in the selected region to obtain a (locally optimal) objective function value *u*. If this fails, set $u = +\infty$.

5. (Pruning) If $U > u$, set $U = u$. Delete all regions in the list that have lower bounds bigger than U as they cannot possibly contain the global minimum.

6. (Check Region) If $u - l \leq \varepsilon$, accept *u* as the global minimum for this region and return to step2. Otherwise, we may not yet have located the region global minimum, so we proceed to the next step.

7. (Branching) Apply a branching rule to the current region to split it into sub-regions. Add these to the list of regions, assigning to them an (initial) lower bound of *l*. Go back to step 2

II. Smith's sBB Algorithm

The most outstanding feature of Smith's sBB algorithm is the automatic construction of the convex relaxation via symbolic reformulation [8,9]. This involves identifying all the non-convex terms in the problem and replacing them with the respective convex relaxations. The algorithm that carries out this task is symbolic in nature as it has to recognize the non-convex operators in any given function. Smith assumes that the NLP solved by his algorithm is of the form:

$$Min_x g(x) = 0$$

$$\bar{f}(x) = 0$$

$$\alpha \leq x \leq \beta$$

introducing slack variables to convert any inequalities to the equality constraints $\bar{f}(x) = 0$ above. Below, we consider some of the key steps of the algorithm in more detail.

2.1 Choice of region The region selection at step 2 follows the simple policy of choosing the region in the list with the lowest lower objective function bound as the one which is most promising for further consideration.

2.2 Convex relaxation The convex relaxation solved at step 3 of the algorithm aims to find a guaranteed lower bound to the objective function value. In most global optimization algorithms, convex relaxations are obtained for problems in special (e.g. factorable) forms. In Smith's sBB, the convex relaxation is calculated automatically for a problem provided this is available in closed analytic form, which allows symbolic manipulation to be applied to it. The convex relaxation is built in two stages: first the problem is reduced to a standard form where nonlinear terms of the same type are collected in lists; then each nonlinear term is replaced by the corresponding convex under- and over-estimators. The standard form also provides ways to simplify branch point calculation, branch variable choice and feasibility based bounds tightening.

2.3 Reformulation to standard form This is the first stage toward the construction of the convex relaxation of the original problem via symbolic manipulation of the variables and constraints. In this form, the problem nonlinearities are isolated from the rest of the problem and thus are easy to tackle by symbolic and numerical procedures. Smith defined the following standard form:

$$
\begin{aligned}
&Minx_{obj} \\
&l \le Ax \le u \\
&x_k = x_i x_j (i,j,k) \in M \\
&x_k = \frac{x_i}{x_k} \forall (i,j,k) \in D \\
&x_k = x_i^v \forall (i,k,v) \in P \\
&x_k = g_\mu (x_i) \forall (i,k,\mu) \in u \\
&x^L \le x \le x^U
\end{aligned}
$$

where X_{obj} = objective function and X = (x_1, x_2 ……x_n)are the problem variables,A is constant,usually sparse, matrix l,u are the linear constraint bounds,and X^L ,X^U are the variable bounds.

2.4 Convexification This is the second stage of the process where the actual convex relaxation of the original problem within the current region $\left[x^L, x^U\right]$ is built. The algorithm for convexification is entirely symbolic (as opposed to numeric) and hence performs very efficiently even in the presence of very complicated mathematical expressions. Having reduced a problem to standard form, we replace every non-convex term with a convex envelope consisting of convex over- and underestimating inequality constraints.

1. $X_i = X_j X_k$ is replaced by four linear inequalities (McCormick's envelopes)

$$x_i \geq x_j^L x_k + x_k^L x_j - x_j^L x_k^L$$
$$x_i \geq x_j^U x_k + x_k^U x_j - x_j^U x_k^U$$
$$x_i \leq x_j^L x_k + x_k^U x_j - x_j^L x_k^U$$
$$x_i \leq x_j^U x_k + x_k^L x_j - x_j^U x_k^L$$

2. $X_i = X_j / X_k$ is reformulated to $X_i X_k = X_J$ and the convexification rules for above bilinear terms are applied.

