ROBUST PROCESS SIMULATION

USING

INTERVAL METHODS

Carol A. Schnepper*

and

Mark A. Stadtherr**

Department of Chemical Engineering
University of Illinois
600 S. Mathews Avenue
Urbana, IL 6l80l
U.S.A.

*Current Address: American Air Liquide, Chicago Research Center, 5230 S. East Avenue, Countryside, IL 60525, U.S.A.

**Author to whom correspondence should be addressed

2/94
(revised 1/95)

ABSTRACT

Ideally, for the needs of robust process simulation, one would like a nonlinear equation solving technique that can find any and all roots to a problem, and do so with mathematical certainty. In general, currently used techniques do not provide such rigorous guarantees. One approach to providing such assurances can be found in the use of interval analysis, in particular the use of interval Newton methods combined with generalized bisection. However, these methods have generally been regarded as extremely inefficient. Motivated by recent progress in interval analysis, as well as continuing advances in computer speed and the availability of parallel computing, we consider here the feasibility of using an interval Newton/generalized bisection algorithm on process simulation problems. An algorithm designed for parallel computing on an MIMD machine is described, and results of tests on several problems are reported. Experiments indicate that the interval Newton/generalized bisection method works quite well on relatively small problems, providing a powerful method for finding all solutions to a problem. For larger problems, the method performs inconsistently with regard to efficiency, at least when reasonable initial bounds are not provided.

INTRODUCTION

The central problem in steady-state process simulation is the solution of a system of nonlinear equations $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, where $\boldsymbol{f} \in \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{R}^n$. Newton and quasi-Newton methods are extensively used in this context, but may not reliably converge, especially if the initial guess is poor or if singular points are encountered. To improve convergence in these circumstances, various methods have been used. These include trust-region techniques such as Powell's dogleg method (Powell, 1970; Chen and Stadtherr, 1981), homotopy-based methods (e.g., Wayburn and Seader, 1987; Kuno and Seader, 1988), and techniques based on iterative mathematical programming, as reviewed recently by Bullard and Biegler (1991).

An additional difficulty is that in process simulation there are invariably upper and lower bounds on the variables, $\boldsymbol{x}^L \le \boldsymbol{x} \le \boldsymbol{x}^U$, violation of which may cause some functions to become undefined. Bounds are often dealt with in an *ad hoc* manner (e.g., Zitney and Stadtherr, 1988) involving truncation or reflection of the correction step. A more natural way of dealing with bounds is to use the iterative mathematical programming approach (e.g., Bullard and Biegler, 1991; Swaney and Wilhelm, 1990), in which case the bounds become an integral part of the problem. While a number of these techniques demonstrate excellent global convergence properties in practice, none offer a rigorous mathematical guarantee of convergence, with the exception of the method of Swaney and Wilhelm (1990), which makes use of bounds generated using interval arithmetic, and if necessary uses a bisection strategy to guarantee convergence.

Another difficulty in solving the nonlinear equation system arising in process simulation is that it may have multiple solutions. With the exception of homotopy-based methods, none of the techniques mentioned above are designed for finding multiple solutions when they exist. While in practice homotopy-based methods are frequently able to locate all solutions to a problem, they offer no guarantee that all solutions have been found, except in special cases.

Ideally then, what is desired is a technique that can find, *with mathematical certainty*, any and *all* solutions to a system of nonlinear equations lying within the variable bounds. The techniques of interval analysis provide just such a class of methods (Kearfott, 1990a), namely interval Newton methods combined with generalized bisection. In general, the thought of using such techniques for process simulation problems has been dismissed forthwith, on the assumption that they would be extremely inefficient. However, recent advances in interval methods (e.g., Neumaier, 1990; Kearfott and Novoa, 1990), together with the continuing rapid advance in computer speed and the availability of parallel computing, may make such methods viable, as originally suggested by Schnepper and Stadtherr (1990). In this paper we report the results of a *feasibility* study on the use of interval Newton/generalized bisection methods for solving process simulation problems.

INTERVAL COMPUTATIONS

In this section we provide a brief introduction to interval computations, with emphasis on those aspects needed for discussion of the nonlinear equation solving problem addressed here. For a more complete and detailed discussion, the reader is referred to the excellent recent monographs by Neumaier (1990), Hansen (1992) and Alefeld and Herzberger (1983), and the classic work of Moore (1966).

A real *interval number*, or simply *interval*, X, can be defined by $X = [a,b] = \{x \in \mathbb{R} \mid a \le x \le b\}$, where $a,b \in \mathbb{R}$ and $a \le b$. The set of all such real intervals is denoted $\mathbb{IR}$. Note that any $x \in \mathbb{R}$ can be represented by a degenerate (or thin) interval $X = [x,x] \in \mathbb{IR}$. Thus, whenever interval and real numbers appear in the same expression here, it is understood that the real numbers are treated as degenerate intervals. Interval vectors and matrices are analogous to

real vectors and matrices. Thus, an interval vector $\boldsymbol{X} = (X_i) = (X_1,X_2,...,X_n)^{\mathrm{T}} \in \mathbb{IR}^n$ has n real interval components $X_i \in \mathbb{IR}$, $i = 1,2,...,n$. Similarly an interval matrix $\boldsymbol{A} = (A_{ij}) \in \mathbb{IR}^{n \times m}$ has real interval elements $A_{ij} \in \mathbb{IR}$, $i = 1,2,...,n$ and $j = 1,2,...,m$. Geometrically, an interval vector can be interpreted as an n-dimensional rectangle, which is frequently referred to as a *box*.

Some useful definitions are: 1. The midpoint $\bar{x} \in \mathbb{R}$ of the interval number $X = [a,b]$ is $\bar{x} = (a + b)/2$. 2. The midpoint $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ of the interval vector $\boldsymbol{X} = (X_i)$ is $\bar{\boldsymbol{x}} = (\bar{x}_1,\bar{x}_2,...,\bar{x}_n)^{\mathrm{T}}$. 3. The width of the interval number X, $w(X) \in \mathbb{R}$, is $w(X) = b - a$. 4. The width (or diameter) of the interval vector $\boldsymbol{X}$, $w(\boldsymbol{X}) \in \mathbb{R}$, is $w(\boldsymbol{X}) = \max_i w(X_i)$. 5. The absolute value $|X| \in \mathbb{R}$ of the interval number X is $|X| = \max \{|a|,|b|\}$. 6. The vector norm $\|\boldsymbol{X}\| \in \mathbb{R}$ of the interval vector $\boldsymbol{X}$ is $\max_i |X_i|$. 7. The volume of the box $\boldsymbol{X}$ is $V(\boldsymbol{X}) = \prod_i w(X_i)$.

