Interval Methods Revisited

P. Van Hentenryck
Brown University
Box 1910
Providence, RI 02912
pvh@cs.brown.edu

D. McAllester
MIT AI Lab
Technology Square, 545
Cambridge, USA
dam@ai.mit.edu

D. Kapur
SUNY at Albany
Dep. of Computer Science
Albany, NY-12222
kapur@cs.albany.edu

Abstract

This paper presents a branch & cut algorithm to find all isolated solutions of a system of polynomial constraints. Our findings show that fairly straightforward refinements of interval methods inspired by AI constraint propagation techniques result in a multivariate root finding algorithm that is competitive with continuation methods on most benchmarks and which can solve a variety of systems that are totally infeasible for continuation methods. For example, we can solve the Broyden Banded function benchmark for hundreds of variables and hundreds of cubic equations.

1 Introduction

In this paper we consider the classical problem of finding solutions to systems of nonlinear polynomial equations in many variables. This is an old problem with many applications and a large literature. In engineering applications it is generally sufficient to find an assignment of floating point numbers to variables such that the given constraints are satisfied to within the uncertainty introduced by the quantization of floating point numbers and numerical error. The problem of determining the existence of such floating point numbers is NP complete.

Floating point solutions to a system of nonlinear equations can be found using variants of branch and bound search techniques that have been developed for other NP complete problems. This paper describes a particular "branch and cut" search method based on interval techniques. Interval techniques manipulate upper and lower bounds on variables and have been explored by a variety of authors [12, 5, 4, 8]. The main innovation presented here consist of two "local interval cuts" that are effective in regions of large uncertainty, i.e., where the bounds associated with variables remain quite loose. Our two novel cuts are local in the sense that each cut examines only a single constraint. Our emphasis on local constraints is inspired by constraint propagation techniques developed for AI and logic programming communities [18, 3]. Our local interval cuts seem empirically to reduce the time required to solve certain benchmarks, such as the Broyden Banded function, from exponential to linear in the number of variables.

Continuation methods provide another approach to solving systems of nonlinear equations, e.g., [17, 23]. Continuation methods are quite different from the traditional branch and bound or branch and cut approaches to NP complete problems. Unlike standard branch and cut approaches, continuation methods have exponential *best case* performance — the best case run time grows exponentially in the number of nonlinear constraints. Of course both approaches have exponential worst case behavior and the existence of good best case behavior for branch and cut does not imply that it will run faster on practical problems. We report here, however, that with our improved pruning techniques our interval based branch and cut system outperformed continuation methods on most of the benchmark problems we found in the continuation literature.

The heart of our technique is an algorithm for taking a system of nonlinear constraints, and a given set of bounds on the values of variables, and iteratively improving the known bounds using a set of inference rules called interval cuts. An interval cut is a method of inferring an inequality of the form $x \leq b$ or $x \geq b$ where b is a floating point number.

2 Three Cuts

In order to define our cuts some preliminary terminology is needed. Our algorithms work with floating point numbers which we assume to be a finite set of rational numbers extended with both a positive and negative infinity, denoted $+\infty$ and $-\infty$ respectively. We will generally use the term "interval" to refer a closed interval of the form $[a,\ b]$ where a and b are floating point numbers. Our algorithms also work with symbolic expressions constructed from floating point constants, symbolic variables, addition, subtraction, multiplication, and "power expressions" of the form e^n where e is an expression and n is a positive integer constant. The value of power expressions, in addition to multiplication, involves the accuracy of interval evaluation and is discussed below. We will generally use the term "expression" to refer to symbolic expressions of this form. We will use the term "box" to refer to an assignment of intervals to symbolic variables. For box B and variable x we write $B(x)$ to denote the interval assigned to x in the box B. A "point" is an assignment of particular floating point numbers to each variable. Clearly, if we are working with n variables then a box is a rectilinear subset of R^n. We allow $B(x)$ to be interval of the form $[a,\ a]$ and hence a point is really just a box in which all intervals are of this form.

All interval methods are based on *interval evaluation* — an evaluation process which computes an interval value for an expression from given intervals for the variables it contains. The most straightforward interval evaluation method is obtained by associating each of the operations of addition, subtraction, multiplication, and powers with a corresponding operation from intervals to intervals. More specifically we have the following operations on intervals.

$$[a,\ b] + [c,\ d] = [a + c,\ b + d]$$

$$[a,\ b] - [c,\ d] = [a - d,\ b - c]$$

$$[a,\ b] * [c,\ d] = [min(ac,\ ad,\ bc,\ bd),\ max(ac,\ ad,\ bc,\ bd)]$$

$$[a,\ b]^n = \begin{cases} [a^n,\ b^n] & \text{for } n \text{ odd} \\ [0, max(a^n,\ b^n)] & \text{for } n \text{ even and } 0 \in [a, b] \\ [min(a^n, b^n),\ max(a^n, b^n)] & \text{for } n \text{ even and } 0 \notin [a,\ b] \end{cases}$$