3. $X_i = g_\mu(X_j)$ where g_μ is concave univariate is replaced by two inequalities:such as

$$x_i \leq g_\mu(x_j)$$
$$x_i \geq g_\mu(x_j^L) + \frac{g_\mu(x_j^U) - g_\mu(x_j^L)}{x_j^U - x_j^L}(x_j - x_j^L)$$

4. $X_i = g_\mu(X_j)$ where g_μ is convex univariate is replaced by:

$$x_i \geq g_\mu(x_j)$$

$$x_i \leq g_\mu(x_j^L) + \frac{g_\mu(x_j^U) - g_\mu(x_j^L)}{x_j^U - x_j^L}(x_j - x_j^L)$$

5. $X_i=(X_j^v)$ where $0<v<1$ is treated as a concave univariate function in the manner described in point 3 above.

6. $X_i=(X_j^{2m})$ for any m€N is treated as a convex univariate function in the manner described in point 4 above.

2.5 Branching There are many branching strategies [10] available for use in spatial Branch-and-Bound algorithms. Generally branching involves two steps, namely determining the point (i.e. set of variable values) on which to branch, and finding the variable whose domain is to be sub-divided by the branching operation. Smith uses the solution of the upper bounding problem (step 4) as the branching point, if such a solution is found; otherwise the solution of the lower bounding problem (step 3) is used. He then identifies the nonlinear (non-convex) term with the largest error with respect to its convex relaxation. The branch variable is chosen as the variable whose value at the branching point is nearest to the midpoint of its range.

2.6 Bounds tightening These procedures arrear in steps 1 and 2 of the algorithm structure. They are optional in the sense that the algorithm will, in principle, converge even without them. Depending on how computationally expensive and how effective these procedures are, in some cases convergence might actually be faster if these optional steps are not performed. In the great majority of cases, however, the bounds tightening steps are essential to achieve fast convergence.

The ooOPS Software Framework for Optimization

Any computer implementation of general-purpose spatial Branch and Bound algorithms for global optimization requires a large amount of information on the problem being solved. This includes:

- Numerical information of the type required by local optimization solvers, e.g. values of the objective function and the residuals of the constraints for given values of the problem variables;

- Structural information, e.g. regarding the sparsity pattern of the constraints; this is essential not only for improving the code efficiency (a feature that is also shared with local optimization codes), but also in implementing advanced techniques such as the model reformulation algorithm.

- Symbolic information on the objective function and constraints, needed for implementing general reformulation and/or convexification techniques.

Most implementations of sBB-type algorithms in existence today have been either standalone software codes (e.g. the αBB system [11,12] with their own language for defining optimization problems, or have been embedded within existing modeling tools which provide them with the necessary information; for example, Smith's [13] code was embedded in the gPROMS modeling tool [14] while the BARON code [15] has recently been made available within GAMS [16]. Whilst these developments are undeniably useful from the immediate practical point of view, the wider dissemination of global optimization technology requires a different approach which allows the software to be directly embedded within larger software systems such as mathematical modeling tools, domain-specific optimization codes (e.g. for pooling and blending) and so on.
The above considerations provided the motivation for the development of the *ooOPS* (object oriented Optimization System) system in the context of the work described in this paper. *ooOPS* is a comprehensive library of callable procedures for the definition, manipulation and solution of large, sparse nonlinear programming problems.

