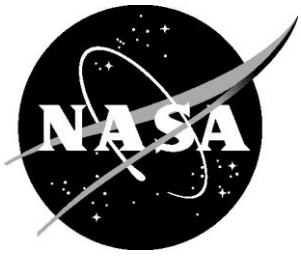
NASA/TM–2015-218776



Kodiak: An Implementation Framework for Branch and Bound Algorithms

Andrew P. Smith
National Institute of Aerospace, Hampton, Virginia

César A. Muñoz and Anthony J. Narkawicz
Langley Research Center, Hampton, Virginia

Mantas Markevicius
University of York, York, United Kingdom

July 2015

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

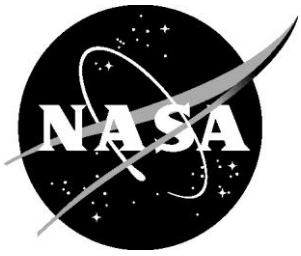
- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM–2015-218776



Kodiak: An Implementation Framework for Branch and Bound Algorithms

Andrew P. Smith
National Institute of Aerospace, Hampton, Virginia

César A. Muñoz and Anthony J. Narkawicz
Langley Research Center, Hampton, Virginia

Mantas Markevicius
University of York, York, United Kingdom

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

July 2015

Acknowledgments

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

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

Recursive branch and bound algorithms are often used to refine and isolate solutions to several classes of global optimization problems. A rigorous computation framework for the solution of systems of equations and inequalities involving nonlinear real arithmetic over hyper-rectangular variable and parameter domains is presented. It is derived from a generic branch and bound algorithm that has been formally verified, and utilizes self-validating enclosure methods, namely interval arithmetic and, for polynomials and rational functions, Bernstein expansion. Since bounds computed by these enclosure methods are sound, this approach may be used reliably in software verification tools. Advantage is taken of the partial derivatives of the constraint functions involved in the system, firstly to reduce the branching factor by the use of bisection heuristics and secondly to permit the computation of bifurcation sets for systems of ordinary differential equations. The associated software development, Kodiak, is presented, along with examples of three different branch and bound problem types it implements.

1 Introduction

Branch and bound is a numerical computation method for successive refinement of a solution set over a bounded domain, whereby the starting search space is recursively partitioned (*branching*) into sub-domains, over which solutions can be more tightly enclosed (*bounding*) and then combined. This method yields a search tree, where every branch in the tree represents a sub-domain of the original problem. Some of these sub-domains are provably inconsistent with the solution sought and may thus soundly be pruned from the tree. This branching and bounding process leaves zero or more sub-domains remaining as solution candidates for the original problem. A formally verified depth-first branch and bound algorithm with generic types for problem domains and solution types is presented in [1]. That algorithm is the basis of a family of proof-producing strategies for the PVS theorem prover [2] that automatically discharge singly quantified Boolean expressions over real numbers.

For many applications, a suitable starting search domain is a box (hyperrectangle) subset of $\mathbb{R}^n$, which is bounded and connected, where n is the number of variables. This starting box is recursively subdivided into sub-boxes, typically by performing a single bisection of one of the component intervals at each branch. Over each box, outer approximations for the range of a real-valued function may be computed by employing a suitable enclosure method. Boxes that are thereby proven not to contain a solution to the problem are discarded. Several kinds of problems involve a set of constraints that can be expressed by a predicate formed from simple Boolean combinations of relational expressions involving nonlinear functions over $\mathbb{R}^n$. If both the branching and bounding steps are implemented rigorously, so as to exclude numerical errors, the method may be used reliably to attempt formal proofs of theorems asserting the validity or satisfiability of such formulas. The method can also be used to compute guaranteed upper and lower bounds of real-valued multivariate functions over box domains subject to constraints, a type of global optimization problem. Both applications are common in the verification and analysis of safety-critical systems that interact with the physical environment.

This paper also presents a software development, called Kodiak, that closely follows the formally verified PVS branch and bound algorithm described in [1]. The software implementation takes advantage of the genericity of the PVS algorithm to propose a number of refinements for specific instantiations of the algorithm. In particular, Kodiak implements several pruning strategies based on the monotonicity properties of the expressions involved in the original problem.

The rest of this paper is organized as follows. Section 2 provides an overview of the branch and bound method and two rigorous enclosure techniques, interval arithmetic and Bernstein expansion. The generic branch

and bound algorithm, and its implementation in Kodiak, is detailed in Section 3. Section 4 explores three specific instantiations of the branch and bound method, with examples using the Kodiak library. The paper concludes with a summary and directions for future work.

2 Branch and Bound and Enclosure Methods

Let $\mathbb{IR}$ be 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.

The objective of the branch and bound method presented in this paper is to compute a paving for a set S that is guaranteed to be valid and close to S up to a given accuracy. The method proceeds by recursively dividing the original box into sub-boxes, yielding a search space structured in the form of a tree, which is traversed according to a specified strategy, e.g., recursive depth-first search, breadth-first, or a mixed strategy. Sooner or later, many of the sub-boxes can typically be excluded from the search, until a satisfactory paving, possibly empty, is obtained. If the paving is empty, it holds that S is empty.