We use the notation $NE(e,\ B)$ to denote the interval value of an expression e when variables are assigned the interval values given in B and operations are interpreted as operations on intervals according to the above rules (NE stands for "natural evaluation"). For example if $B(x)$ is $[-1,\ 1]$ then $NE(x + x,\ B)$ is the interval $[-2,\ 2]$; $NE(x * x,\ B)$ is the interval $[-1,\ 1]$; but $NE(x^2,\ B)$ is $[0,\ 1]$. For most nontrivial expressions the interval $NE(e,\ B)$ is larger than the actual range of values of e over points in B. If $B(x)$ is $[-1,\ 1]$ we have that $NE(x^2 - x,\ B)$ is $[-1,\ 2]$ while the actual range of values in this box is $[-1/4,\ 2]$. In our implementation outward rounding is used in the above rules — in arithmetic for computing lower bounds we round down and in arithmetic for computing upper bounds we round up. This provides a guarantee that $NE(e,\ B)$ contains the actual range of values of e independent of numerical errors. This guarantee is often expressed as the statement that interval evaluation is "conservative".

We will assume a given set of constraints of the form $e \leq 0$ where e is an expression as defined above. An equation $e_1 = e_2$ can obviously be represented with two constraints of the form $e_1 - e_2 \leq 0$ and $e_2 - e_1 \leq 0$. For technical reasons it will be easier to work with inequalities.

An interval cut is a procedure which operates on a set of constraints and a current box. An interval cut reduces the given box by deriving a new bound on one of the variables. Cuts are presented below as nondeterministic procedures — performing a cut often involves guessing one or more values. In practice these guesses are generated by deterministic heuristic methods such as those described in section 3. A nondeterministic cut procedure can also be viewed as an inference rule for deriving new bounds independent of heuristic methods for finding the appropriate inferences to perform.

2.1 Newton Cuts

A Newton cut is represented schematically in figure 1. The notation used in the figure is defined rigorously in the following description. In the Newton cut, and in later cuts, we abuse notation and use de/dx to denote the expression that is the symbolic derivative of e with respect to x.

> *Newton Cut:* Let $e \leq 0$ be a constraint, B a given box, x a variable appearing in e, and let $[a,\ b]$ be the interval $B(x)$. Let c and c' be two cut values such that $c < b$ and c' in $[c,\ b]$. Let s be the lower bound of $NE(e,\ B[x := [c', c']])$. We require $s > 0$. Let $[D_l,\ D_u]$ be the interval $NE(de/dx,\ B[x := [c,\ b]])$. To ensure there is no solution with $x \in [c', b]$ we require that either $D_l \geq 0$ or $D_l < 0$ and $c' - \frac{s}{D_l} > b$. If $D_u \leq 0$ we can infer $x \leq c$ and if $D_u > 0$ we can infer $x \leq max(c,\ c' - \frac{s}{D_u})$.

The lines of slope D_u and D_l in the figure 1 represent the fastest possible rate of descent of the value of the expression e as a function of x based on the interval evaluation of the derivative de/dx. It should be clear that all values of x inside the triangle defined by these lines must violate the constraint $e \leq 0$ and can be pruned.

An interesting special case is when $c = a$ and $c' = b$. Whenever the lower bound of $NE(e,\ B[x := b, b]])$ is greater than zero the upper bound b can be ruled out as a value of x. If we are working with a finite box, i.e., one where all variable intervals are finite, and ignoring roundoff errors, D_u must be finite and the choice of $c = a$ and $c' = b$ always achieves some reduction of the upper bound. However, larger cuts are usually achieved by other choices of c and c'. Increasing the value of c reduces the size of the interval $[c, b]$ and hence reduces the range of possible derivative values $NE(de/dx,\ B[x := [c, b]])$. This often gives a smaller value for D_u and hence a larger cut.

Decreasing the value of c' away from b can also give a larger cut. As in the statement of the cut, let $s(c')$ be the lower bound of $NE(e,\ B[x := [c',\ c']])$, as a function of c'. In practice a reduction in the upper bound can only be achieved when $s(b) > 0$, i.e., the value b can be ruled out. As c' is reduced from b, the lower bound $s(c')$ will also typically be reduced. However, it typically falls more slowly than D_u, which is the largest possible value of de/dx in the interval $[c, b]$. If ds/dc' is smaller than D_u then reducing c' away from b will result in a larger cut. However if we reduce c'

