MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1542 May, 1995

Three Cuts for Accelerated Interval Propagation

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

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

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

Abstract

This paper addresses the problem of nonlinear multivariate root finding. In an earlier paper we describe a system called Newton which finds roots of systems of nonlinear equations using refinements of interval methods. The refinements are inspired by AI constraint propagation techniques. Newton is competitive with continuation methods on most benchmarks and can handle a variety of cases that are infeasible for continuation methods. This paper presents three "cuts" which we believe capture the essential theoretical ideas behind the success of Newton. This paper describes the cuts in a concise and abstract manner which, we believe, makes the theoretical content of our work more apparent. Any implementation will need to adopt some heuristic control mechanism. Heuristic control of the cuts is only briefly discussed here.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research was provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038. This research was also 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, an NSF National Young Investigator Award.

1 Introduction

In this paper we address the problem of finding solutions to large systems of nonlinear equations. This is an old problem with many applications and a large literature. The problem of of determining the existence of a solution to algebraic constraints on real numbers is known to be NP-hard and to be in PSPACE but it is not known whether this problem is in NP. In engineering applications it is generally sufficient to consider a simpler problem — the problem of determining the existence of floating point numbers that satisfy the given constraints up to the accuracy of floating point computations. This problem is easily shown to be NP-hard. However, since one can always guess the floating point numbers involved, floating point constraint satisfaction is in NP and hence NP-complete. Our work on floating point constraints emphasizes refinements of constraint propagation techniques for NP complete problems that have been developed in the AI and logic programming communities [1, 2, 3].

In an earlier paper we describe an implemented system called Newton [4]. Newton uses a branch and propagate algorithm whose propagation phase manipulates upper and lower bounds on the variables appearing in the given equations. Algorithms which manipulate upper and lower bounds are called interval methods — Newton is based on interval methods [5]. A well-studied alternative to interval methods is the so-called "continuation method" or "homotopy technique" [6, 14]. Table 1 summarizes our previously published results comparing Newton with earlier interval based systems and with continuation methods.[1] The benchmarks were taken from papers on numerical analysis [12], interval analysis [8, 9, 11], and continuation methods [14, 6, 13, 10]. The table shows that Newton outperforms an earlier interval based system and is competitive with continuation methods on the benchmarks tried. The novelty of Newton resides in the use of bound propagation techniques which are stronger than those used in previous interval systems. Newton can solve a variety of problems for which continuation methods are infeasible (those benchmarks with total degree greater than, say, 2^{40}).

Prob.	v	d	range	Newt	HRB	CONT
Broy	10	3^{10}	[-1,1]	1.6	18.2	
Broy	20	3^{20}	[-1,1]	4.2	?	
Broy	320	3^{320}	[-1,1]	113.7	?	
Broy	320	3^{320}	$[-10^8, 10^8]$	143.4	?	
MC	20	3^{20}	$[-4, 5]$	24.5	968.2	
MC	40	3^{40}	$[-4, 5]$	192.8	?	
MC	80	3^{80}	$[-4, 5]$	1752.6	?	
MC	80	3^{80}	$[-10^8, 0]$	1735.1	?	
i1	10	3^{10}	[-2,2]	0.1	14.3	
i2	20	3^{20}	[-1,2]	0.3	1821.2	
i3	20	3^{20}	[-2,2]	0.3	5640.8	
i4	10	6^{10}	[-1,1]	73.9	445.3	
i5	10	11^{10}	[-1,1]	0.1	33.6	
kin1	12	4608	$[-10^8, 10^8]$	14.2	1630.1	
kin2	8	256	$[-10^8, 10^8]$	353.1	4730.3	35.6
eco	4	18	$[-10^8, 10^8]$	0.6		1.1
eco	5	54	$[-10^8, 10^8]$	3.4		5.9
eco	6	162	$[-10^8, 10^8]$	22.5		50.2
eco	7	486	$[-10^8, 10^8]$	127.6		991.4
eco	8	1458	$[-10^8, 10^8]$	915.2		
eco	9	4374	$[-10^8, 10^8]$	8600.3		
comb	10	96	$[-10^8, 10^8]$	9.9		57.4
chem	5	108	$[0, 10^8]$	6.3		56.6
neuro	6	1024	$[-10, 10]$	0.9		5.0
neuro	6	1024	$[-10^3, 10^3]$	172.7		5.0

