A Rigorous Generic Branch and Bound Solver for Nonlinear Problems

Andrew P. Smith
National Institute of Aerospace
Hampton, Virginia
Email: andrew.smith@nianet.org

César A. Muñoz and Anthony J. Narkawicz
NASA Langley Research Center
Hampton, Virginia
Email: {cesar.a.munoz,anthony.narkawicz}@nasa.gov

Mantas Markevicius
University of York,
York, UK
Email: mm1080@york.ac.uk

***Abstract*—Recursive branch and bound algorithms are often used, either rigorously or non-rigorously, to refine and isolate solutions to global optimization problems or systems of equations and inequalities involving nonlinear functions. The presented software library, Kodiak, integrates numeric and symbolic computation into a generic framework for the solution of such problems over hyper-rectangular variable and parameter domains. The correctness of both the generic branch and bound algorithm and the self-validating enclosure methods used, namely interval arithmetic and, for polynomials and rational functions, Bernstein expansion, has been formally verified. The algorithm has three main instantiations, for systems of equations and inequalities, for constrained global optimization, and for the computation of equilibria and bifurcation sets for systems of ordinary differential equations. Advantage is taken of the partial derivatives of the constraint functions, which are symbolically manipulated. Pavings (unions of box subsets) for a continuum of solutions to underdetermined systems may also be produced. The capabilities of the software tool are outlined, and computational examples are presented.**

***Keywords*-branch and bound; nonlinear problems; formal verification; interval arithmetic; software tool;**

I. Introduction

Many categories of problems involving nonlinear functions prove to be intractable for algebraic techniques, due to the number of variables involved or the complexity and unsuitability of constraint functions. Numerical methods may be widely applied to these problems, but in general only compute an approximation to the solution. Self-validated numerical methods typically compute a safe interval enclosure for intermediate values at each step, and thus produce a guaranteed result, albeit at the cost of extra computational effort and possible loss of precision. Where the ranges for variables and parameters are too wide to obtain meaningful results in a single step, it is necessary to partition the search space. Branch and bound is a method for recursively partitioning the starting space into sub-domains, over which local solutions can be more tightly enclosed, allowing for successive refinement of the overall solution set.

The most common type of branch and bound solvers are designed for constrained global optimization; for an overview of interval approaches, see [1]. These include commercial solvers, e.g., [2], and rigorous interval solvers, e.g., [3]. There also exist branch and bound approaches for systems of equations or inequalities, e.g., [4], [5].

This paper summarizes a software development in C++, called Kodiak[1], and associated research. The core algorithm closely follows the formally verified depth-first branch and bound algorithm with generic types for problem domains and solution types is described in [6]. That algorithm is the basis of a family of proof-producing strategies for the PVS theorem prover [7] that automatically discharge simply quantified Boolean expressions over real numbers.

The following notation is used: $\mathbb{IR}$ denotes the set of closed non-empty intervals with real endpoints. A member of this set is written as $\mathbf{x} = [\underline{x}, \overline{x}] \in \mathbb{IR}$. A Cartesian product of n intervals, a hyper-rectangle or *box*, is written as $\mathbf{X} = [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n] \in \mathbb{IR}^n$. A set of nearly-disjoint boxes of the same dimension, where any two different boxes may intersect only at their boundaries, is termed a *paving*. A *valid paving* for a set $S \in \mathbb{R}^n$ is such that the union of all its members is a superset of S. Henceforth, it is assumed that S is specified by a set of equalities and inequalities involving real-valued functions and variables ranging over a box $\mathbf{X} \in \mathbb{IR}^n$, where n is the number of variables.

II. Enclosure Methods

It is necessary to compute guaranteed upper and lower bounds for a family of functions $f : \mathbb{R}^n \to \mathbb{R}$ over a box $\mathbf{X}$. An *interval extension* for f is an interval-valued function, $\mathbf{f} : \mathbb{IR}^n \to \mathbb{IR}$, s.t. $\forall \mathbf{X} \in \mathbb{IR}^n : x \in \mathbf{X} \implies f(x) \in \mathbf{f}(\mathbf{X})$.