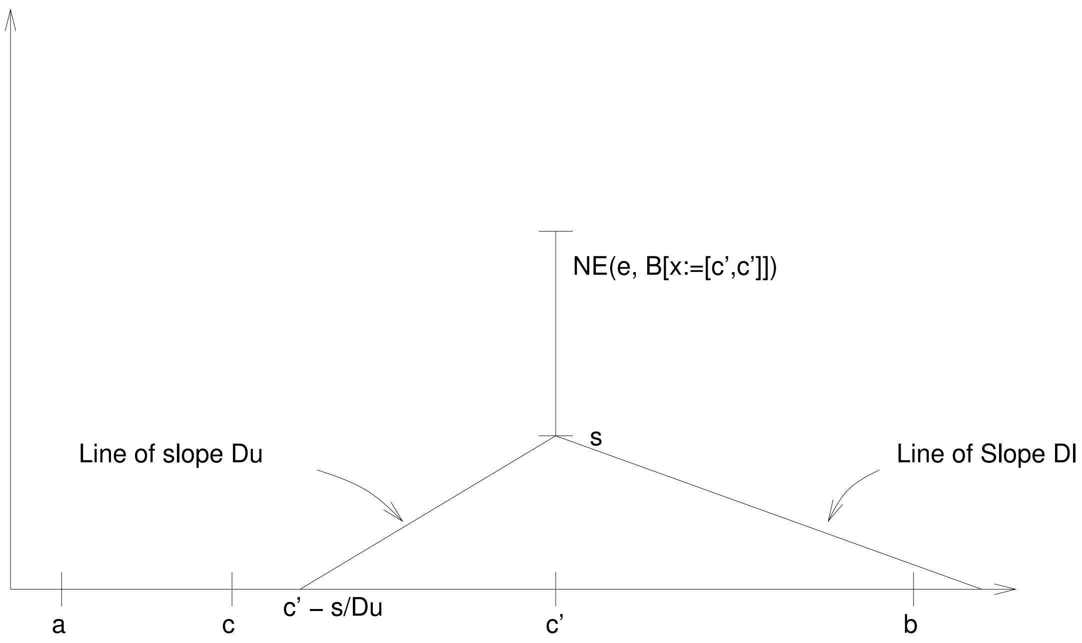

Figure 1: A Newton cut deriving $x \leq c' - \frac{s}{D_u}$

too aggressively then the cut may fail altogether either because we fail to get $s(c') > 0$ or we fail to rule out feasible points with $x \in [c',\ b]$.

The above cut allows for the reduction of upper bounds. We call it an upper bound cut. Every upper bound cut has a dual lower bound cut. We will only present upper bound cuts.

2.2 Snake Cuts

Consider a given constraint $e \leq 0$, a given variable x appearing in the constraint, a given box B, and let $[a,\ b]$ be the interval $B(x)$. Now consider a value c' in the interval $[a,\ b]$ and consider the lower bound s of the interval $NE(e,\ B[x := [c', c']])$ as a function of c'. We will write $s(c')$ to express s as a function of c' for fixed values of e and B. If $s(c') > 0$ then no point in B with $x = c'$

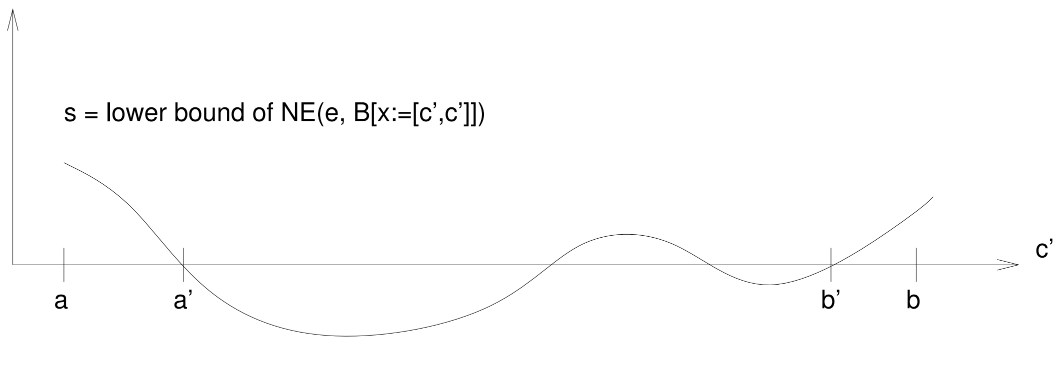

Figure 2: The snake $s =$ the lower bound of $NE(e,\ B[x := [c', c']])$

can satisfy $e \leq 0$ so c' can be ruled out as a value of x. This situation is drawn in figure 2. If $s(a) > 0$ or $s(b) > 0$ then Newton cuts can be used to reduce the interval. In figure 2 the interval can be reduced with Newton cuts to $[a', b']$. In informal discussions we began calling the function $s(c')$ a "snake" and the cut based on an analysis of this function became known as the snake cut. Snake cuts provide a way of finding the roots a' and b' in a way that often requires fewer cuts than are required with Newton cuts.