The two essential steps in a branch and bound algorithm are:

- A *subdivision* (or branching, splitting) 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.

Some approaches also incorporate a *contraction* step, whereby a sub-box is reduced in size, by shrinking one or more of its component intervals, e.g., by the use of constraint propagation techniques [3, 4], prior to performing a subdivision.

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. 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. At the end, zero or more sub-boxes of small size remain and these boxes are used to compute the output. The output of the method depends on the specific problem. It can be either a concrete paving, the search tree, or any other information derived from the paving.

At the core of the bounding step there is an approach for computing guaranteed upper and lower bounds for a family of functions $f : \mathbb{R}^n \to \mathbb{R}$ over a box $\mathbf{X}$. An obvious choice for this approach is interval arithmetic [5], but other enclosure methods are possible. An *interval extension* for f is an interval-valued function, $\mathbf{f} : \mathbb{IR}^n \to \mathbb{IR}$, such that

$$\forall \mathbf{X} \in \mathbb{IR}^n \colon x \in \mathbf{X} \implies f(x) \in \mathbf{f}(\mathbf{X}). \tag{1}$$

Furthermore, to be used in the context of branch and bound, an enclosure method has to satisfy the property of inclusion isotonicity.

$$\forall \mathbf{X}, \mathbf{Y} \in \mathbb{IR}^n \colon \mathbf{X} \subseteq \mathbf{Y} \implies \mathbf{f}(\mathbf{X}) \subseteq \mathbf{f}(\mathbf{Y}). \tag{2}$$

This property guarantees that box subdivision can only improve the precision of the paving computed by the branch and bound method. As a result of the fundamental theorem of interval arithmetic, the *natural interval extension*, obtained by replacing the real variables and operations in a function expression by their corresponding interval equivalents, is inclusion isotone.

Since the enclosure method is employed at every recursive step, it is essential for the performance of the method that each evaluation requires little computational effort. However, if the enclosures computed are too crude, too many spurious boxes will be retained, degrading the overall performance by increasing the branching factor and number of boxes that are processed. Therefore there is a trade-off between the tightness of intervals produced by an enclosure method, and the computational cost incurred.

The most straightforward way of performing arithmetic with intervals on a computer is to use the natural interval extension for each real-valued function. There are standard interval definitions for many useful mathematical operations, such as logarithmic and trigonometric functions, exponentiation, square root, etc. There are also definitions for relational operators, which are more nuanced than for real numbers. It is beyond the scope of this work to list all of these, but a comprehensive list may be found in [6]. For a further introduction to interval arithmetic, see [5]; treatments specific to branch and bound computation may be found in [7, 8].

It is well-known that the usual arithmetic laws for real numbers must be relaxed whenever an independent variable appears more than once in an interval expression. This phenomenon is known as 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.

If the set of constraints defining S is restricted to polynomial constraints, Bernstein polynomial expansion yields an enclosure method that converges faster than interval arithmetic, at the expense of extra computational effort. 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. Therefore a rigorous evaluator for p over the unit interval consists of a computation of all the Bernstein coefficients and taking their minimum and maximum. For other intervals, the polynomial (and the Bernstein basis polynomials) can be affinely transformed. The Bernstein coefficients over a generalized interval $\mathbf{x} = [\underline{x}, \overline{x}]$ are

$$b_i \;=\; \sum_{j=0}^{i} \frac{\binom{i}{j}}{\binom{d}{j}} (\overline{x} - \underline{x})^j \sum_{k=j}^{d} \binom{k}{j} \underline{x}^{k-j} a_k, \quad i = 0, \ldots, d. \tag{3}$$

For the multivariate case, i.e., the domain is a box rather than an interval, the same formula holds, but i is interpreted as a multi-index (vector index), $\sum$ denotes a nested sum, and $\binom{\cdot}{\cdot}$ denotes a generalized binomial coefficient (product of binomial coefficients) [9].

Once the Bernstein coefficients over a starting box have been computed from the power-form coefficients, coefficients over subdivided boxes can be obtained from them, using a more efficient difference table scheme. Compared to interval arithmetic, the Bernstein enclosure usually delivers tighter enclosures, and exhibits second-order convergence to the true range as interval widths tend to zero, as opposed to first-order convergence for interval arithmetic. The Bernstein enclosure is also independent of the way the polynomial is written — although it requires the power-form coefficients to be available — whereas the quality of the interval enclosure depends upon several factors, including the sparsity and local monotonicity of the polynomial, as well as the way in which it is written.

The main computational burden of the Bernstein enclosure is the requirement to exhaustively compute all of the Bernstein coefficients, the number of which exhibits exponential complexity with respect to n. As a result, the most efficient approach for branch and bound — either interval enclosure or

Bernstein enclosure — varies from example to example. Further properties of the Bernstein coefficients, and associated references, may be found in [9]. A formally-verified treatment of Bernstein polynomials is given in [10].