Table 1: Summary of the Experimental Results

Our earlier paper on Newton describes the algorithm used in sufficient detail to allow replication of the empirical results [4]. This includes a detailed description of the heuristics used. The details tend to obscure what we feel are the essential technical ideas behind the algorithms. In this paper we give an abstract presentation of three "cuts" where each cut specifies a way of inferring a new upper or lower bound on a variable. These abstract cuts, while insufficient in themselves to allow exact replication of our results, seem to capture the essence of Newton's ability to prune the search space and a description of the cuts should allow others to replicate the qualitative performance of the Newton system.

The heart of Newton's algorithm is a propagation process which iteratively improves known bounds on variables using a set of inference rules we call 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. The next section specifies three abstract cuts used in the Newton system. Section 3 briefly outlines how these cuts can be used in a branch and propagate procedure for findings all roots of a system of nonlinear equations. Section 3 is a simplification of the algorithm actually used in Newton.

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

[1] The Newton system was run on a `Sun Sparc 10` workstation to obtain all solutions to each benchmark system of equations. The column labeled Newton gives the run time (in seconds) of the Newton system on a given benchmark problem. 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 [7] 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. The column labeled `CONT` gives timings for a state of the art continuation method [14] 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 run time of each system 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). Further details on these timings can be found in [4].

generally use the term "interval" to refer to 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. Power expressions improve the accuracy of interval evaluation. 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. Clearly, if we are working with n variables then a box is a rectilinear subset of R^n. We also use $B[x := [a, b]]$ to denote the box which is identical to B except that it assigns the variable x the interval $[a, b]$. A "point" is an assignment of particular floating point numbers to each variable. We allow $B(x)$ to be interval of the form $[a,\ a]$. A point can be viewed as 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.

$$[a,\ b] + [c,\ d] = [a + c,\ b + d]$$
$$[a,\ b] - [c,\ d] = [a - d,\ b - c]$$
$$[a,\ b] * [c,\ d] = [min(ac,\ ad,\ bc,\ bd),\ max(ac,\ ad,\ bc,\ bd)]$$
$$[a,\ b]^n = \begin{cases} [a^n,\ b^n] & n \text{ odd} \\ [0, max(a^n,\ b^n)] & n \text{ even; } 0 \in [a, b] \\ [min(a^n, b^n),\ max(a^n, b^n)] & n \text{ even; } 0 \notin [a,\ b] \end{cases}$$

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.

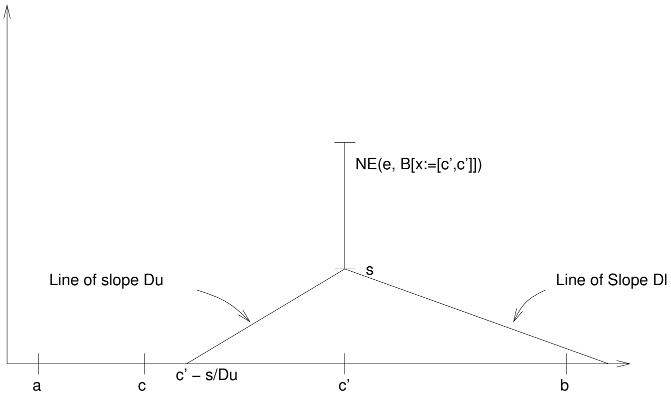

Figure 1: A Newton cut deriving $x \leq c' - \frac{s}{D_u}$

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 useful inferences.

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' 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. The snake cut is identical to the Newton cut except that de/dx is replaced by ds/dc'.[2] It turns out that the range of values of ds/dc' over the interval $[c,\ b]$ is often significantly less than the range of possible values of values of de/dx. This fact allows cuts to be larger.

In order to replace de/dx by ds/dc' we need to first 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 $s_0 + s_1x + s_2x^2 + \ldots + s_nx^n$ 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.