The snake cut evaluated the derivative ds/dc' rather than de/dx. It turns out that the range of values of ds/dc' over the interval $[c, b]$ is usually significantly less than the range of possible values of values of de/dx. This fact allows each individual cut to be larger.

The first step in the snake cut is to represent the function $s(c')$ as a symbolic expression $s(x)$ involving only the variable x. To do this we first symbolically rewrite e as a polynomial $P(x)$ of the form $s_0 + s_1 x + s_2 x^2 + \ldots + s_n x^n$ where each coefficient s_i is an expression not involving x. We then define two polynomials in x, $P^-(x)$ and $P^+(x)$, each of which has constant coefficients (and hence no variables other than x). $P^-(x)$ gives a lower bound on e for negative values of x and $P^+(x)$ gives a lower bound on e for positive values of x. For odd powers of x the coefficients of $P^-(x)$ are taken to be the upper bound of $NE(s_i, B)$ where s_i is the corresponding symbolic coefficient of the polynomial $P(x)$. In all other cases the coefficients of $P^-(x)$ and $P^+(x)$ are taken to be the lower bound of $NE(s_i, B)$. A function $s(x)$ can now be represented symbolically by the conditional expression $\mathtt{if}(x > 0,\ P^+(x),\ P^-(x))$ which has the property that it equals $P^+(x)$ for positive values of x and $P^-(x)$ for other values of x. The interval evaluation function NE and the symbolic differentiation operators can be easily extended to handle conditional expressions. The symbolic expression $s(x)$ does not correspond exactly to the function $s(c')$ as defined above because rewriting an expression as a polynomial in x changes the interval returned by interval evaluation. However, the symbolic expression $s(x)$ does give a valid lower bound on the values of e for points in the box B as a function of the value for x. The snake cut is identical to the Newton cut except that we use ds/dx rather than de/dx.

> *Snake Cut:* Let $e \leq 0$, x, B, $[a, b]$ and the symbolic expression $s(x)$ be defined as above. Let c and c' be two additional parameters as in the Newton cut with $c < b$ and c' in $[c, b]$. We require that $s(c') > 0$. Let $[D_l, D_u]$ be the interval $NE(ds/dx, B[x := [c, b]])$. To ensure there is no solution with $x \in [c', b]$ we require that either $D_l \geq 0$ or $D_l < 0$ and $c' - \frac{s}{D_l} > b$. If $D_u \leq 0$ we can infer $x \leq c$ and if $D_u > 0$ we can infer $x \leq max(c,\ c' - \frac{s}{D_u})$.

To see the advantage of the snake cut over the Newton cut consider the expression $zx - 2 \leq 0$ where the box B assigns both x and z the interval $[1, 4]$. It is not difficult to see that since z is in $[1, 4]$ and $zx \leq 2$ we have $x \leq 2$. We compare a Newton cut with a snake cut where we take $c = 1$ and $c' = 4$ in both cuts (i.e., $c = a$ and $c' = b$). In the Newton cut we have that $NE(de/dx, B)$ equals $NE(z, B)$ which is $[1, 4]$. So the infered bound is $x \leq 4 - \frac{2}{4} = 3\frac{1}{2}$. However the lower bound polynomial $s(x)$ (for positive values of x) is $x - 2$. So $NE(ds/dx, B)$ equals $NE(1, B)$ equals $[1, 1]$ and the infered bound is $x \leq 4 - \frac{2}{1} = 2$, an optimal cut.

Unfortunately snake cuts require rewriting the constraint in a way that makes interval evaluation less accurate. So although snake cuts find the roots a' and b' quickly, these roots result in a wider interval than that resulting from Newton cuts on a version of the constraint allowing more accurate interval evaluation. Hence we use Snake cuts at first to achieve large initial cuts and then convert to Newton cuts to get tighter bounds with less being cut at each step.

2.3 Combination Cuts

The combination cut is used to gain some of the power of multivariate Newton's method once the search has begun to converge on a solution. Interval versions of multivariate Newton's method are widely used in interval systems, e.g., [5, 4]. The Newton cuts we use involve two ideas. First, we compute combined constraints by inverting the "center Jacobian" matrix of the constraint system. Second, in the combination cuts we use a different method of interval evaluation called Taylor evaluation which is more accurate over small boxes.