Interval arithmetic is an extension of real arithmetic. For an elementary real arithmetic operation op $\in \{+, -, *, /\}$, the corresponding interval operation for intervals $X = [a,b]$ and $Y = [c,d]$ is defined by

$$X \text{ op } Y = \{x \text{ op } y \mid x \in X, y \in Y\}. \tag{1}$$

Thus, in terms of the endpoints of X and Y, the following formulae can be developed:

$$X + Y = [a + c, b + d]$$

$$X - Y = [a - d, b - c]$$

$$X * Y = [\min (ac, ad, bc, bd), \max (ac, ad, bc, bd)]$$

$$X / Y = [a,b] * [1/d, 1/c], \quad 0 \notin [c,d].$$

For X / Y when $0 \in Y$, an extended interval arithmetic is available (Hansen, 1968) which is useful in the execution of the interval Newton methods discussed below. The extended arithmetic produces the same set as defined by Eq. (1).

The foregoing assumes that we are able to compute the endpoints of an interval result

exactly. Of course, when this is done on a computer, there may be round-off problems, so steps must be taken to ensure that the resulting intervals enclose the exact values. The use of rounded-interval arithmetic solves this problem. Essentially a directed outward rounding is used, so that when a lower endpoint a is computed, it is rounded down to the largest machine-representable number less than or equal to a, and when an upper endpoint b is computed, it is rounded up to the smallest machine-representable number greater than or equal to b.

As emphasized by Kearfott (1989,1990a), a key feature of interval methods is the ability to compute *inclusion monotone interval extensions* $F(\boldsymbol{X})$ of real functions $f(\boldsymbol{x})$. An inclusion monotone interval extension $F(\boldsymbol{X})$ of $f(\boldsymbol{x})$ has the property $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \boldsymbol{X}\} \subseteq F(\boldsymbol{X})$, and the property that $\boldsymbol{X} \subset \boldsymbol{Y}$ implies that $F(\boldsymbol{X}) \subset F(\boldsymbol{Y})$. Inclusion monotone interval extensions of rational real functions can be determined simply by replacing $\boldsymbol{x}$ with $\boldsymbol{X}$ and replacing the real arithmetic operations with the corresponding elementary interval operations. Inclusion monotone interval extensions of transcendental functions can also be readily obtained, and software for computing the inclusion monotone interval extensions of the elementary functions (e.g., exponentiation, logarithm, exponential) is available. Note that if we compute (using rounded-interval arithmetic) an inclusion monotone interval extension $F(\boldsymbol{X})$ of $f(\boldsymbol{x})$, and $0 \notin F(\boldsymbol{X})$ then this is proof that there is no root of $f(\boldsymbol{x}) = 0$ in $\boldsymbol{X}$. It is safe to assume that all real functions of interest to us here have easily computed inclusion monotone interval extensions, and we shall assume below that all interval functions used are inclusion monotone interval extensions of the corresponding real functions.

The interval extension $F(\boldsymbol{X})$ encloses all values of $f(\boldsymbol{x})$ for $\boldsymbol{x} \in \boldsymbol{X}$. In general the quality (tightness) of this enclosure depends on the form in which $F(\boldsymbol{X})$ is expressed and evaluated. For example, if $f(\boldsymbol{x}) = x_1(x_2 - x_3) = x_1x_2 - x_1x_3$, alternate interval extensions are $F(\boldsymbol{X}) = X_1 * (X_2 - X_3)$

and $F(\boldsymbol{X}) = (X_1 * X_2) - (X_1 * X_3)$. Evaluating the first of these, for say $X_1 = X_2 = X_3 = [1,2]$ yields $[1,2] * ([1,2] - [1,2]) = [1,2] * ([-1,1]) = [-2,2]$, which is precisely the range of $f(\boldsymbol{x})$ over $\boldsymbol{X}$. However, evaluating the second expression yields $([1,2] * [1,2]) - ([1,2] * [1,2]) = [1,4] - [1,4] = [-3,3]$, which contains the range of $f(\boldsymbol{x})$ over $\boldsymbol{X}$ but is an overestimate. In general, such overestimations may occur when an interval variable appears more than once in an expression. This so-called "dependence problem" occurs because interval arithmetic essentially treats all occurrences of a variable independently rather than recognizing their dependence.

Finally, we consider the nature of interval equations. Take, for example, the system of linear interval equations $\boldsymbol{A}x = \boldsymbol{B}$, where $\boldsymbol{A} \in \mathbb{IR}^{n \times n}$ and $\boldsymbol{B} \in \mathbb{IR}^{n}$. If $\tilde{A} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{R}^{n}$, then the solution to the linear interval system is the set $S = \{\boldsymbol{x} \mid \tilde{A}\boldsymbol{x} = \boldsymbol{b}, \tilde{A} \in \boldsymbol{A}, \boldsymbol{b} \in \boldsymbol{B}, \boldsymbol{x} \in \mathbb{R}^{n}\}$. In general this set is not an interval box, and may have a very complex geometry. Thus, to "solve" the linear interval system, one instead seeks a box $\boldsymbol{X}$ containing S; there are various techniques for doing this, producing enclosures of S of varying quality, as discussed further below. Note that if the interval matrix $\boldsymbol{A}$ contains any matrix $\tilde{A}$ that is singular, then the solution set S will be unbounded. It should also be noted that in considering an interval equation, the term "equation" may need to be interpreted loosely. For example, consider the linear system $Ax = B$ in one variable, with $A = [2,3]$ and $B = [3,4]$. The solution set S is $X = B/A = [3,4]/[2,3] = [1,2]$. However, if this is substituted back into the original "equation", the result is $B = [2,3] * [1,2] = [2,6]$, which is not the original B, but only contains it. This occurs because of the dependency problem discussed above.

INTERVAL NEWTON METHODS

Consider the solution of the system of real nonlinear equations $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, where it is desired to find all solutions in an specified initial box $\boldsymbol{X}^{(0)}$. The basic idea in interval Newton methods is to solve the linear interval equation system

$$\boldsymbol{F'}(\boldsymbol{X}^{(k)})(\boldsymbol{N}^{(k)} - \boldsymbol{x}^{(k)}) = -\boldsymbol{f}(\boldsymbol{x}^{(k)}) \tag{2}$$

for $\boldsymbol{N}^{(k)}$, where k is an iteration counter, $\boldsymbol{F'}(\boldsymbol{X}^{(k)})$ is a suitable interval extension of the real Jacobian $J(\boldsymbol{x})$ of $\boldsymbol{f}(\boldsymbol{x})$ over the current box $\boldsymbol{X}^{(k)}$, and $\boldsymbol{x}^{(k)}$ is a point in the interior of $\boldsymbol{X}^{(k)}$, usually taken to be the midpoint. It can be shown (Moore, 1966) that any root $\boldsymbol{x}^{*} \in \boldsymbol{X}^{(k)}$ of $\boldsymbol{f}(\boldsymbol{x})$ is also contained in $\boldsymbol{N}^{(k)}$. This suggests the iteration

$$\boldsymbol{X}^{(k+1)} = \boldsymbol{X}^{(k)} \cap \boldsymbol{N}^{(k)}. \tag{3}$$