> *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 $c' - \frac{s(c')}{D_l} > b$. If $D_u \leq 0$ we can infer $x \leq c$ and if $D_u > 0$ we can infer $x \leq max(c,\ c' - \frac{s(c')}{D_u})$.

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. In our experience snake cuts are most useful in very large boxes and Newton cuts become more useful as the boxes get smaller.

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., [8, 7]. 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 for 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.

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

[2] 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.

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 Branch and Propagate Root Finding

To find a feasible point for a set of inequality constraints, or a root to a set of equations, one can use a simple branch and propagate procedure. One starts with a box large enough to contain all points of interest. One then reduces the box by applying cuts to derive new bounds. At some point one may decide to branch. To branch one selects a variable x. Let $[a,\ b]$ be the interval $B(x)$ where B is the box in use at the time of the branch. One then "splits" the box into the two branches $B[x := [a,\ \frac{a+b}{2}]]$ and $B[x := [\frac{a+b}{2}, b]]$. One then recursively computes a solution inside the first box, or if there is no such solution, a solution inside the second box. The procedure terminates when the current box is sufficiently small, e.g., every variable is assigned an interval of width less than, say, 10^{-8}. Newton uses a simple round-robin strategy to select the variable to split — down any branch of the search tree Newton splits a variable that has gone the longest time without splitting.

The basic branch and propagate procedure can be used with a wide variety of heuristics for finding useful cuts and for determining when to branch. Our experience with Newton indicates that one useful heuristic is to only perform cuts that result in at least a 10% reduction of an interval. This ensures that the number of cuts in a single propagation phase is at worst linear in the number of variables in the constraints. However, this same heuristic will force the procedure to branch before all possible cuts have been exhausted. In our experience such "early branching" is usually desirable.

Now consider selecting the values of c and c' in a cut on variable x with interval $[a,\ b]$. A simple strategy is to select c to be $b - \frac{b-a}{2^n}$ for some non-negative integer n and to select c' to be the midpoint of $[c, b]$. In practice we need only consider values of n up to 3 since we are only interested in cuts which reduce the interval by at least 10%. For a given variable we can start with $n = 0$ and if that cut fails increase n (up to 3) looking for a viable cut.

Combination cuts require selecting not only c and c' but also a linear combination of the constraints. Because computing appropriate coefficients for the linear combination can be expensive we suggest performing all possible Newton and snake cuts before attempting combination cuts. In performing combination cuts we assume that the input is given as a set of equations of the form $e_i = 0$. This allows one 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

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

Assuming that the matrix $[B_{i,j}]$ is nonsingular one can compute $[A_{i,j}]$ as of $[B_{i,j}]^{-1}$. 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 we can perform combination cuts on this *fixed* set of combined constraints selecting c and c' as specified above. In our experience it is best to perform only a single matrix inversion in any one propagation phase of the procedure — once the coefficients of the combinations constraints have been computed they should remain fixed throughout the remainder of that cutting phase. If the initial box is reasonably small, propagation relative to a fixed set of combined constraints will converge on an exact solution and only a single matrix inversion is needed. When the heuristics for selecting cuts fail to find any viable cuts the procedure branches.

The precise details of Newton's procedure for selecting cuts can be found in [4]. Although these details may be required to exactly replicate Newton's behavior, it seems likely that the qualitative performance of the system can be duplicated using only the heuristics outlined here.

4 Conclusion

This paper describes a branch and propagate algorithm to find all isolated solutions for a system of polynomial equations over the reals. Our techniques were originally inspired by local constraint propagation techniques used in AI and logic programming. The mathematics behind our techniques is quite straightforward and far less sophisticated than that underlying continuation methods. Yet our techniques seem to compare well with continuation methods on a wide variety of benchmarks.

References

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

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

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

[4] P. van Hentenryck, D. McAllester, and D. Kapur *Solving Polynomial Systems using a Branch and Prune Approach.* to appear, *SIAM Journal of Numerical Analysis*, also available through http://www.ai.mit.edu/people/dam/dam.html

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

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

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

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

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

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

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

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

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

[14] 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.