Like the natural evaluation operator NE, Taylor interval evaluation takes an expression e and a box B and returns an interval containing the range of values of e on points in B. Taylor interval evaluation tends to be more accurate in the case where B is small. To define the Taylor interval evaluation we first define B_c to be the center of the box B, i.e., $B_c(x)$ is the interval $[c, c]$ where c is the center of the interval $B(x)$. We use the notation $x_i \in e$ to indicate that x_i is a variable appearing in the expression e.

$$TE(e,\ B) = NE(e,\ B_c) + \sum_{x_i \in e} NE(de/dx_i, B) * (B(x_i) - B_c(x_i))$$

All arithmetic operations in the above expression are operations on intervals. Taylor interval evaluation is more accurate than natural interval evaluation in small intervals. For example, consider the polynomial $x^2 - x$ where $B(x)$ is the interval $[x_c - \epsilon,\ x_c + \epsilon]$ where $x_c > \frac{1}{2}$ and ϵ is a small positive number. We have

$$NE(x_c^2 - x_c,\ B) = [(x_c^2 - x_c) - (2x_c + 1)\epsilon + \epsilon^2,\ (x_c^2 - x_c) + (2x_c + 1)\epsilon + \epsilon^2]$$
$$TE(x_c^2 - x_c,\ B) = [(x_c^2 - x_c) - (2x_c - 1)\epsilon + 2\epsilon^2,\ (x_c^2 - x_c) + (2x_c - 1)\epsilon + 2\epsilon^2]$$

For small values of ϵ the width of the first interval is approximately $2(2x_c + 1)\epsilon$ while the width of the second interval is approximately $2(2x_c - 1)\epsilon$. Since we have assumed $x_c > \frac{1}{2}$ the first interval is wider than the second. (If we assume $x_c < \frac{1}{2}$ then the interval product in the Taylor interval calculation gives a different symbolic answer and the Taylor interval is still smaller.) The natural evaluation does not take into account the correlations between the two terms in the polynomial $x^2 - x$. More generally, if $P(x)$ is a polynomial in x and $B(x)$ is an interval of the form $[x_c - \epsilon,\ x_c + \epsilon]$ then to first order in ϵ we have that the width of $TE(P(x),\ \epsilon)$ is $2P'(x_c)\epsilon$ where P' is the symbolic derivative of P. $P'(x_c)$ is nonzero then as ϵ goes to 0 the ratio of the width of $TE(P(x),\ B)$ to the width of the true range of values of $P(x)$ approaches 1 — Taylor interval evaluation becomes exactly accurate in the limit of small intervals. As the above example shows, this is not true of the natural evaluation.

Combination cuts build new constraints which are linear combinations of given constraints. If we are given $e_1 \leq 0$ and $e_2 \leq 0$ then for any positive α and β we can infer $\alpha e_1 + \beta e_2 \leq 0$. A combination cut is identical to a Newton cut except that it uses a constraint derived as a linear combination of input constraints and uses TE rather than NE in the computation of s.

> *Combination Cut:* Let x be a variable, B a box, and $[a, b]$ the interval $B(x)$. Let c and c' be two additional parameters with $c < b$ and c' in $[c, b]$. Let $e_1 \leq 0,\ e_2 \leq 0,\ \ldots,\ e_n \leq 0$ be a set of constraints such that each e_i contains x. Let u be a linear combination $\alpha_1 e_1 + \alpha_2 e_2 + \ldots + \alpha_n e_n$ where each α_i is positive. Let s be the lower bound of $TE(u,\ B[x := [c',\ c']])$. We require $s > 0$. Let $[D_l,\ D_u]$ be the interval $NE(du/dx,\ B)$.

To ensure there is no solution with $x \in [c', b]$ we require that either $D_l \geq 0$ or $D_l < 0$ and $c' - \frac{s}{D_l} > b$. If $D_u \leq 0$ we can infer $x \leq c$ and if $D_u > 0$ we can infer $x \leq max(c,\ c' - \frac{s}{D_u})$.

To see the power of the combination cut consider the two constraints $x+y-2 \leq 0$ and $x-y \leq 0$. If both $B(x)$ and $B(y)$ are the interval $[0,\ 2]$ then no Newton or snake cut can be used to improve the bounds — each constraint is "satisfied at the end points". For example $NE(x+y-2,\ B[x := [2,2]])$ is $[0,2]$ so $x = 2$ can not be ruled out on the basis of this constraint alone. To reduce the box B we must examine both constraints simultaneously. Adding the two constraints together (with unit coefficients) gives $2x - 2 \leq 0$. This immediately gives the tighter bound $x \leq 1$. If we are given the four inequality constraints representing $x + y = 2$ and $x - y = 0$ then four combination cuts can be used to solve for x and y exactly. A method for finding appropriate combination cuts is described in the next section.