The various interval Newton methods differ in how they determine $\boldsymbol{N}^{(k)}$ from Eq. (2) and thus in the tightness with which $\boldsymbol{N}^{(k)}$ encloses the solution set of Eq. (2). $\boldsymbol{N}^{(k)}$ may be computed using interval Gaussian elimination; frequently, however, it is computed component by component using a Gauss-Seidel-like procedure, as described below. In this case, the intersection in Eq. (3) may be applied after each individual component of $\boldsymbol{N}^{(k)}$ is determined, and the result used in computing subsequent components of $\boldsymbol{N}^{(k)}$. This strategy yields a truncated $\boldsymbol{N}^{(k)}$ that does not enclose the complete solution set of Eq. (2), but that does enclose the part of the solution set necessary for the interval Newton iteration. Preconditioning Eq. (2), as discussed below, is frequently used (e.g., Kearfott, 1990b) to obtain a tighter enclosure. Note that if $\boldsymbol{X}^{(k+1)} = \varnothing$, this means that there is no zero of $\boldsymbol{f}(\boldsymbol{x})$ in the box $\boldsymbol{X}^{(k)}$, since if there was a zero of $\boldsymbol{f}(\boldsymbol{x})$ in $\boldsymbol{X}^{(k)}$ it would have to also be in $\boldsymbol{N}^{(k)}$, resulting in a nonempty set $\boldsymbol{X}^{(k+1)}$.

While the iteration scheme given by Eqs. (2)-(3) can be used to tightly enclose a solution, what is of most significance here is the power of Eq. (2) to provide an existence and uniqueness

test. For several techniques for finding $\boldsymbol{N}^{(k)}$ from Eq. (2), it can be proven (Neumaier, 1990) that if $\boldsymbol{N}^{(k)} \subset \boldsymbol{X}^{(k)}$, then there is a *unique* zero of $\boldsymbol{f}(\boldsymbol{x})$ in $\boldsymbol{X}^{(k)}$, and furthermore that Newton's method with real arithmetic *will converge* to that solution starting from *any* point in $\boldsymbol{X}^{(k)}$. Thus, if $\boldsymbol{N}^{(k)}$ is determined using one of these techniques, the computation can be used as a root inclusion test for any interval $\boldsymbol{X}^{(k)}$. If $\boldsymbol{X}^{(k)} \cap \boldsymbol{N}^{(k)} = \varnothing$, then there is no root in $\boldsymbol{X}^{(k)}$; if $\boldsymbol{N}^{(k)} \subset \boldsymbol{X}^{(k)}$, then there is exactly one root and Newton's method with real arithmetic will find it; otherwise, no conclusion can be drawn. In the last case, one could then repeat the root inclusion test on the next interval Newton iterate $\boldsymbol{X}^{(k+1)}$, assuming it is sufficiently smaller than $\boldsymbol{X}^{(k)}$, or one could bisect $\boldsymbol{X}^{(k+1)}$ and repeat the root inclusion test on the resulting intervals. This is the basic idea of interval Newton/generalized bisection methods. Assuming it has been properly implemented, and assuming that $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ has a finite number of real solutions in the specified initial box, the interval Newton/generalized bisection method can find *with mathematical certainty* any and all such solutions to a specified tolerance, or can determine *with mathematical certainty* that there are no solutions in the given box (Kearfott, 1987a,1989,1990a).

The technique used here for computing $\boldsymbol{N}^{(k)}$ is the preconditioned Gauss-Seidel-like technique developed by Hansen and Sengupta (1981) and Hansen and Greenburg (1983). Eq. (2) is first preconditioned with a real matrix $Y^{(k)}$, which is often chosen to be the inverse of the matrix formed from the midpoints of the components of $\boldsymbol{F}'(\boldsymbol{X}^{(k)})$. Eq. (2) now becomes

$$Y^{(k)}\boldsymbol{F}'(\boldsymbol{X}^{(k)})(\boldsymbol{N}^{(k)} - \boldsymbol{x}^{(k)}) = -Y^{(k)}\boldsymbol{f}(\boldsymbol{x}^{(k)}). \tag{4}$$

Defining $\boldsymbol{M} = Y^{(k)}\boldsymbol{F}'(\boldsymbol{X}^{(k)})$ and $\boldsymbol{b} = Y^{(k)}\boldsymbol{f}(\boldsymbol{x}^{(k)})$, the interval Gauss-Seidel procedure proceeds component by component according to

$$N_i^{(k)} = x_i^{(k)} - \frac{b_i + \sum_{j=1}^{i-1} M_{ij}\left(X_j^{(k+1)} - x_j^{(k+1)}\right) + \sum_{j=i+1}^{n} M_{ij}\left(X_j^{(k)} - x_j^{(k)}\right)}{M_{ii}} \tag{5a}$$

and

$$X_i^{(k+1)} = N_i^{(k)} \cap X_i^{(k)} \tag{5b}$$

for $i = 1,...,n$. If $M_{ii} \ni 0$ in Eq. (5a), then extended interval arithmetic must be used. Note that after component i of $\boldsymbol{N}^{(k)}$ is computed using Eq. (5a), the intersection in Eq. (5b) is performed, and the result used in the computation of subsequent components of $\boldsymbol{N}^{(k)}$. As noted above, this yields a truncated $\boldsymbol{N}^{(k)}$ that, while not enclosing the full solution set of Eq. (2), does enclose the part of the solution set necessary for the interval Newton iteration. A proof of the existence and uniqueness test for this method of determining $\boldsymbol{N}^{(k)}$ is summarized by Neumaier (1990). Neumaier (1990) also shows that this method always yields a tighter interval $\boldsymbol{N}^{(k)}$ than the Krawczyk (1969) method, an alternative method for determining $\boldsymbol{N}^{(k)}$ that has been extensively studied.

Note that if $\boldsymbol{X}^{(k)}$ contains a singular point, that is if the interval Jacobian $\boldsymbol{F'}(\boldsymbol{X}^{(k)})$ contains a singular matrix, then the complete solution set of Eq. (2), and its enclosure $\boldsymbol{N}^{(k)}$, will be unbounded. In this case, the existence and uniqueness condition, $\boldsymbol{N}^{(k)} \subset \boldsymbol{X}^{(k)}$, will never be satisfied, nor will the interval Newton iteration in Eq. (3) be of any use. For this situation, the truncated enclosure $\boldsymbol{N}^{(k)}$, found using the Gauss-Seidel procedure above, is useful since all or some of its components may still be bounded despite the singularity. Thus, Eq. (3) may still be useful. It can be seen from Eq. (5a) that, whether or not there is a singular point in $\boldsymbol{X}^{(k)}$, all components of the truncated $\boldsymbol{N}^{(k)}$ are bounded if $0 \notin M_{ii}$ for all i, and if $0 \in M_{ii}$ for any i, then the first i-1 components of the truncated $\boldsymbol{N}^{(k)}$ are bounded. However, even when the truncated

$\boldsymbol{N}^{(k)}$ is used, if there is a singular point in $\boldsymbol{X}^{(k)}$, then the existence and uniqueness condition, $\boldsymbol{N}^{(k)} \subset \boldsymbol{X}^{(k)}$, will never be satisfied. This means that the existence and uniqueness condition cannot be used to identify a solution at a singular point. Thus, to deal with this possibility, in implementing interval Newton/generalized bisection a stopping criterion based on the norm of $F(\boldsymbol{X}^{(k)})$ must be included (Step 2 in the root inclusion test procedure below).