The most straightforward way of performing arithmetic with intervals on a computer is to use the natural interval extension for each real-valued function, although numerous enclosure methods exist. There are standard interval definitions for logarithmic and trigonometric functions, exponentiation, square root, etc., as well as for relational operators. For an introduction to interval arithmetic, see [8]; a comprehensive list of interval operations may be found in [9]. Kodiak utilizes the C++ library filib++ [9], which efficiently implements these operations, using direct floating-point rounding modes to assure soundness.

The usual arithmetic laws for real numbers must be relaxed whenever an independent variable appears more than once in an interval expression, a phenomenon known as

[1]Kodiak is released under NASA's Open Source Agreement. It is electronically available from `http://github.com/nasa/Kodiak`.

the *dependency problem*. For example, the distributive law becomes: $\mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) \subseteq \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}$ for $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{IR}$. As a consequence of the dependency problem, natural interval extensions for functions with long expressions can exhibit a significant amount of overestimation, which must be overcome with repeated subdivision of variable ranges.

If the set of constraints defining S involve only polynomial or rational functions, there is available in Kodiak the option to use the Bernstein enclosure in place of interval arithmetic. This exhibits second-order convergence to the true range as interval widths tend to zero, as opposed to first-order convergence for interval arithmetic. A univariate polynomial p of degree d is typically presented in power form as a sum of terms $p(x) = \sum_{i=0}^{d} a_i x^i$, where the a_i, $i = 0, \ldots, d$, are the usual power-form coefficients. The same polynomial may be rewritten in Bernstein form, i.e., $p(x) = \sum_{i=0}^{d} b_i B_i(x)$, where the $B_i(x)$, $i = 0, \ldots, d$ are the set of $d + 1$ Bernstein basis polynomials, forming a basis for the vector space of polynomials of degree d, and where the b_i are the Bernstein coefficients.

For function approximation, a key attribute of the Bernstein expansion is the range enclosing property, namely that the range of p over the unit interval is contained within the interval hull of the Bernstein coefficients. For other intervals, the polynomial (and the Bernstein basis polynomials) can be affinely transformed, and the formulae can readily extended to the multivariate case, cf. [10]. Therefore the rigorous evaluator in Kodiak for p over an interval or box consists of a computation of all the Bernstein coefficients and determination of their minimum and maximum.

Kodiak makes use of an efficient implicit representation for the Bernstein coefficients, whereby it is often possible to avoid computing many of the explicit coefficients by use of monotonicity properties; for further details and references, see [10]. A description of the computation of the Bernstein enclosure for rational functions is given in [11].

III. Branch and Bound

The objective of the method is to compute a paving for a set S that is guaranteed to be valid and close to S up to a given accuracy. There are two key steps:

- A *branching* step, which partitions the current sub-box into two or more smaller sub-boxes.
- A *bounding* (or pruning) step, where sub-boxes are either safely discarded, when they are proven not to contain a solution, or retained, when they may either possibly or definitely contain a solution.

This yields a search space structured in the form of a tree, which is traversed according to a recursive depth-first strategy. For most instantiations where the entire tree must be traversed, this is memory-efficient. This does not necessarily hold where boxes may additionally be pruned according to optimality, e.g., for global optimization. After sufficient branching, many of the sub-boxes can typically be excluded from the search, until a satisfactory paving, possibly empty, is obtained. If it is empty, it holds that S is empty.

A subdivision strategy specifies the variable(s) in which a sub-box is to be subdivided and whether this subdivision results in sub-boxes of the same size or not. By default, a round-robin strategy may be used. Heuristics based upon enclosures for the partial derivatives of the constraint functions [5] can be used to somewhat reduce the size of the search tree.

Termination criteria are generally required, which stipulate when sub-boxes become satisfactorily small, or the computed paving is close to S for a given accuracy such that no further subdivision is required. Additionally, a maximum search depth may be specified. The output of the method can be computed from the final paving.