3 Generic Algorithm and Implementation in Kodiak

Kodiak is an implementation, written as a C++ library[1], of the formally verified generic depth-first branch and bound algorithm presented in [1]. It provides a programmatic framework for the implementation of rigorous numerical algorithms based on the brand and bound method.

The pseudo-code of the generic procedure that is implemented is presented in Algorithm 1. The algorithm 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, whose type is generic. It first computes an over-approximation, which may be crude, to the solution of the input problem by calling the function `bound`. Some instances of the branch and bound method require an early termination condition. This 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` puts together the answers computed at each recursive call and the algorithm returns that value.

As in the case of the PVS algorithm presented in [1], most datatypes and functions in Algorithm 1 are generic so that they can be implemented in different ways for different instances of the branch and bound method. However, the Kodiak library includes concrete implementations for these generic datatypes and functions so that actual branch and bound programs can be constructed. Examples of such programs are presented in Section 4.

3.1 Real Number Expressions

For each abstract function involved in a problem, a corresponding real number expression must be input. As noted earlier, foundational interval analysis

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

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
```

theory formalizes the relationship between a function, its expression, and the natural interval extension of that expression.

The Kodiak library provides a concrete representation of real number expressions. This representation 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 such as addition to 0, multiplication by 0, and multiplication by 1.

Since polynomials are ubiquitous in engineering problems, Kodiak provides datatypes and functions that operate directly on polynomials. In particular, functions are included that recognize polynomial and rational expressions and store them as a collection of monomials. Furthermore, depending upon configuration parameters, polynomials can be automatically rewritten in Bernstein form using an implicit representation of Bernstein coefficients.

In addition to symbolic real expressions, Kodiak includes datatypes for re-

lations and systems of relations. For simplicity, and without loss of generality, only equalities and inequalities with respect to 0 are supported. For all supported expressions, symbolic partial differentiation functions are provided. As explained in Section 3.3, information derived from enclosures for the partial derivatives is convenient for the definition of pruning strategies that improve the performance of the branch and bound method.

In the case of systems of relations, the Jacobian matrix of the left hand side of the system is symbolically computed. Functions for symbolic computation of the characteristic polynomial of a square matrix, its associated Hurwitz matrix, and corresponding Hurwitz determinants are also implemented in Kodiak. These functions are useful when analyzing stability of control systems by using branch and bound methods, as illustrated in Subsection 4.2.

3.2 Bounding Methods

The function `bound` in Algorithm 1 represents a generic enclosure method. By default, Kodiak uses interval arithmetic to bound a symbolic expression within a given box. For interval computations, Kodiak relies on the C++ library filib++ [6, 11], which is an efficient implementation of interval arithmetic. This library utilizes direct floating-point rounding modes to deliver rigorous approximations.

In the case of polynomials and rational functions, Kodiak can also use an enclosure method based on Bernstein expansion. In general, this enclosure method is computationally more expensive than interval arithmetic. However, for many categories of sparse multivariate polynomials appearing in practical problems, it is possible to reduce the computational requirement to compute the entire set of Bernstein coefficients, by the use of an efficient implicit representation scheme [9, 12]. This allows the determination of the Bernstein enclosure from a small subset of the Bernstein coefficients, which are computed explicitly only as needed.

In such cases, the Bernstein coefficients exhibit monotonicity properties that can be exploited to achieve a reduction in the number of coefficients that have to be actually computed. In [9, 12], three related tests are presented as sufficient conditions for the monotonicity of the Bernstein coefficients of the whole polynomial, potentially restricting the location (in the tensor) of the minimum and maximum Bernstein coefficients. These tests are based on the values and ranges of the univariate coefficients only, and are thus relatively cheap. Where monotonicity is found to hold for one or more of the variables over a box, that information is retained and inherited by any sub-boxes. All these tests are implemented in Kodiak.

This implicit representation for Bernstein coefficients also has advantages with respect to bisection. Whenever a box is bisected with respect to a single