Main features of *ooOPS*

Rather than being a stand-alone code, *ooOPS* is designed to provide a number of high-level services to a client software code (e.g. written in C++, C or FORTRAN) via an Application Programming Interface (API). Its main features are as follows:

- *ooOPS* allows its client codes to construct and, if necessary, subsequently modify large scale NLPs involving large sets of variables and linear and nonlinear constraints.
- The construction of complex problems is facilitated by recognizing that, in most practical applications, variables and constraints can be categorized into relatively small numbers of sets which can be defined generically. Thus *ooOPS* allows variables and constraints to be defined in a "structured" fashion as multi-dimensional arrays with an arbitrary number of dimensions.
- The construction of an NLP in *ooOPS* is done in a symbolic manner. Thus, each nonlinear expression occurring in the objective function and/or the constraints is built by the client issuing a sequence of calls (i.e. on a term-by-term, factor-by-factor basis etc.). Consequently, *ooOPS* is fully aware at all times of the symbolic form of any NLP within it and can supply this information to its clients.

- It allows a client to manipulate any number of NLPs simultaneously; this is important for supporting applications which require iterating between two or more optimization problems, with information derived from the solution of one of these NLPS being used to define or modify one or more of the others.

- It allows its clients to evaluate the objective function and constraints of any NLP held within it. Moreover, it automatically derives, and makes available to its clients, structural (e.g. sparsity pattern) and symbolic (e.g. exact partial derivatives) information on these NLPs. An algorithm for fast evaluation of symbolic expressions has been devised and implemented.

An sBB solver for the *ooOPS* framework

The sBB algorithm of Smith [17, 18 and 19] for global optimization with the minor improvements has been implemented for use within the *ooOPS* framework. The sBB implementation makes use of two sub-solvers: a local NLP solver for obtaining upper bounds by solving the original NLP in each region; and an optimization solver for obtaining lower bounds by solving the convex relaxation of the original NLP in each region. Each of these sub-solvers is itself implemented as an *ooOPS*-compliant optimization solver. In addition, the sBB code makes use of a convexifier, a software component which, given an ops object, constructs another ops object that represents a convex relaxation of the former. More specifically, given an ops object describing a non-convex NLP, our sBB solver executes the following sequence of operations:

1- Create an **opssolvermanager** for the local solver which will be used to solve the upper bounding problem.

2- Pass the original ops object to the above local **opssolvermanager** to create an upper bounding opssystem.

3- Create a **convexifiermanager**.

4- Create a new ops object representing a convex relaxation of the original ops object by passing the latter to the above **convexifiermanager**.

5- Create an **opssolvermanager** for the optimization solver to be used for the solution of the lower bounding problem.

6- Create a lower bounding **opssystem** by passing the ops object constructed at step 4 to the **opsslvermanager** constructed at step 5.

7- During the branch-and-bound search:

(a) Repeatedly call the Solve methods of the **opssystems** constructed at steps 2 and 6 to solve the upper and lower bounding problems respectively.

(b) On changing the variable ranges during branching, update the lower bounding problem by involving the **UpdateConvexVarBounds** ( ) method in the **convexifiermanager**.

8. De-allocate all objects created by the global solver code.

As can be seen, the above procedure takes advantage of *ooOPS*'s capabilities to simultaneously manipulate two ops objects respectively describing the original NLP problem and its convex relaxation. Our current sBB implementation uses SNOPT [19] for solving the upper bounding problem, and the CPLEX [20] linear programming code as the solver for the convex relaxation. It is worth noting,

however, that, in principle, any *ooOPS*-compliant local NLP solver can be used as a sub-solver within our sBB code.

IV. Conclusion

In this paper, we presented an outline of the symbolic reformulation spatial Branch-and Bound (sBB) algorithm by Smith [8,9] and proposed some minor improvements to it. We also considered the software implementation of general sBB algorithms, and the requirements that this imposes in terms of numeric, structural and symbolic information on the NLP being solved. Our analysis led us to the development of *ooOPS* a general software framework for optimization that can support both local and global optimization solvers.

A key concept in the design of *ooOPS* is the separation of the NLP problem being solved from the optimization solver itself; the former is described by a software object ops while the latter is implemented as a software component exposing an opssolvermanager object interface. The combination of a solver manager with ops object leads to the creation of an opssystem object which can be solved by the invocation of an appropriate method. We borrowed all of these ideas from the design of standardized software components for the solution of sets of nonlinear algebraic and mixed differential-algebraic equations in the CAPE-OPEN project, an international initiative for the standardization of process engineering software. Having extended these concepts to the solution of nonlinear optimization problems, we have contributed some of them back to the CAPE-OPEN initiative to form the basis of a new standard for optimization solvers [21].