Algorithm 1 Generic depth-first branch and bound algorithm

Input: expression, box **Global:** exit
Output: answer
1: answer := `bound`(expression, box)
2: exit := exit **or** `global_exit`(answer)
3: **if** box is empty **or** maximum depth reached **or** exit **or** `prune`(answer) **then**
4: **return**
5: **end if**
6: direction := `select`(expression, box)
7: (box_1, box_2) := `split`(direction, box)
8: expression_1 := `branch`(direction, expression, box_1)
9: answer_1 := first recursive call with arguments expression_1 and box_1
10: **if** exit **then**
11: answer := `combine`(answer, answer_1)
12: **return**
13: **end if**
14: expression_2 := `branch`(direction, expression, box_2)
15: answer_2 := second recursive call arguments expression_2 and box_2
16: answer := `combine`(answer_1, answer_2)
17: **return**

IV. Design and use of Kodiak

The pseudo-code of the main generic branch and bound procedure is listed in Algorithm 1. It takes as input an expression, which is a symbolic representation of a problem such as a system of inequalities, and a box, which is a list of intervals, each one representing the range of a variable. The algorithm produces an answer of generic type. An over-approximation to the solution of the input problem, possibly crude, is firstly computed by calling the function `bound`. Some instantiations of the branch and bound method require an early termination condition, which is facilitated by the global variable `exit` and the function `global_exit`. Pruning at every recursive step is achieved by the function `prune`. If none of the conditions for blocking the recursive call are satisfied, a variable direction is chosen by the function `select` and corresponding sub-boxes are computed by the function `split`. The function `branch` adjusts the input expression to each one of the sub-boxes according to the variable direction. Finally, the function `combine` accumulates the answers computed at each recursive call into the return value.

There are currently available three instantiations of the core algorithm. The first is for paving the solution set of a system of equations and inequalities. A system of relations is a collection of j formulas of the form $f_i(\mathbf{x})\ R_i\ 0$, where $0 \leq i \leq j$, $f_i : \mathbb{R}^n \to \mathbb{R}$, and R_i is a real-order relation in $\{<, \leq, >, \geq, =\}$. The solution set S consists of all points in $\mathbb{R}^n$ that simultaneously satisfy all constraints. In non-degenerate cases, it exhibits $n-l$ degrees of freedom, where l is the number of equalities in the system. In general, where S lacks an algebraic description, it is desired to compute reliable over- and under-approximations of S over a box $\mathbf{X}$.

The first output paving is a guaranteed inner approximation of S, where every point in each member box definitely satisfies all constraints. The second consists of candidate solution boxes in which it is possible that some points satisfy all constraints, and it is possible that some points violate at least one constraint. Together, these comprise a guaranteed outer approximation of S. If desired, a third output paving for the complement of S can be produced, where every point in each box definitely violates at least one constraint.

Generally, if the system of relations has equalities, the first paving is empty. In non-degenerate cases where $l = n$, zero or more point solutions may exist, and the solution paving consists of one or more boxes of terminal width, some of which enclose each individual solution. Where $l < n$, i.e., for underdetermined systems, there is a (possibly empty) continuum of solutions, with $n-l$ degrees of freedom. Only in the latter case does the number of boxes in the solution paving increase with search depth.

The second instantiation is for the bifurcation analysis of a system of parameterized ordinary differential equations. One may first compute a guaranteed paving for the set of state-parameter combinations corresponding to an equilibrium, either stable or unstable. Here, the procedure is the same as for a system of equations, where variables and parameters are treated equally with respect to bisection. Variations in the parameter vector may effect quantitative and qualitative changes in the equilibrium points. A local bifurcation is a point in the state-parameter space at which a transition in the number or type of equilibria occurs. Knowledge of their location is important for the stability analysis of nonlinear dynamic systems . Sufficient tests for the existence of two types of local bifurcation are implemented; in either case, the existing system of equations is augmented with additional automatically-derived constraints, which remove one degree of freedom, and the same core paving procedure is performed.