IMPLEMENTATION

In this section we describe an implementation of an interval Newton/generalized bisection (IN/GB) algorithm for use in process simulation problems. While for relatively small problems, IN/GB is usually efficient (Kearfott, 1987b; Kearfott and Novoa, 1990), for larger problems efficiency is less predictable (Kearfott, 1989; Kearfott and Novoa, 1990). In fact, for a properly implemented IN/GB method applied to an initial box with a finite number of roots, the only mode of failure is an excessive computational requirement. However, bisection provides natural opportunities for large-grained parallel computation, and by taking advantage of this, computation times can potentially be significantly decreased. Thus we implement an algorithm here based on parallel computing, though it can also be performed on a single processor for small problems.

The key to designing efficient large-grained parallel algorithms is identifying independent tasks, having approximately equal computational requirements, for concurrent execution. For the parallel algorithm presented here, independent tasks at the subroutine programming level arise from application of the bisection process. In a serial implementation of an IN/GB algorithm, after a bisection step is completed, one of the subboxes is stored on a stack for later consideration while the program continues processing the other subbox according to the interval Gauss-Seidel method. However, the calculations performed on any given subbox are independent of the calculations performed on any other subbox, and the boxes on the stack may be considered in

any order. Therefore, on computers with the appropriate multiple-processor architecture, each available processor may execute the Gauss-Seidel procedure on a different box from the stack.

The starting point for our implementation was the code INTBIS (Kearfott and Novoa, 1990), an implementation of a serial IN/GB algorithm (Kearfott, 1987a,b). INTBIS was designed for the serial solution of small, dense systems of polynomials. Thus, a number of extensions were necessary for application to process simulation problems. Primarily these involve the development of interval extensions of the appropriate function types, and a provision for efficient handling of sparse matrices.

To provide a library of interval functions appropriate for process simulation problems, we began with the set of real function types needed for function and Jacobian evaluation in the equation-based flowsheeting code SEQUEL-II (Zitney and Stadtherr, 1988). Using interval arithmetic routines supplied with INTBIS, a library of routines for computing interval extensions of these real functions was developed. INTBIS was then modified to use the sparse storage scheme used in SEQUEL-II. The efficient sparse solver LU1SOL (Stadtherr and Wood, 1984; Chen and Stadtherr, 1984; Kaijaluoto *et al.*, 1989) was used in performing the preconditioning and in connection with the point-Newton iteration done in intervals having a positive root inclusion test. Before summarizing the algorithm used, we discuss some of its components in more detail.

Two tolerances are supplied to the algorithm. The tolerance ε essentially represents the smallest allowable box dimension and is used to distinguish between roots that are close together. Roots that are closer than ε together in some dimension may lie in the same small box. The stopping criterion using ε uses the scaled diameter $d(\boldsymbol{X}) = \max_i \{w(X_i)/\max(|X_i|,1)\}$. A second tolerance, ε_F is needed because, as explained above, an interval Newton inclusion test will not

confirm a solution at which the Jacobian is singular. Thus if there is a box $\boldsymbol{F}(\boldsymbol{X}) \ni \mathbf{0}$ that has become sufficiently small, based on $\|\boldsymbol{F}(\boldsymbol{X})\| < \varepsilon_F$, then the root inclusion test is taken to be satisfied. More information concerning the computational use of ε and ε_F can be found in Kearfott (1987b).

The root inclusion test is called repeatedly in the main algorithm. It is outlined as a separate procedure here. For a given box $\boldsymbol{X}$, encountered at any point in the solution process, the root inclusion test returns a result T($\boldsymbol{X}$) = *true*, *false*, or *unknown*. If T($\boldsymbol{X}$) is set *true* or *unknown* in Step 3, then the next interval Newton iterate $\boldsymbol{X}^+$ is also returned. The root inclusion test procedure is as follows:

1. Compute the interval extension $\boldsymbol{F}(\boldsymbol{X})$. If $\mathbf{0} \notin \boldsymbol{F}(\boldsymbol{X})$ then set T($\boldsymbol{X}$) = *false* and return, since there can be no zero of $\boldsymbol{f}(\boldsymbol{x})$ in this interval.
2. If $\|\boldsymbol{F}(\boldsymbol{X})\| < \varepsilon_F$, then set T($\boldsymbol{X}$) = *true*. Set $\boldsymbol{X}^+ = \boldsymbol{X}$ and return.
3. Perform the preconditioned Gauss-Seidel test to compute the truncated $\boldsymbol{N}$, component by component, as indicated by Eqs. (4) and (5a-b). The results of the intersection in Eq. (5b) are stored as components of $\boldsymbol{X}^+$. If for any value of i during the Gauss-Seidel test, the intersection in Eq. (5b) yields $X_i^+ = \varnothing$, then set T($\boldsymbol{X}$) = *false* and return without computing additional elements of $\boldsymbol{X}^+$. If $\boldsymbol{N} \subset \boldsymbol{X}$, then set T($\boldsymbol{X}$) = *true* and return; otherwise set T($\boldsymbol{X}$) = *unknown* and return.

For the preconditioning matrix $Y^{(k)}$ in Eq. (4), we use the inverse of $J_m = J(\bar{\boldsymbol{x}})$, the Jacobian of $\boldsymbol{f}(\boldsymbol{x})$ evaluated at the midpoint of the current box $\boldsymbol{X}$. J_m is an easily computed approximation of the matrix formed from the midpoints of the components of $\boldsymbol{F}'(\boldsymbol{X}^{(k)})$, the inverse of which is often suggested (e.g., Neumaier, 1990) as a good preconditioner. Other preconditioners for improving the performance of the Gauss-Seidel procedure are being developed

(Kearfott, 1990b; Kearfott *et al*., 1991a); however these are computationally more expensive than the inverse Jacobian, especially for large problems. INTBIS computes and stores the entire inverse Jacobian explicitly before proceeding with the Gauss-Seidel test. However, this is undesirable in dealing with relatively large and sparse problems. Thus, since in Eq. (5a) $\boldsymbol{M}$, and thus $Y^{(k)} = J_m^{-1}$, is needed only a row at a time, and may not be needed for all rows if for some $i < n$ the intersection in Eq. (5b) is the empty set, we compute $Y^{(k)}$ a row at a time as needed from an LU factorization of J_m. Should J_m be singular, the result of the root inclusion test is treated as *unknown*, and the box is bisected.