It is worth pointing out, however, that our current sBB implementation is rather basic without many of the sophisticated implementation details that accelerate the performance of such algorithms (e.g. good range- reduction techniques, improved branching procedures etc.). Nevertheless, *ooOPS* provides an open architecture within which more sophisticated implementations may be embedded in the future.

References:

[1]. H. Ryoo and N. Sahinidis. Global optimization of non-convex NLPs and MINLPs with applications in process design. Computers & Chemical Engineering, 19:551–566, 1995.

[2]. C. Adjiman, S. Dallwig, C. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs: I. Theoretical advances. Computers & Chemical Engineering, 22:1137–1158, 1998.

[3]. C. Adjiman, I. Androulakis, and C. Floudas. A global optimization method, αBB, for general twice-differentiable constrained NLPs: II. Implementation and computational results. Computers & Chemical Engineering, 22:1159–1179, 1998.
.
[4]. C. Adjiman, I. Androulakis, C. Maranas, and C. Floudas. A global optimization method, αBB, for process design. Computers & Chemical Engineering, 20:S419–S424, 1996.
[5]. I. Androulakis, C. Maranas, and C. Floudas. α BB: A global optimization method for general constrained non-convex problems. Journal of Global Optimization, 7:337–363, 1995.

[6]. T. Epperly and E. Pistikopoulos. A reduced space branch and bound algorithm for global optimization. Journal of Global Optimization, 11:287:311, 1997.

[7]. H. Ryoo and N. Sahinidis. Global optimization of non-convex NLPs and MINLPs with applications in process design. Computers & Chemical Engineering, 19:551–566, 1995.

[8]. E. Smith. On the Optimal Design of Continuous Processes. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, October 1996.

[9]. E. Smith and C. Pant elides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimization of nonconvex MINLPs. Computers and Chemical Engineering, 23:457–478, 1999.

[10]. T. Epperly. Global Optimization of Nonconvex Nonlinear Programs using Parallel Branch and Bound. PhD thesis, University of Wisconsin – Madison, 1995.

[11]C. Adjiman, S. Dallwig, C. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs: I. Theoretical advances. Computers & Chemical Engineering, 22:1137–1158, 1998.

[12]. C. Adjiman, I. Androulakis, and C. Floudas. A global optimization method, αBB, for general twice-differentiable constrained NLPs: II. Implementation and computational results. Computers & Chemical Engineering, 22:1159–1179, 1998.

[13]. E. Smith. On the Optimal Design of Continuous Processes. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, October 1996.

[14]. N. Sahinidis. Baron: Branch and reduce optimization navigator: User's manual, version 4.0. http://archimedes.scs.uiuc.edu/baron/manuse.pdf, 1999.

[15]. Process Systems Enterprise. GPROMS v2.2 Introductory User Guide. Process Systems Enterprise, Ltd., London, UK, 2003.

[16]. A. Brook, D. Kendrick, and A. Meeraus. Gams, a user's guide. ACM SIGNUM Newsletter, 23:10–11, 1988.
[17]. E. Smith. On the Optimal Design of Continuous Processes. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, October 1996.

[18]. E. Smith and C. Pant elides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimization of nonconvex MINLPs. Computers and Chemical Engineering, 23:457–478, 1999.

[19]. P.Gill. User's Guide for SNOPT 5.3. Systems Optimization Laboratory, Department of EESOR, Stanford University, California, 1999.

[20]. ILOG. ILOG CPLEX 8.0 User's Manual. ILOG S.A., Gentilly, France, 2002.

[21]. C. Pantelides, L. Liberti, P. Tsiakis, and T. Crombie. Mixed integer linear/nonlinear program- ming interface specification. Global Cape-Open Deliverable WP2.3-04, 2002.

.