Steady-state bifurcations correspond to state-parameter values where the Jacobian matrix of the system is singular. Its determinant is symbolically computed and the system is augmented with the associated constraint. Hopf bifurcations, which originate limit cycles, occur where the characteristic polynomial associated with the aforementioned Jacobian matrix has a conjugate pair of complex solutions with zero real part and all other roots have a negative real part. The coefficients of the characteristic polynomial, as symbolic expressions, are assembled by Kodiak into a so-called Hurwitz matrix, and boolean combinations of sign conditions on the determinants of its minors are used as sufficient conditions for a Hopf bifurcation. Further details and references may be found in [12].

The third instantiation is for constrained global optimization problems, in which an objective function $f : \mathbb{R}^n \to \mathbb{R}$ is added to the above system of relations. It is often desired to compute both an interval enclosure for the minimum value and a paving for the minimizers of f, i.e., the points in $S \cap \mathbf{X}$ at which the minimum is attained.

In Kodiak, the constrained global optimization problem is solved by adding a local exit condition to the pruning strategy, i.e., boxes are pruned with respect to optimality, based upon a computed enclosure for the objective function, as well as feasibility. This requires an upper bound for the minimum to be stored and updated during the recursion.

V. Summary of Features

Kodiak provides a concrete representation of real number expressions; it supports numerical literals, which can be either rational, decimal, or machine floating-point numbers, symbolic variables, symbolic parameters, user-defined constants, the mathematical constants π and e, the four basic arithmetic operations, the power operator, and a collection of real-valued functions such as absolute value, square root, trigonometric functions and their inverses, exponential, and natural logarithm. Inexact values (floating-point numbers) are input and stored as safe intervals. When constructing a symbolic expression, Kodiak automatically performs basic arithmetic simplifications. Datatypes for relations and systems of relations are also included.

Guaranteed (sound) results are obtained by the exclusive use of interval arithmetic and interval variables in place of floating-point, throughout the branching and bounding steps. This claim relies upon sound hardware and component libraries that are bug-free. Polynomials (in power form) and rational functions are detected automatically, and such expressions can be rewritten into Bernstein form using an implicit representation of Bernstein coefficients. The partial derivatives of constraint functions are symbolically computed, with basic simplification. Their enclosures enable the tightening of intervals for constraint functions, where monotonicity with respect to one or more variables is proven. Furthermore, for global optimization problems, in the case of boxes that are feasible everywhere, monotonicity allows the box to be reduced in *dimension*, or, if a previous subdivision assures the existence of a suitable neighbor, eliminated entirely. The heuristic for the choice of subdivision variable assigns weight in proportion to normalized (with respect to the starting box) box width in each variable and to an estimate of normalized change in each involved function

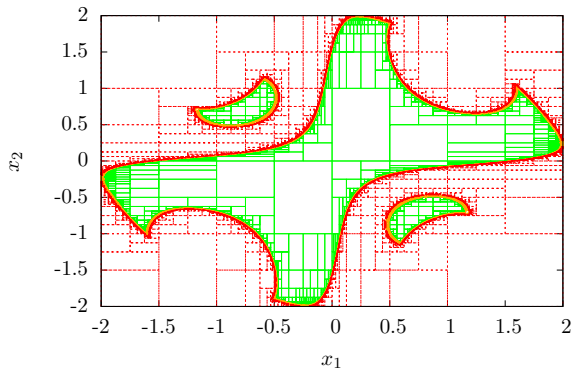

Figure 1. Output pavings for Example 1: boxes that are certainly feasible (green), possibly feasible (orange), and certainly infeasible (red)

over the box with respect to that variable. Solution sets for underdetermined systems with one or more degree of freedom may be paved.

VI. Examples

Example 1. The safe domain for a certain control system is given by a small system of two polynomial inequalities in two variables [13]:

$$\begin{aligned} x_1^2x_2^4 + x_1^4x_2^2 - 3x_1^2x_2^2 - x_1x_2 + \frac{x_1^6 + x_2^6}{200} - \frac{7}{100} &\leq 0 \\ -\frac{x_1^2x_2^4}{2} - x_1^4x_2^2 + 3x_1^2x_2^2 + \frac{x_1^5x_2^3}{10} - \frac{9}{10} &\leq 0 \end{aligned}$$

The starting box is $[-2, 2] \times [-2, 2]$ and the maximum depth is set to 20. On a 3 GHz Intel Xeon PC, using interval arithmetic, the three output pavings, depicted in Figure 1, consisting of a few thousand boxes each, are computed in 3 s. Approximately 78000 bisections are required overall; use of Bernstein enclosure reduces this by a factor of three but increases the total computation time to 13 s.