variable, only the univariate Bernstein coefficients in that variable need to be recomputed; the others are unchanged. This is a significant improvement over the usual de Casteljau algorithm for the computation of explicit Bernstein coefficients over bisected boxes.

In the case of a rational expression, the Bernstein expansions of its numerator and its denominator can be used to compute tighter bounds for the function over a box. This range is enclosed within the interval hull of the quotients of the Bernstein coefficients of the numerator and denominator, pointwise [13]. For a rational expression, the same implicit storage mechanism can be used for the component Bernstein coefficients of its numerator and denominator, although the monotonicity tests for speedup can no longer be applied.

3.3 Pruning Strategy

The traditional approach in branch and bound is that, when branching, a box is split (partitioned) into two equally-sized sub-boxes, by bisecting one of its component intervals (variable ranges), either according to a widest-first or a round-robin strategy. It is however possible to subdivide a box into two unequally-sized sub-boxes, either with the aid of a heuristic, or coupled with a contraction operator. In many cases, round-robin can be improved upon markedly by the use of a heuristic to select the subdivision direction, i.e., the variable whose box is split at a given step of the recursion. This is usually done by computing a measure that assigns a weight to each variable and selecting the variable according to this measure. There is an overhead cost associated with computing such a measure, however this is often outweighed by a significant reduction in the branching factor and therefore the total number of sub-boxes and overall computation time. Heuristics based upon enclosures for the partial derivatives of the constraint functions are considered and tested in [14, 15].

The approach pursued in this work is to use single-variable bisection with a context-insensitive heuristic to select the variable. It is proposed that a suitable heuristic is obtained where, for each candidate variable, weight is assigned in proportion to (1) normalized box width in that variable and (2) normalized change in each involved function over the box with respect to that variable. In both cases, normalization is with respect to that measure computed for the starting box. The former imposes a kind of implicit context sensitivity, in that, if for a given box a variable has been repeatedly selected, it is increasingly penalized for its sub-boxes. The latter correctly makes the choice independent of any scalar multiplication of constraint functions. These weights can be estimated from computed enclosures for the partial derivatives of the symbolic expressions involved in the problem or from the difference in the function enclosures over opposite pairs of faces of the box, or by several other approaches. In the case of polynomial functions, tight enclosures for the

partial derivatives are obtained readily from simple differences of Bernstein coefficients [9]. The maximum of this measure over each function separately can be taken, or the measure can be summed over all functions.

As formally proved in [1], the pruning strategy does not affect the soundness of the branch and bound algorithm, therefore such measures could be computed using floating-point arithmetic instead of the more expensive interval arithmetic. Experience suggests that the resulting performance improvement, i.e., the level of sensitivity to the direction selection strategy, varies considerably from problem to problem. Alternative estimates for the rate of function change can be used in place of the partial derivatives, giving similar performance.

3.4 Soundness

Although the correctness of the generic algorithm and enclosure methods have been formally verified in PVS, no formal argument is presented here regarding the correctness of their software implementations. However, the principle strictly followed in the implementation is that the only floating-point operations permitted are those low-level operations executed by the interval arithmetic library; in other words, all real values are stored as intervals. For example, directed rounding needs to be used for the new endpoints when an interval or box is bisected. Such a scheme allows one to reliably exploit the speed of floating-point arithmetic. As with any computer-aided formal verification, the correctness argument relies upon sound hardware and libraries that are bug-free.

4 Examples

Three main instantiations of the branch and bound algorithm have been developed as part of Kodiak and are detailed in the following subsections.

4.1 Paving Systems of Nonlinear Equalities and Inequalities

A system of relations is a collection of j formulas of the form

$$f_i(\mathbf{x}) \; R_i \; 0, \tag{4}$$

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 of such a system 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. Systems of nonlinear relations occur widely in many application areas, including geometric modeling, chemical engineering, and kinematics. Robust control problems are often formulated as constraint satisfaction problems of this form.