If the result of the root inclusion test is *unknown* then it must be decided whether to bisect the interval tested or repeat the test on the next interval Newton iterate. This decision is based on the ratio ρ of the volumes of $\boldsymbol{X}^{(k+1)}$ and $\boldsymbol{X}^{(k)}$:

$$\rho = \frac{V(\boldsymbol{X}^{(k+1)})}{V(\boldsymbol{X}^{(k)})} = \prod_i^n \left(\frac{w(X_i^{(k+1)})}{w(X_i^{(k)})} \right)$$

In order to avoid computing the ratio of two very small numbers in determining ρ, if $w(X_i^{(k)}) < \varepsilon$ for any i, then the corresponding factor in the product above is set to one. In INTBIS the Gauss-Seidel test is repeated on $\boldsymbol{X}^{(k+1)}$ using the same inverse Jacobian preconditioner if $\rho < 0.4$, and is repeated using a reevaluated preconditioner if $0.4 \le \rho \le 0.6$. Otherwise $\boldsymbol{X}^{(k+1)}$ is bisected. In our algorithm, since the preconditioner is computed just a row at a time as needed, and is thus never computed and stored explicitly, there is no strong incentive in reusing the same preconditioner. Thus, we reevaluate the preconditioner and repeat the Gauss-Seidel test on $\boldsymbol{X}^{(k+1)}$ whenever $\rho \le 0.6$, and bisect $\boldsymbol{X}^{(k+1)}$ otherwise.

When a box $\boldsymbol{X} = (X_i)$ is bisected, the resulting sub-boxes $\boldsymbol{X}_U$ and $\boldsymbol{X}_L$ are $\boldsymbol{X}_U = (X_1,X_2,...,[\overline{x}_q,b_q],...X_n)^{\mathrm{T}}$ and $\boldsymbol{X}_L = (X_1,X_2,...,[a_q,\overline{x}_q],...X_n)^{\mathrm{T}}$, where $X_q = [a_q,b_q]$ and q is the coordinate

direction chosen for bisection. Three strategies were tested for choosing the coordinate direction for bisection. In the first scheme, the coordinate with the widest component interval is chosen; that is, q is chosen so that $w(X_q) = w(\boldsymbol{X})$. In the second scheme, the coordinate with the largest scaled diameter is chosen; that is, q is chosen so that $d(X_q) = d(\boldsymbol{X})$. The third scheme attempts to determine the coordinate direction in which the values of the component functions f_i change most rapidly. After completion of the Gauss-Seidel test, this is accomplished by computing for each coordinate direction j the quantity $s_j = w(X_j^{(k+1)})\|\boldsymbol{F}_j'(\boldsymbol{X}^{(k)})\|$, where $\boldsymbol{F}_j'(\boldsymbol{X}^{(k)})$ is the j-th column of the interval Jacobian $\boldsymbol{F}'(\boldsymbol{X}^{(k)})$. Then q is chosen so that $s_q = \max_j s_j$. Our computational experiments on process simulation problems (Schnepper, 1992) have shown the third strategy to be the most effective, and that is what is used here.

Algorithm (serial)

The basic IN/GB algorithm is outlined first in serial form. Its parallel implementation will then be described. In the following steps, T($\boldsymbol{X}$) and $\boldsymbol{X}^+$ are as defined in the root inclusion test procedure above, $\mathcal{L}$ represents a list of boxes that have been determined to contain a solution, and $\boldsymbol{X}_b$ is an geometrically similar expansion of $\boldsymbol{X}^+$ constructed to have the same midpoint as $\boldsymbol{X}^+$ but with component widths four times as large. This serial algorithm essentially performs a depth-first search in the binary tree generated by the bisection.

I. Initialization

(a) Set $\boldsymbol{X} = \boldsymbol{X}^{(0)}$, the initial box.

(b) Set values for tolerances ε and ε_F.

(c) Push $\boldsymbol{X}$ onto the stack of boxes to be processed.

II. Check stack.

(a) If the stack is empty, then go to Step IV.

III. Process a box. Steps III(b), III(g), and III(e(2)) are used to deal with roots on or near a boundary.

(a) Remove a box $\boldsymbol{X}$ from the stack.

(b) If $d(\boldsymbol{X}) < \varepsilon/4$ and $\boldsymbol{X}$ has a nonnull intersection with a box $\boldsymbol{X}^*$ in the list $\mathcal{L}$ of solution-containing boxes, and $d(\boldsymbol{X}) + d(\boldsymbol{X}^*) \le \varepsilon/4$, then go to Step III(i).

(c) Call the root inclusion test procedure to determine $T(\boldsymbol{X})$ and $\boldsymbol{X}^+$.

(d) If $T(\boldsymbol{X})$ = *false*, then go to Step III(i).

(e) If $T(\boldsymbol{X})$ = *true*, then:

(1) Store $\boldsymbol{X}^+$ in the solution-containing list $\mathcal{L}$.

(2) If $\boldsymbol{X}^+$ has a nonnull intersection with a box $\boldsymbol{X}^*$ in $\mathcal{L}$, and $d(\boldsymbol{X}^+) + d(\boldsymbol{X}^*) \le \varepsilon/4$, then delete $\boldsymbol{X}^*$ from $\mathcal{L}$.

(3) Go to Step III(i).

(f) If $T(\boldsymbol{X})$ = *unknown* and $d(\boldsymbol{X}^+) \ge \varepsilon/16$, then compute the ratio $\rho = V(\boldsymbol{X}^+)/V(\boldsymbol{X})$:

(1) If the volume ratio $\rho > 0.6$, then go to Step III(h).

(2) If the volume ratio $\rho \le 0.6$, then reset $\boldsymbol{X} = \boldsymbol{X}^+$, and return to Step III(c).

(g) If $T(\boldsymbol{X})$ = *unknown* and $d(\boldsymbol{X}^+) < \varepsilon/16$, then adjust for roots on a boundary:

(1) Replace $\boldsymbol{X}^+$ by the expanded box $\boldsymbol{X}_b$ with the same midpoints as $\boldsymbol{X}^+$ but four times as large.

(2) Delete from $\mathcal{L}$ all $\boldsymbol{X}^* \in \mathcal{L}$ for which $\boldsymbol{X}^* \cap \boldsymbol{X}_b \ne \varnothing$ and $d(\boldsymbol{X}_b) + d(\boldsymbol{X}^*) \le \varepsilon/4$.

(3) Store $\boldsymbol{X}_b$ in $\mathcal{L}$.

(4) Go to Step III(i).

(h) Bisect $\boldsymbol{X}^+$:

(1) Form sub-boxes $\boldsymbol{X}_U$ and $\boldsymbol{X}_L$.