3 The Branch and Cut Procedure

Our system finds feasible points using a branch and cut technique. We start with a box large enough to contain all the points of interest. We then prune this box using a procedure described below for applying cuts. After pruning there are three cases. First, the pruning operation may have resulted in an empty box in which case we return failure. Second, it may be the case that the interval associated with each variable has been reduced to a width below a specified accuracy parameter. In this case we terminate and return the box. If the pruning operation results in a box B which is not of sufficient accuracy, then we select a variable x and let $[a,\ b]$ be the interval $B(x)$. We then "split" the box and generate the two "branches" $B[x := [a,\ \frac{a+b}{2}]]$ and $B[x := [\frac{a+b}{2}, b]]$. We then recursively look for solutions in each branch. The first step of each branch is to prune the half-box being considered in that branch. Our implementation uses a simple round-robin strategy to select the variable x to split — down any branch of the search tree we split a variable that has gone the longest time without splitting.

Space does not allow a precise specification of the pruning procedure used to generate the data given in the next section. It seems likely that the procedure used to generate the data in section 4 is overly complex and that a wide variety of strategies for applying the cuts described in the previous section give essentially similar performance. The basic idea is to approximate a greedy algorithm which iteratively heuristically selects large cuts while not wasting too much time trying to find optimal cuts. We give a precise description of a fairly simple procedure which we believe captures the spirit of our implementation and which we believe has essentially the same performance characteristics. The reader should keep in mind, however, that this is not a completely faithful description of the algorithm used to generate table 1.

We maintain a priority queue of "cut specifications". Each cut specification is a data structure of the form $Newton_b(e,\ x,\ n)$, $Snake_b(e,\ x,\ n)$, or $Comb_b(x)$ where b is either u (for an upper bound cut) or l (for a lower bound cut). For example, the data structure $Newton_u(e,\ x,\ n)$ specifies that we are to attempt a Newton cut to tighten the upper bound on variable x using the constraint $e \leq 0$. The parameter n specifies how to select c and c' as described below. $Newton_l(e,\ x,\ n)$ specifies the analogous cut to tighten the lower bound of x.

The parameter n in a Newton or snake cut specification is called the "generation" parameter. For an upper bound cut specification of generation n we always take c to be $b - \frac{b-a}{2^n}$ and c' to be the midpoint of $[c, b]$. So for upper bound cuts of generation 0 we have $c = a$ and $c' = \frac{a+b}{2}$. For upper bound cut specifications of generation 1 we have $c = \frac{a+b}{2}$ and $c' = b - \frac{b-a}{4}$. Higher generations upper bound cut specifications take c and c' to be closer to b. Lower bound cut specifications start with $c = b$ and move c and c' closer to a in higher generation cuts. An upper bound cut, if successful, always eliminates the interval $[c',\ b]$. Hence cuts specifications of generation 0 are "aggressive" in that they attempt to cut at least half of the interval. Higher generation cuts are less aggressive but more likely to succeed.

To prune a given box the queue is initialized to contain all snake cut specifications of the form $Snake_u(e,\ x,\ 0)$ and $Snake_l(e,\ x,\ 0)$ for each constraint $e \leq 0$ and variable x appearing in e. The prune procedure iteratively removes a cut specification from the priority queue, attempts to perform the corresponding cut, and then places some number (possibly zero) of new cut specifications on the queue. Snake cut specifications always take priority over Newton cut specifications which take priority over combination cuts. Lower generation snake cuts take priority over higher generation snake cuts and similarly for Newton cuts. When a snake or Newton cut specification is removed from the queue and attempted, the cut can either succeed or fail. (To succeed in an upper bound cut we require that $s > 0$ and either $D_l > 0$ or $D_l < 0$ and $c' - \frac{s}{D_l} > b$ — see figure 1.) If any snake or Newton cut succeeds then we place on the queue all cut specifications of the form $Snake_b(e',\ y,\ 0)$ where e' is any constraint involving the newly restricted variable x and y is a variable in e'. If a cut specification $Snake_b(e,\ x,\ n)$ fails then there are two sub-cases. If n is less than 3 then we place the specification $Snake_b(e,\ x,\ n+1)$ on the queue. If n equals 3 then we place the specification $Newton_b(e,\ x,\ 0)$ on the queue. If a cut specification $Newton_b(e,\ x,\ n)$ fails then there are two analogous sub-cases. If n is less than 3 then we place the specification $Newton_b(e,\ x,\ n+1)$ on the queue. If n equals 3 then we place the specification $Comb(x)$ on the queue. By not allowing snake and Newton cuts of generation larger than 3 we ensure that each successful snake or Newton cut removes at least 1/16 of the interval of some variable. This implies that the interval of a given variable shrinks geometrically in the number of successful snake and Newton cuts that have been applied to that variable.