In general, the solution set S lacks an algebraic description. Hence, it is often desired to compute reliable over- and under-approximations of S over a box $\mathbf{X}$. Approaches to compute valid pavings of S include a solver for constraint satisfaction problems with nonlinear constraints employing narrowing and pruning procedures and the interval Newton method [4], and a solver for systems of polynomial equations utilizing the Bernstein expansion [14].

Kodiak's generic branch and bound procedure has been instantiated to construct pavings of a given system of relations over a given initial box. The output of this procedure is three pavings:

- a paving for the guaranteed solution, where every point in each member box definitely satisfies all constraints,
- a (borderline) paving of candidate solution boxes, where, in each member box, it is possible that some points satisfy all constraints, and it is possible that some points violate at least one constraint,
- a paving for the complement of the solution, where every point in each member box definitely violates at least one constraint.

The first paving is a guaranteed inner approximation of S and the union of the first and second is a guaranteed outer approximation of S. These pavings can be computed to any given accuracy with respect to S.

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. The number of boxes in the paving does not increase with depth in the search tree. Where $l < n$, i.e., for underdetermined systems, there is a (possibly empty) continuum of solutions, with $n - l$ degrees of freedom. Here, the number of boxes in the solution paving increases with search depth.

Example 1. The safe domain for a certain control system is given by a small system of two polynomial inequalities in two variables [16]:

$$x_1^2x_2^4 + x_1^4x_2^2 - 3x_1^2x_2^2 - x_1x_2 + \frac{x_1^6 + x_2^6}{200} - \frac{7}{100} \leq 0 \tag{5}$$

$$-\frac{x_1^2x_2^4}{2} - x_1^4x_2^2 + 3x_1^2x_2^2 + \frac{x_1^5x_2^3}{10} - \frac{9}{10} \leq 0 \tag{6}$$

The starting box is $[-2, 2] \times [-2, 2]$ and the maximum depth is set to 20. Results of branch and bound computation using Kodiak (on a 3 GHz Intel

Table 1. Sizes of final pavings, total numbers of bisections, and computation time for varying bisection and enclosure techniques

	RR, IA	**RR, BE**	**H, IA**	**H, BE**
Strictly feasible boxes	4164	4184	3922	3952
Uncertain boxes	5760	4328	5426	4018
Strictly infeasible boxes	4444	4386	4074	4010
Total bisections	76323	28971	77899	27475
Computation time (s)	2.4	13.4	2.9	13.0

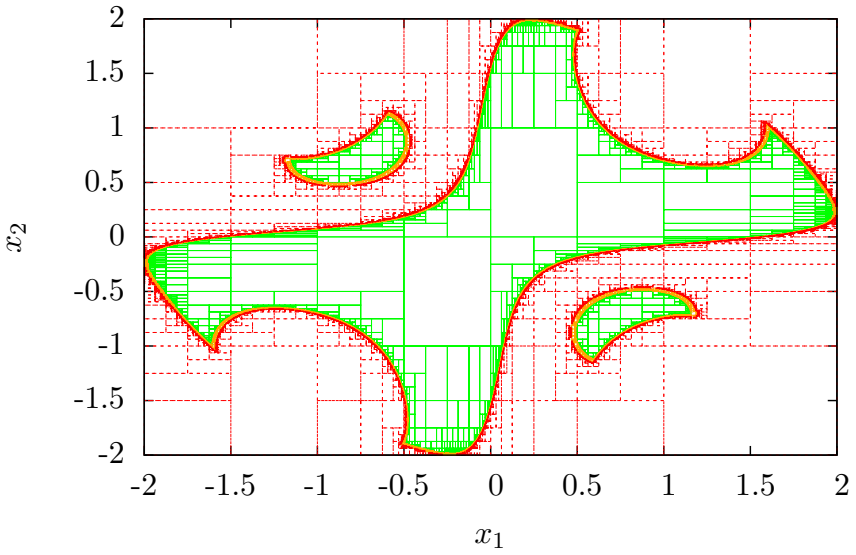


Figure 1. Three pavings for the feasible set (**H, IA**): boxes that are certainly feasible (green), possibly feasible (orange), and certainly infeasible (red)

Xeon PC, 4 Gb RAM) with round-robin (**RR**) or heuristic (**H**) bisection, and with interval arithmetic (**IA**) or Bernstein enclosure (**BE**) are summarized in Table 1. The case of heuristic bisection with interval arithmetic is depicted in Figure 1.

The computation should ideally be fast (efficient) and deliver a paving of high quality, i.e., a paving that is both tight and that contains few boxes. It can be seen that use of the heuristic for subdivision reduces the number of boxes in each category of paving, but there is a small additional computational cost. For higher-dimensional problems, the improvement often becomes much more significant and outweighs this cost. Use of Bernstein enclosure tightens the uncertain paving significantly, but at greater computational cost; the cost-benefit analysis will vary from example to example.

4.2 Equilibrium and Bifurcation Analysis of Systems of Differential Equations

As a sub-category of paving, this problem concerns a set of parameterized nonlinear ordinary differential equations (equality constraints)

$$\begin{aligned} \dot{x}_1 &= h_1(x_1, \ldots, x_n, q_1, \ldots, q_m), \\ &\vdots \\ \dot{x}_n &= h_n(x_1, \ldots, x_n, q_1, \ldots, q_m), \end{aligned} \tag{7}$$

where the dot notation indicates differentiation with respect to time, t, $x \in \mathbb{R}^n$ is the state vector, and $q \in \mathbb{R}^m$ is the parameter vector. Given a starting state at time $t = 0$, these equations define how the state vector changes with increasing t. Certain state-parameter combinations may correspond to an equilibrium, which may be either stable or unstable, where $\dot{x}_1, \ldots, \dot{x}_n$ are simultaneously zero.

A relatively straightforward task is to compute a paving for the equilibrium set. In this case, the system (7) can be treated purely as a system of equations, and a paving for its zero set can be computed, as before. The search space is a box subset of $\mathbb{R}^{m+n}$, where variables and parameters are treated equally with respect to bisection.

The local stability of an equilibrium point is determined by the vector field associated with (7) in its immediate vicinity. Variations in the parameter vector q 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 bifurcation points occurs. Such points (or loci) form a subset of the equilibrium set that is important for the stability analysis of nonlinear dynamic systems and closely related to safety for many real-world applications.