(2) Store $\boldsymbol{X}_U$ on the stack of boxes for later processing.

(3) Store $\boldsymbol{X}_L$ on the stack of boxes for later processing.

(i) Go to Step II.

IV. Final Point Solution.

(a) If the solution-containing list $\mathcal{L}$ is empty, report that there are no solutions in $\boldsymbol{X}^{(0)}$, and stop.

(b) Execute the traditional point Newton method for each box stored in the solution-containing list $\mathcal{L}$, starting at the midpoint of the box.

(c) Report solution(s), and stop.

In Step IV(b), a safeguard may be added to deal with any $\boldsymbol{X}^+$ added to $\mathcal{L}$ because T($\boldsymbol{X}$) was set *true* in Step 2 of the root inclusion test procedure. In this case, $\boldsymbol{X}^+$ was added to $\mathcal{L}$ because the function norm became sufficiently close to zero, not because the existence and uniqueness test was satisfied. This means that there is no guarantee that the point Newton method will converge in this box. Thus, if the diameter of such an $\boldsymbol{X}^+$ is sufficiently small, one may wish to accept its midpoint as the final point solution. If the diameter of this $\boldsymbol{X}^+$ is not small enough, then a tighter tolerance ε_F can be used.

Algorithm (parallel)

The parallel algorithm described here is designed for a MIMD computer supporting globally shared memory. It is also assumed that each processor has its own local workspace.

In the algorithm, Steps I and II are performed on one "controlling" processor, while Step III can be performed on the remaining "working" processors as well as on the controlling processor. Step II may be executing in parallel with Step III, and several Step III tasks may be executing in parallel at any given time.

I. Initialization. Execute on the controlling processor.

(a-c) Same as serial algorithm Steps I(a-c).

(d) Set $L = 1$. L is the new box counter, indicating the number of new boxes placed on the stack since boxes were last assigned for processing in Step II(b(1)); thus L represents the number of boxes currently unassigned for processing.

II. Check stack. Execute on the controlling processor.

(a) If the stack is empty and $L = 0$, then:

(1) If any working processors are active, then:

(i) Wait for one to finish.

(ii) Return to Step II(a).

(2) If no working processors are active, then go to Step IV.

(b) If the stack is not empty or if $L \geq 1$, then:

(1) If $L = 0$, go to Step III.

(2) If $L \geq 1$, there are new boxes on the stack that have not yet been assigned for processing. Assign these L boxes for processing at one box per processor.

(3) Activate up to $L-1$ inactive working processors and start them executing Step III in parallel on boxes now assigned for processing.

(4) Reset new box counter $L = 0$, and execute Step III.

III. Process a box. Execute in parallel, one processor per box, on working processors and the controlling processor.

(a) Remove from the stack a box $\boldsymbol{X}$ that has been assigned for processing.

(b-g) Same as serial algorithm Steps (b-g).

(h) Bisect $\boldsymbol{X}^+$:

(1) Form sub-boxes $\boldsymbol{X}_U$ and $\boldsymbol{X}_L$.

(2) Store $\boldsymbol{X}_U$ on the stack of boxes for later processing.

(3) Set $L = L + 1$.

(4) Store $\boldsymbol{X}_L$ on the stack of boxes for later processing.

(5) Set $L = L + 1$.

(i) If this is the controlling processor, return to Step II. If this is a working processor, then:

(1) If there are boxes assigned for processing still waiting on the stack, then go to Step III(a).

(2) If there are no boxes assigned for processing still on the stack, then remain inactive until reactivated by the controlling processor in Step II(b(3)).

IV. Final Point Solution.

(a-c) Same as serial algorithm Steps IV(a-c). Steps IV(b-c) can be parallelized if desired).

Note that in Step II(b(2)) there are L new boxes to assign for processing. $L-1$ of these boxes are given to the working processors in Step II(b(3)). The remaining box is processed by the controlling processor in Step II(b(4)), after which it returns in Step III(i) to Step II, where it may

generate additional parallel tasks for the working processors, thus activating as many processors as possible. If there are more boxes assigned for processing in Step II(b(2)) than there are inactive working processors, then the excess boxes assigned for processing simply remain on the stack and are processed as soon as other working processors become available [Step III(i(1))].

In Step III each processor must be able to change the values of its own variables without influencing the other processors, but all of the processors must access the same stack and stack-related variables. Additionally, the list of boxes $\mathscr{L}$ must be shared among the processors so that all of the boxes in the list are available for comparison at Steps III(b), III(e(2)), and III(g(2)). Thus, for Step III, each processor is provided with its own private copy of all of the variables and arrays except for the stack of boxes awaiting the root inclusion test and its associated pointers, the new box counter L, and the list $\mathscr{L}$ of solution-containing boxes, all of which are stored in global memory.

In terms of the speedup provided by parallel processing, the performance of the algorithm will depend on the structure of the binary tree generated from the bisections. In a worst case scenario the tree would be unbalanced and grow in depth only, as shown schematically in Figure 1. Such a tree would be produced if, after every bisection, one of the subboxes returned a test value of *true* or *false*, while the other subbox returned a value of *unknown*. In this case, the maximum speedup that could be obtained is no more than two, regardless of the number of processors. In a best case scenario, the tree would be balanced and grow in both breadth and depth, as shown schematically in Figure 2. Note, however, that here speedup will be limited by the startup time necessary to get all processors busy. For instance, for an eight-processor system, the fourth level in the binary tree would have to be reached before all processors could be kept busy. Both the difficulties mentioned here, startup time and lack of breadth in branching, are due to the binary nature of the tree. However, in principle there is no reason that a box that tests

unknown has to be subdivided into only two subboxes. Boxes could be subdivided into as many subboxes as there are available processors at any given time. An algorithm that does this would generate more impressive speedup statistics than the bisection-based algorithm used here. There are no guarantees that such an algorithm would solve a given problem any faster in terms of elapsed time; on the average, however, it might do so.

TEST PROBLEMS

In order to investigate the feasibility of using an interval Newton/generalized bisection algorithm on process simulation problems, we applied the algorithm described above to a number of test problems. Each problem is relatively small. Some have only one solution and some have multiple solutions. Each problem, and any variations of it considered, is described briefly below. Simple models for enthalpies and vapor-liquid phase equilibrium were used: the Sternling-Brown corresponding-states model for liquid heat capacities was used to estimate liquid phase enthalpies, ideal heat capacities were used to estimate vapor phase enthalpies, and K-values for vapor-liquid equilibrium were estimated using Raoult's law with the Antoine equation for vapor pressure. In the IN/GB algorithm, values of $\varepsilon = 10^{-5}$ and $\varepsilon_F = 10^{-10}$ were used. Except as noted below, the initial boxes used were wide enough to enclose any physically feasible solution.