The processing of snake and Newton cuts will either terminate in a sufficiently precise box, as it does in the Broyden Banded function benchmark, or it terminates in a state where the box is not yet sufficiently precise but the queue contains only cut specifications of the form $Comb_b(x)$. Our current implementation works with systems of constraints of the form $e_i = 0$. This allows us to compute combined constraints of the form $u_x = 0$ where each u_x is a linear combination of the e_i and where u_x is intended to constrain x independent of the value of other variables. To compute the combined constraints u_x let B be the box existing at the time we are computing the combined constraints. We define the "center Jacobian" matrix $[B_{i,j}]$ by

$$B_{i,j} = \text{the midpoint of the interval } \ NE(de_i/dx_j,\ B).$$

If the matrix $[B_{i,j}]$ is singular then our procedure terminates and no combination cuts are tried. Otherwise we let $[A_{i,j}]$ be the inverse of $[B_{i,j}]$. We then define the expression u_{x_i} by

$$u_{x_i} = \sum_{k=1}^{n} A_{i,k} e_k.$$

Inside the box B, and especially for reasonably small boxes, we have that du_{x_i}/dx_j is near 1 if $i = j$ and near 0 otherwise. So $u_x = 0$ is primarilly a constraint on x. Once the combined constraints of the form $u_x = 0$ have been computed the system restricts all cut attempts to combined cuts on these combined constraints. The matrix $A_{i,j}$ which defines the combined constraints is only computed once — it is not recomputed as the box continues to shrink.

When a cut specification of the form $comb_u(x)$ is removed from the queue the system attempts combined cuts on the constraints $u_x \leq 0$ and $-u_x \leq 0$. In attempting an upper bound combined cut the system always selects $c = a$ and $c' = \frac{a+b}{2}$. In sufficiently small intervals, and using combined constraints specific to individual variables, these "zeroth generational" cuts succeed. If the box is large then the cut attempt may fail in which case nothing is placed back on the queue. If any new bound for x is generated, we place on the queue all cut specifications of the form $Comb(y)$ such that x appears in the combined constraint $u_y = 0$ for y. This process terminates when either the box has reached adequate precision or no combined cut on any variable can reduce the box further.

The above queue processing rules yield a two phase process. First the box is reduced using snake and Newton cuts of generation no larger than 3. Then the system computes the inverse of the center Jacobian to construct combined constraints. It then performs cuts of generation zero on these combined constraints until no more such cuts are possible. At each node in the search tree this two phase pruning operation is repeated until the box is no longer reduced. Since all successful cuts in this process prune at least 1/16 of an interval, the pruning process must terminate in time linear in the number of variables and the number of bits of accuracy required.

4 Experimental Results

Table 1 summarizes our experimental results. The table gives run times for three systems on a variety of benchmark problems. As mentioned in the previous section, the implementation of our techniques used to generate the table 1 is somewhat more complex than the procedure described above, although we expect the procedure described above to have essentially the same behavior. The benchmarks were taken from papers on numerical analysis [15], interval analysis [5, 7, 14], and continuation methods [23, 17, 16, 11]. Our system was run on a `Sun Sparc 10` workstation to obtain all solutions. In our runs the final intervals must have widths smaller than 10^{-8}. The timings for our system are listed in the column labeled Newton. The column labeled `HRB` gives timings for a traditional interval method using Hansen-Segupta's operator, range testing, and branching. This method uses the same implementation technology as ours, C code running on a SPARC 10. Some interval methods such as [4] have better convergence properties near solutions. Our main contribution aims at speeding up the computation when far from a solution and hence comparing it to `HRB` is meaningful. Finally, the column labeled `CONT` gives timings for a state-of-the-art continuation method [23] running on a `DEC 5000/200` which is perhaps slightly slower than a SPARC 10. For each benchmark, we give the number of variables (n), the total degree of the system (d), the initial range for the variables, and the results of each method in seconds. A space in a column means that the result is not available for the method. A question mark means that the method does not terminate in a reasonable time ($>$ 1 hour).

5 Conclusion

In this paper, we presented a branch & cut algorithm to find all isolated solutions to a system of polynomial constraints over the reals. Our techniques were originally inspired by local constraint propagation techniques used in AI and logic programming. The mathematics behind our techniques