Example 2. Kodiak has been successfully applied to the bifurcation analysis of a detailed model for the longitudinal dynamics of a jet airliner [12]. There are highly non-trivial functions in four variables and five parameters; the longest symbolic expressions for the coefficients of the characteristic polynomial and the Hurwitz determinants occupy several lines and about a page, respectively. In the studied test case where two parameters are free, the bifurcation set has one degree of freedom in 6D space; good-quality pavings for both bifurcation categories are computed in about two hours, requiring seven million bisections. In the 9D case where all five parameters are allowed to vary simultaneously, a sizable guaranteed exclusion box can be computed.

Acknowledgment

Funding of the first author's research under NASA Cooperative Agreement NNL09AA00A is gratefully acknowledged.

References

[1] E. R. Hansen and G. W. Walster, *Global Optimization Using Interval Analysis*, 2nd ed. New York, Basel: Marcel Dekker, Inc., 2004.

[2] N. V. Sahinidis, "BARON: A general purpose global optimization software package," *J. Global Optimization*, vol. 8, no. 2, pp. 201–205, 1996.

[3] R. B. Kearfott, "GlobSol user guide," *Optimization Methods and Software*, vol. 24, no. 4-5, pp. 687–708, August 2009.

[4] L. Granvilliers and F. Benhamou, "Algorithm 852: RealPaver: An interval solver using constraint satisfaction techniques," *ACM Trans. on Mathematical Software*, vol. 32, no. 1, pp. 138–156, 2006.

[5] J. Garloff and A. P. Smith, "Investigation of a subdivision based algorithm for solving systems of polynomial equations," *J. of Nonlinear Analysis: Series A Theory and Methods*, vol. 47, no. 1, pp. 167–178, 2001.

[6] A. Narkawicz and C. Muñoz, "A formally verified generic branching algorithm for global optimization," in *Fifth Working Conference on Verified Software: Theories, Tools and Experiments (VSTTE)*, ser. Lecture Notes in Computer Science, E. Cohen and A. Rybalchenko, Eds., vol. 8164, 2014, pp. 326–343.

[7] S. Owre, J. Rushby, and N. Shankar, "PVS: A prototype verification system," in *Proceedings of the 11th International Conference on Automated Deduction — CADE-11*, ser. Lecture Notes in Artificial Intelligence, D. Kapur, Ed., vol. 607. Springer, June 1992, pp. 748–752.

[8] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*. Philadelphia: SIAM, 2009.

[9] M. Lerch, G. Tischler, J. Wolff von Gudenberg, W. Hofschuster, and W. Krämer, "filib++, a fast interval library supporting containment computations," *ACM Trans. on Mathematical Software*, vol. 32, no. 2, pp. 299–324, 2006.

[10] A. P. Smith, "Fast construction of constant bound functions for sparse polynomials," *J. Global Optimization*, vol. 43, no. 2–3, pp. 445–458, 2009.

[11] A. Narkawicz, J. Garloff, A. P. Smith, and C. A. Muñoz, "Bounding the range of a rational function over a box," *Reliable Computing*, vol. 17, pp. 34–39, 2012.

[12] A. P. Smith, L. G. Crespo, C. A. Muñoz, and M. H. Lowenberg, "Bifurcation analysis using rigorous branch and bound methods," in *2014 IEEE International Conference on Control Applications (CCA)*, ser. Part of 2014 IEEE Multi-conference on Systems and Control, Antibes, France, October 2014, pp. 2095–2100.

[13] L. G. Crespo, C. A. Muñoz, A. J. Narkawicz, S. P. Kenny, and D. P. Giesy, "Uncertainty analysis via failure domain characterization: Polynomial requirement functions," in *Proceedings of European Safety and Reliability Conference*, Troyes, France, September 2011.