Problem 1 -- Simple Ethylene Plant. This problem is adapted from a sample problem in Motard and Lee (1971). The flowsheet involves 10 units, 14 streams, and 7 hydrocarbon components. When modeled using SEQUEL-II a system of 163 equations must be solved.

Problem 2 -- Mixer/Divider Network. This problem is also adapted from a problem in Motard and Lee (1971). It involves 6 units, 10 streams, and 10 hydrocarbon components. When modeled using SEQUEL-II a system of 146 equations must be solved.

Problem 3 -- Multicomponent Flash. This problem is based on Example 8-14 in Reid *et al*. (1987). It involves a single isothermal flash unit separating ethane and n-heptane. Two formulations of this problem were considered. They differ in how the equilibrium relationships are written.

In the first formulation (*Problem 3A*), the equilibrium relationships for each component (i = 1,...,*C*) are written in terms of K-values and the exiting vapor and liquid flowrates V_i and L_i:

$$LV_i - K_iVL_i = 0 \qquad \text{(phase equilibrium)}$$

$$P_i^{SAT}(T) - K_iP = 0 \qquad \text{(explicit K-value evaluation)}$$

Here V and L are the total vapor and liquid flowrates. Of course, any other appropriate method for evaluating K-values could be used. This type of formulation is commonly used in modeling vapor-liquid equilibrium operations (e.g., Naphtali and Sandholm, 1971). When this formulation is used there may be as many as three solutions, one solution with no vapor phase ($V = V_i = 0$), one solution with no liquid phase ($L = L_i = 0$), and one with both phases. Only one of these solutions is physically correct. Other formulations of the flash problem also may have trivial solutions (Michelsen, 1993). For this particular problem there are three solutions, of which the two-phase result is the physically correct one. As modeled using SEQUEL-II, Problem 3A involves 21 equations.

There are other formulations of the flash problem, in which the two-phase result is implicitly assumed, that do not have the trivial roots. One such formulation (*Problem 3B*) is to introduce the vapor and liquid mole fractions y_i and x_i in place of K_i:

$$x_iP_i^{SAT}(T) - y_iP = 0 \qquad \text{(implicit K-value evaluation)}$$

$$x_iL - L_i = 0,\ i = 1,...,C-1 \qquad \text{(definition of } x_i\text{)}$$

$$y_iV - V_i = 0,\ i = 1,...,C-1 \qquad \text{(definition of } y_i\text{)}$$

$$\sum_{i=1}^{C} x_i = 1$$

$$\sum_{i=1}^{C} y_i = 1 \quad .$$

This formulation has one solution for the problem here. The SEQUEL-II model for Problem 3B has 23 variables.

Problem 4 -- Adiabatic CSTR. This problem is taken from Example 5-4 in Smith (1981). It has also been used by Kuno and Seader (1988). We consider two formulations of the problem. The simplest formulation (*Problem 4A*) is that given in Smith (1981) and Kuno and Seader (1988). It involves two balance equations and a feed temperature specification. Depending on the feed temperature specified, there may be more than one solution. We use the same three feed temperatures used by Kuno and Seader (1988). For T_f = 298 K (*Problem 4A-1*), there are three solutions. For T_f = 258 K (*Problem 4A-2*) there is one solution, and for T_f = 318 K (*Problem 4A-3*) there is also one solution.

For the second formulation (*Problem 4B*) a SEQUEL-II flowsheeting model was used. In this case there are 11 equations. There are additional equations and variables in this formulation because stream variables such as component flowrates, total flowrate, and enthalpy appear as independent variables in a flowsheeting system. The same three feed temperatures are also used in connection with this formulation, resulting in *Problems 4B-1*, *4B-2* and *4B-3*.

Problem 5 -- Flash with Recycle. This problem involves a flash unit whose vapor product is condensed, then split into a product stream and a stream recycled to the feed. The problem involves four units, seven streams, and two components. The SEQUEL-II model has 50 equations. Two variations of this problem are considered, based on the initial box used. In *Problem 5-1*, as in all the previous problems, the initial boxes specified were wide enough to contain any physically attainable solution. In *Problem 5-2*, a more intelligently chosen initial box

was used, based on knowledge of design specifications and physical properties.

Problem 6 -- Ammonia Plant. This problem is adapted from a problem in Seader *et al.* (1977). The flowsheet involves 11 units, 15 streams, and 5 components. When modeled using SEQUEL-II a system of 177 equations must be solved.

RESULTS AND DISCUSSION

The problems described above were solved using a BBN TC2000 multiple-processor computer. Both sequential runs on one processor, and parallel runs on up to 32 processors were performed. The algorithm was implemented using the Uniform System programming model. This was used because it provides memory and process management tools that efficiently map the algorithm onto the hardware configuration of the BBN TC2000.

Sequential Results

Detailed results for the sequential runs are shown in Table 1. On all problems except Problem 5, which will be discussed further below, the correct number of roots was found. To put the CPU times reported for the BBN TC2000 into rough perspective, we note first that some sequential runs were repeated on a Sun 3/50 workstation, which proved to be an order of magnitude slower than the TC2000. Second we note that, based on the well-known LINPACK benchmark (Dongarra, 1993), a machine (e.g., HP 9000/735) currently regarded as a fast workstation runs about 500 times faster than a Sun 3/50. From this one might infer that the times reported in Table 1 are at least an order of magnitude slower than what might be obtained on a fast (by current standards) workstation. Figures are also given for the lowest level reached in the binary tree generated in the bisection process, as well as for the maximum stack depth reached.

We first note that while the algorithm performs quite well on all the relatively small problems (3A, 3B, 4A, and 4B), its performance on the larger problems is inconsistent. On Problems 1 and 2, very few root inclusion tests were required, while on Problems 5 and 6 many more were required. As such, the tests run on this limited sample of problems seem to bear out the remarks of Kearfott (1989) and Kearfott and Novoa (1990) that performance on large problems is fairly unpredictable and that a larger number of equations does not necessarily imply a larger amount of work. It is interesting to note, however, that Problems 1 and 2 here are both relatively linear, involving no phase equilibrium calculations, while Problems 5 and 6, which do involve phase equilibrium, are more highly nonlinear.

Looking at Problem 5-1, the flash with recycle and with arbitrary initial box, we see an illustration of the only mode of failure for a properly implemented interval Newton/generalized bisection algorithm, namely an excessive computational requirement. On this problem, after one hundred thousand root inclusion tests and over six hours of CPU time, no solution was found, though we know there is at least one. On the other hand, when some effort was made to use a reasonable initial box, as in Problem 5-2, the problem could be solved immediately without bisection. The key variables here, for which better initial bounds lead to better algorithm performance, were the temperature variables. When information about critical temperatures and normal boiling points were used to intelligently bound the temperature variables, as in Problem 5-2, a solution was readily found. However, since the initial box used does not cover the entire feasible space, we cannot say with absolute certainty that this problem has only one solution. In fact, multiple solutions have been obtained for similar problems (e.g., Jacobsen and Skogestad, 1991). One thing that these results suggest is that the mathematical criterion used to select the variables to be bisected may be inadequate for process simulation problems. For instance, in Problem 5-1, if the temperature variables were selected, on physical grounds, to be preferentially

bisected, better performance might have been obtained.