This branch and bound instantiation implements sufficient tests for the existence of two types of local bifurcation. In either case, the existing system of equations is augmented with additional automatically-derived constraints, and the same paving procedure is performed. For non-degenerate problems, the resultant paving for the bifurcation set of either category is a subset of the paving for the equilibrium set. The bifurcation set itself has one fewer degree of freedom.

The simpler type to consider is steady-state bifurcations, e.g., pitchfork and saddle-node bifurcations. They correspond to state-parameter values where the order n Jacobian matrix of $h = (h_1, \ldots, h_n)$ is singular, i.e., it has a zero eigenvalue. The Jacobian matrix is obtained from the symbolic partial derivatives of $h_1, \ldots, h_n$, and a single expression tree — which may be large — for its determinant can be computed. Wherever the enclosure for the determi-

nant over a box contains zero, then that box is a candidate for a steady-state bifurcation.

The more complex type of local bifurcations considered is Hopf bifurcations, which originate limit cycles, and 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, can be assembled into a so-called Hurwitz matrix, and boolean combinations of sign conditions on the determinants of its minors can be used as sufficient conditions for a Hopf bifurcation. Further details and references may be found in [17].

Two advantages of this approach, as compared to the usual numerical continuation method for the computation of bifurcation sets, are that a guaranteed enclosure is obtained, without missing any branches of the bifurcation set, and that the bifurcation set can be computed directly, without needing to first compute the equilibrium set.

Example 2. Kodiak has been successfully applied to the bifurcation analysis of a detailed model for the longitudinal dynamics of a jet airliner [17]. The model consists of 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 case where all five parameters are allowed to vary simultaneously, a sizable guaranteed exclusion box is computed, thereby verifying the exclusion of these categories of bifurcation in this portion of the state-parameter space for the model.

4.3 Constrained Global Optimization Problems

To a system of relations (4) an objective function $f : \mathbb{R}^n \to \mathbb{R}$ may be added, to obtain a constrained global optimization problem:

$$\min_{x \in S \cap \mathbf{X}} f(x) \tag{8}$$

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. Application areas of global optimization include the optimization of manufacturing, chemical processes, logistics, and trajectories, amongst many others.

Branch and bound techniques for global optimization problems [7, 18] rely not just on pruning boxes according to feasibility, but also according to optimality. Over each box, a lower bound for the objective function is computed. This is compared against a global variable which records the minimum upper bound for boxes processed so far which definitely contain a feasible point; where the lower bound for a box exceeds this, it can safely be discarded. Related solvers, e.g., [19, 20] commonly use a mixture of techniques, such as interval arithmetic, constraint propagation, automatic differentiation, interval Newton methods, and relaxations; some are fully rigorous, but most only deliver numerical approximations. A wide range of alternative techniques exist, including evolutionary algorithms, simulated annealing, and stochastic and algebraic approaches.

A strict depth-first search is not generally best suited to solve global optimization problems. It is inefficient to traverse a large portion of the feasible set that may be far from the actual solution. In particular, depth-first search can perform poorly in the presence of multiple local minima. Instead, it is more important to accelerate the reduction of the bound for the minimum and bypass as many feasible but non-optimal boxes as possible.

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. In the case of boxes that are feasible everywhere (all inequality constraints are satisfied and there are no non-degenerate equality constraints), monotonicity information enables a useful contraction operator. When the objective function is found to be monotone with respect to one or more variables over a box, the box can be reduced in *dimension*, or, if a previous subdivision assures the existence of a suitable neighbor, eliminated entirely. The heuristic for subdivision can be chosen with respect to the objective function, or the constraint functions, or some combination thereof. Both possibilities are enabled in Kodiak. It is not immediately clear which approach is superior as the best approach likely depends on the particular problem to be solved.

Example 3. It is desired to compute tight outer bounds for the range of a rational function of 12 variables, representing a property of a biological signaling network[2]. Both the numerator and denominator are lengthy high-degree Kirchhoff polynomials of directed graphs; the formulas and variable ranges are given in the appendix. Bounding the range of such a function has an application in reliable model rejection for certain types of steady-state biochemical models