Benchmarks	v	d	range	`Newton`	`HRB`	`CONT`
Broyden	10	3^{10}	[-1,1]	1.65	18.23	
Broyden	20	3^{20}	[-1,1]	4.25	?	
Broyden	320	3^{320}	[-1,1]	113.71	?	
Broyden	320	3^{320}	$[-10^8, 10^8]$	143.40	?	
Moré-Cosnard	20	3^{20}	$[-4, 5]$	24.49	968.25	
Moré-Cosnard	40	3^{40}	$[-4, 5]$	192.81	?	
Moré-Cosnard	80	3^{80}	$[-4, 5]$	1752.64	?	
Moré-Cosnard	80	3^{80}	$[-10^8, 0]$	1735.09	?	
i1	10	3^{10}	[-2,2]	0.06	14.28	
i2	20	3^{20}	[-1,2]	0.30	1821.23	
i3	20	3^{20}	[-2,2]	0.31	5640.80	
i4	10	6^{10}	[-1,1]	73.94	445.28	
i5	10	11^{10}	[-1,1]	0.08	33.58	
kin1	12	4608	$[-10^8, 10^8]$	14.24	1630.08	
kin2	8	256	$[-10^8, 10^8]$	353.06	4730.34	35.61
eco	4	18	$[-10^8, 10^8]$	0.60		1.13
eco	5	54	$[-10^8, 10^8]$	3.35		5.87
eco	6	162	$[-10^8, 10^8]$	22.53		50.18
eco	7	486	$[-10^8, 10^8]$	127.65		991.45
eco	8	1458	$[-10^8, 10^8]$	915.24		
eco	9	4374	$[-10^8, 10^8]$	8600.28		
combustion	10	96	$[-10^8, 10^8]$	9.94		57.40
chemistry	5	108	$[0, 10^8]$	6.32		56.55
neuro	6	1024	$[-10, 10]$	0.91		5.02
neuro	6	1024	$[-1000, 1000]$	172.71		5.02

Table 1: Summary of the Experimental Results

are quite straightforward and far less sophisticated than that underlying continuation methods. And yet our techniques seem to compare well with continuation methods on a wide variety of benchmarks. It seems likely that the techniques presented here can be improved by generalizing the cuts to allow arbitrary linear cuts in the sense of linear programming and by introducing linear programming techniques in to the pruning operation. It seems possible that branch and cut techniques provide the most effective approach to nonlinear root finding just as they provide the most effective approach to a wide variety of other NP complete problems.

Acknowledgments

We would like to thank F. Benhamou, A. Colmerauer, and B. Le Charlier for many interesting discussions on this topic. This research was partly supported by the Office of Naval Research under grant N00014-91-J-4052 ARPA order 8225, the National Science Foundation under grant numbers CCR-9357704, a NSF National Young Investigator Award.

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New

York, NY, 1983.

[2] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proceedings of the International Symposium on Logic Programming (ILPS-94)*, pages 124–138, Ithaca, NY, November 1994.

[3] F. Benhamou and W. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 1995. To Appear.

[4] E.R. Hansen and R.I. Greenberg. An Interval Newton Method. *Appl. Math. Comput.*, 12:89–98, 1983.

[5] E.R. Hansen and S. Sengupta. Bounding Solutions of Systems of Equations Using Interval Analysis. *BIT*, 21:203–211, 1981.

[6] E.R. Hansen and R.R. Smith. Interval Arithmetic in Matrix Computation: Part II. *SIAM Journal on Numerical Analysis*, 4:1–9, 1967.

[7] H. Hong and V. Stahl. Safe Starting Regions by Fixed Points and Tightening. To Appear in *Computing*, 1995.

[8] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4:187–201, 1969.

[9] R. Krawczyk. A Class of interval Newton Operators . *Computing*, 37:179–183, 1986.

[10] A.K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[11] K. Meintjes and A.P. Morgan. Chemical Equilibrium Systems as Numerical test Problems. *ACM Transactions on Mathematical Software*, 16:143–151, 1990.

[12] R.E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[13] R.E. Moore. *Methods and Applications of Interval Analysis.* SIAM Publ., 1979.

[14] R.E. Moore and S.T. Jones. Safe Starting Regions for Iterative Methods. *SIAM Journal on Numerical Analysis*, 14:1051–1065, 1977.

[15] J.J. More and M.Y. Cosnard. Numerical Solution of Nonlinear Equations. *ACM Transactions on Mathematical Software*, 5:64–85, 1979.

[16] A.P. Morgan. Computing All Solutions To Polynomial Systems Using Homotopy Continuation. *Appl. Math. Comput.*, 24:115–138, 1987.

[17] A.P. Morgan. *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[18] W. Older and A. Vellino. Extending Prolog with Constraint Arithmetics on Real Intervals. In *Canadian Conference on Computer & Electrical Engineering*, Ottawa, 1990.

[19] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* Ellis Horwood Limited, Chichester, 1988.

[20] P. Van Hentenryck. A Logic Language for Combinatorial Optimization. *Annals of Operations Research*, 21:247–274, 1989.

[21] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series, The MIT Press, Cambridge, MA, 1989.

[22] P Van Hentenryck, V. Saraswat, and Y. Deville. *The Design, Implementation, and Evaluation of the Constraint Language* `cc(FD)`, volume Constraint Programming: Basics and Trends. Springer Verlag, 1995.

[23] J Verschelde, P. Verlinden, and R. Cools. Homotopies Exploiting Newton Polytopes For Solving Sparse Polynomial Systems. *SIAM Journal on Numerical Analysis*, 31(3):915–930, 1994.