[2]The authors would like to thank Pencho Yordanov (ETH Zurich) for providing this example by personal communication.

abstracted by directed graphs. The numerator and denominator have wide ranges over the box, but are highly correlated, such that the range of the quotient is dozens of orders of magnitude less wide than the quotient of the ranges. Without subdivision, the computed enclosures are $[5.16 \times 10^{-41}, 2.41 \times 10^{41}]$ using interval arithmetic (in less than 0.1s) and $[1.66, 156.17]$ using the rational Bernstein form (in 5m 14s). Both enclosures can be tightened further using branch and bound. With a search depth of 20, the former enclosure is improved to $[1.02 \times 10^{-34}, 1.90 \times 10^{35}]$ (after $2^{20} - 1$ bisections and 53m 8s).

5 Conclusions

This paper presents a programmatic framework for the implementation of depth-first branch and bound algorithms, together with instantiations for paving systems of nonlinear equations and inequalities, bifurcation analysis of systems of ordinary differential equations, and optimization problems. These algorithms inherit the high-level correctness properties of the generic algorithm, a closely related version of which has been formally verified. Low-level correctness in numerical computations is ensured by the use of interval arithmetic instead of floating-point arithmetic.

The main features of the proposed approach are: efficient and rigorous implicit use of floating-point arithmetic for the branching and bounding steps; interval arithmetic and Bernstein enclosures to compute guaranteed enclosures for nonlinear real-valued functions; the automatic detection of polynomials and rational functions and an efficient implicit representation for their Bernstein coefficients; automatic symbolic generation and simplification of the partial derivatives of constraint functions; use of enclosures for partial derivatives to enable the tightening of enclosures for constraint functions using monotonicity information; an efficient heuristic for the choice of subdivision direction selection; the ability to pave solution sets for underdetermined systems with one or more degree of freedom, or compute guaranteed exclusion boxes.

This approach, and the related software tool, Kodiak, have been successfully applied to problems involving up to a dozen variables and parameters with highly non-trivial constraint functions. However the curse of dimensionality cannot in general be avoided, meaning that branch and bound approaches tend to become increasingly unsuitable as the number of variables increase.

In future work, the inclusion of further enclosure methods, such as affine arithmetic and Taylor models, may potentially improve the computational performance and paving quality at high search depth. Further work includes performing branch and bound over non-box domains and extending the software library with additional solvers for validity and satisfiability of arbitrary boolean combinations of relational expressions and quantified expressions. Furthermore, a mixed search strategy, coupled with contraction and relax-

ation techniques, could yield a powerful rigorous solver for constrained global optimization problems.

References

1. Narkawicz, A.; and Muñoz, C.: A Formally Verified Generic Branching Algorithm for Global Optimization. *Fifth Working Conference on Verified Software: Theories, Tools and Experiments (VSTTE)*, E. Cohen and A. Rybalchenko, eds., vol. 8164 of *Lecture Notes in Computer Science*, 2014, pp. 326–343.

2. Owre, S.; Rushby, J.; and Shankar, N.: PVS: A Prototype Verification System. *Proceedings of the 11th International Conference on Automated Deduction — CADE-11*, D. Kapur, ed., Springer, vol. 607 of *Lecture Notes in Artificial Intelligence*, June 1992, pp. 748–752.

3. Chabert, G.; and Jaulin, L.: Contractor Programming. *Artificial Intelligence*, vol. 173, no. 11, 2009, pp. 1079–1100.

4. Granvilliers, L.; and Benhamou, F.: Algorithm 852: RealPaver: An Interval Solver Using Constraint Satisfaction Techniques. *ACM Trans. on Mathematical Software*, vol. 32, no. 1, 2006, pp. 138–156.

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

6. Lerch, M.; Tischler, G.; and Wolff von Gudenberg, J.: filib++ — Interval library specification and reference manual. 279, University of Würzburg, Germany, 2001.

7. Hansen, E. R.; and Walster, G. W.: *Global Optimization Using Interval Analysis*. Marcel Dekker, Inc., New York, Basel, second ed., 2004.

8. Ratschek, H.; and Rokne, J.: *New Computer Methods for Global Optimization*. Ellis Horwood Ltd., Chichester, 1988.

9. Smith, A. P.: Enclosure Methods for Systems of Polynomial Equations and Inequalities. Ph.D. Thesis, University of Konstanz, Germany, 2012. URL `http://nbn-resolving.de/urn:nbn:de:bsz:352-208986`.

10. Muñoz, C.; and Narkawicz, A.: Formalization of a Representation of Bernstein Polynomials and Applications to Global Optimization. *Journal of Automated Reasoning*, vol. 51, no. 2, August 2013, pp. 151–196. URL `http://dx.doi.org/10.1007/s10817-012-9256-3`.

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

12. Smith, A. P.: Fast Construction of Constant Bound Functions for Sparse Polynomials. *J. Global Optimization*, vol. 43, no. 2–3, 2009, pp. 445–458.

13. Narkawicz, A.; Garloff, J.; Smith, A. P.; and Muñoz, C. A.: Bounding the Range of a Rational Function over a Box. *Reliable Computing*, vol. 17, 2012, pp. 34–39.

14. Garloff, J.; and Smith, A. P.: 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, 2001, pp. 167–178.

15. Ratz, D.; and Csendes, T.: On the Selection of Subdivision Directions in Interval Branch-and-Bound Methods for Global Optimization. *J. Global Optimization*, vol. 7, 1995, pp. 183–207.

16. Crespo, L. G.; Muñoz, C. A.; Narkawicz, A. J.; Kenny, S. P.; and Giesy, D. P.: Uncertainty Analysis via Failure Domain Characterization: Polynomial Requirement Functions. *Proceedings of European Safety and Reliability Conference*, Troyes, France, September 2011.

17. Smith, A. P.; Crespo, L. G.; Muñoz, C. A.; and Lowenberg, M. H.: Bifurcation Analysis Using Rigorous Branch and Bound Methods. *2014 IEEE International Conference on Control Applications (CCA)*, Part of 2014 IEEE Multi-conference on Systems and Control, Antibes, France, October 2014, pp. 2095–2100.

18. Floudas, C. A.: *Deterministic Global Optimization: Theory, Methods, and Applications*, vol. 37 of *Nonconvex Optimization and its Applications*. Kluwer Acad. Publ., Dordrecht, Boston, London, 2000.

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

20. Kearfott, R. B.: GlobSol User Guide. *Optimization Methods and Software*, vol. 24, no. 4-5, August 2009, pp. 687–708.

Appendix

The function and variable ranges for Example 3 are as follows:

$$
\begin{aligned}
F &= \frac{N_1(N_2+UN_3)(N_4+UN_5)}{D_1(D_2+UD_3)(D_4+UD_5)}, \quad \text{where} \\
N_1 &= \frac{5}{3}(k_a+k_{\text{off}}), \\
N_2 &= d_2d_3d_4+d_3d_4k_d+d_2d_3k_{\text{d1}}+d_3k_dk_{\text{d1}}+\frac{2}{15}d_2k_{\text{off}}(d_4+k_{\text{d1}}+k_{\text{uoff}})+ \\
&\quad d_2d_3k_{\text{uoff}}+d_3k_dk_{\text{uoff}}+\frac{5}{3}d_3Ik_{\text{on}}(d_4+k_{\text{d1}}+k_{\text{uoff}}), \\
N_3 &= k_{\text{uon}}(d_3d_4+d_3k_{\text{d1}}+\frac{2}{15}d_4k_{\text{off}}), \\
N_4 &= d_2d_3d_4+d_3d_4k_d+d_2d_3k_{\text{d1}}+d_3k_dk_{\text{d1}}+d_2d_4k_{\text{off}}+d_4k_dk_{\text{off}}+d_2k_{\text{d1}}k_{\text{off}} \\
&\quad +k_dk_{\text{d1}}k_{\text{off}}+d_2d_3k_{\text{uoff}}+d_3k_dk_{\text{uoff}}+d_2k_{\text{off}}k_{\text{uoff}}+k_dk_{\text{off}}k_{\text{uoff}}+ \\
&\quad d_3Ik_{\text{on}}(d_4+k_{\text{d1}}+k_{\text{uoff}}), \\
N_5 &= k_{\text{uon}}(d_3d_4+d_3k_{\text{d1}}+d_4k_{\text{off}}+k_{\text{d1}}k_{\text{off}}), \\
D_1 &= k_a+\frac{2}{15}k_{\text{off}}, \\
D_2 &= d_2d_3d_4+d_3d_4k_d+d_2d_3k_{\text{d1}}+d_3k_dk_{\text{d1}}+d_2d_4k_{\text{off}}+d_2k_{\text{d1}}k_{\text{off}}+d_2d_3k_{\text{uoff}} \\
&\quad +d_3k_dk_{\text{uoff}}+d_2k_{\text{off}}k_{\text{uoff}}+d_3Ik_{\text{on}}(d_4+k_{\text{d1}}+k_{\text{uoff}}), \\
D_3 &= k_{\text{uon}}(d_3d_4+d_3k_{\text{d1}}+d_4k_{\text{off}}), \\
D_4 &= d_2d_3d_4+d_3d_4k_d+d_2d_3k_{\text{d1}}+d_3k_dk_{\text{d1}}+d_2d_3k_{\text{uoff}}+d_3k_dk_{\text{uoff}}+ \\
&\quad \frac{2}{15}k_{\text{off}}(d_2d_4+d_4k_d+d_2k_{\text{d1}}+k_dk_{\text{d1}}+d_2k_{\text{uoff}}+k_dk_{\text{uoff}})+ \\
&\quad \frac{5}{3}d_3Ik_{\text{on}}(d_4+k_{\text{d1}}+k_{\text{uoff}}), \\
D_5 &= k_{\text{uon}}(d_3d_4+d_3k_{\text{d1}}+\frac{2}{15}d_4k_{\text{off}}+\frac{2}{15}k_{\text{d1}}k_{\text{off}}),
\end{aligned}
$$

where $d_2, d_3, d_4, k_a, k_d, k_{\text{d1}}, k_{\text{on}}, k_{\text{off}}, k_{\text{uon}}, k_{\text{uoff}} \in [\frac{1}{100000}, 1], U, I \in [\frac{1}{100000}, 1000]$.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 07 - 2015	Technical Memorandum	

4. TITLE AND SUBTITLE

Kodiak: An Implementation Framework for Branch and Bound Algorithms

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Smith, Andrew P.; Munoz, Cesar A.; Narkawicz, Anthony J.; Markevicius, Mantas

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

154692.02.50.07.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

L-20559

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA-TM-2015-218776

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 64
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Recursive branch and bound algorithms are often used to refine and isolate solutions to several classes of global optimization problems. A rigorous computation framework for the solution of systems of equations and inequalities involving nonlinear real arithmetic over hyper-rectangular variable and parameter domains is presented. It is derived from a generic branch and bound algorithm that has been formally verified, and utilizes self-validating enclosure methods, namely interval arithmetic and, for polynomials and rational functions, Bernstein expansion. Since bounds computed by these enclosure methods are sound, this approach may be used reliably in software verification tools. Advantage is taken of the partial derivatives of the constraint functions involved in the system,firstly to reduce the branching factor by the use of bisection heuristics and secondly to permit the computation of bifurcation sets for systems of ordinary differential equations. The associated software development, Kodiak, is presented, along with examples of three different branch and bound problem types it implements.

15. SUBJECT TERMS

Bernstein expansion; Bound; Branch; Formal verification; Interval arithmetic; Theorem proving

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	24	(757) 864-9658