Our experience with Problem 6 was similar to that with Problem 5. Using an arbitrary initial box the computational requirement was excessive. However, using a more intelligently chosen initial box, the computation time was reasonable, though for this problem, still fairly large, as shown in Table 1. While in principle the method described here requires no initial guess in the usual sense, the results for Problems 5 and 6 indicate that, at least for larger problems, some reasonable care needs to be taken to specify an initial box.

Parallel Results

The parallel implementation of the algorithm was used on the five problems with the largest computational times on the sequential runs. The speedup obtained using various numbers of processors is shown in Figure 3 for these five problems. Other results are the same as in Table 1, except for maximum stack depth, which differs because the stack is now processed in parallel. A limit of one hundred thousand root inclusion tests was maintained, so in Problem 5-1 the run still terminated without finding a solution. It should also be noted that multiple processors performing simultaneous root inclusion tests may handle the stack differently for every run, even when the same problem is retested using the same number of processors. This is due to small differences between individual processors and in interprocessor communication requirements. Thus CPU times are not exactly repeatable from run to run of the same problem. Since some points in Figure 3 are based on a single run rather than an average over several runs, they do not all fall on monotonically increasing curves.

For Problem 2, little or no speedup is observed. This is due in part to the small number of root inclusion tests required, and a binary tree that branches according to the worst-case scenario (Figure 1), which can be seen from the fact that level seven in the binary tree was

reached in seven root inclusion tests (Table 1). With this branching pattern, even under ideal circumstances speedup will only approach two. In this case, because of the small number of root inclusion tests, the overhead associated with activating multiple processors offsets any gain due to parallel root inclusion tests.

For the other problems, the speedup observed is valuable though far from ideal. For problem 5-1, on which processor utilization was most efficient, the fraction of operations executing concurrently is 86%, based on an Amdahl's law estimate. Some factors limiting performance are inherent in the algorithm. As discussed above, because of the binary nature of the search tree, some processors will necessarily be inactive initially, and perhaps subsequently as well when there is not sufficient breadth in the tree. Also, since the time required to complete a root inclusion test may vary substantially, situations may arise in which the controlling processor is working on a relatively long root inclusion test, while other processors finish shorter root inclusion tests and then have to wait for the controlling processor to finish working before it returns to check the stack and reactivate other processors. These results suggest that the current algorithm is best suited to a relatively small number of processors.

The algorithm used here exploits a large-grained parallelism at the level of the root inclusion test. It should be noted that there are opportunities for smaller-grained parallelism within the root inclusion tests. For example, Kearfott *et al.* (1991b) suggest using a Jacobi-like technique in place of Eqs. (5a-b). In this case, all components of $\boldsymbol{N}^{(k)}$ can be computed independently and in parallel before the intersections are performed, which also can be done in parallel.

CONCLUDING REMARKS

One way of looking at the interval Newton/generalized bisection algorithm is as a technique for identifying, from within specified initial bounds, an initial guess from which Newton's method is guaranteed to converge. Based on the group of problems considered here, it appears that for relatively small problems, it is feasible to specify initial bounds covering the entire physically attainable space. For larger problems, however, providing some reasonable initial bounds may be necessary on some problems. The premium over Newton's method alone can be judged by the number of root inclusion tests required. For small problems, even if a large number of root inclusion tests is needed, the price to pay may be quite reasonable, since each root inclusion test is inexpensive. For large problems, where root inclusion tests are more expensive, the cost of a large number of root inclusion tests may become unreasonable.

For some problems, the most powerful aspect of the IN/GB approach is its ability to guarantee that any and all roots within the specified initial box will be found. For problems with multiple roots, various homotopy-based methods have been successfully applied. However, unlike IN/GB, these methods do not provide a rigorous guarantee that all roots have been found, except in special cases. Both IN/GB and homotopy-based methods can be computationally expensive. The experiments of Kearfott (1987b) indicate that their relative expensive will depend on the nature of the problem. Any additional expense incurred in using IN/GB may be a worthwhile premium to pay to insure that all solutions to a problem are found.

Finally, it should be noted that since even the largest problem considered here is relatively small by process simulation standards, we cannot judge the ultimate value of IN/GB in larger-scale process simulation. Based on its performance on small problems, IN/GB might be especially useful, however, in developing robust procedures or modules for use within larger process simulation programs. Some additional results on the use of IN/GB for small nonlinear

equation solving problems related to process simulation, and with multiple roots, have been presented recently by Stadtherr *et al.* (1994) and Balaji and Seader (1994).

Acknowledgments. This work was funded in part by the National Science Foundation under grants DDM-9024946 and DMI-9322682. The authors thank Professor R. Baker Kearfott for many helpful discussions concerning INTBIS and interval analysis. The authors also thank the Mathematics and Computer Science Division of Argonne National Laboratory for providing time on the BBN TC2000, and Halliburton Energy Services for encouraging publication of this work.

REFERENCES

Alefeld, G. and J. Herzberger, *Introduction to Interval Computations*. Academic Press, New York, NY (1983).

Balaji, G. V. and J. D. Seader, Application of interval-Newton method to chemical engineering problems. Presented at Fourth International Symposium on Foundations of Computer Aided Process Design (FOCAPD'94), Snowmass Village, Colorado, July 10-15, 1994.

Bullard, L. G. and L. T. Biegler, Iterative linear programming strategies for constrained simulation. *Comput. Chem. Eng.*, **15**, 239-254 (1991).

Chen, H. S. and M. A. Stadtherr, A modification of Powell's dogleg method for solving systems of nonlinear equations. *Comput. Chem. Eng.*, **5**, 143-150 (1981).

Chen, H. S. and M. A. Stadtherr, On solving large sparse nonlinear equation systems. *Comput. Chem. Eng.*, **8**, 1-7 (1984).

Dongarra, J. J., Performance of various computers using standard linear equations software. Report CS-89-85, Computer Science Dept., University of Tennessee, Knoxville, TN (1993).

Hansen, E. R., On solving systems of linear equations using interval analysis. *Math Comput*., **22**, 374-384 (1968).

Hansen, E. R., *Global Optimization Using Interval Analysis*. Marcel Dekkar, New York, NY (1992).

Hansen, E. R. and R. I. Greenburg, An interval Newton method. *Appl. Math. Comput*., **12**, 89-98 (1983).

Hansen, E. R. and S. Sengupta, Bounding solutions of systems of equations using interval analysis. *BIT*, **21**, 203-211 (1981).

Jacobsen, E. W. and S. Skogestad, Multiple steady states in ideal two-product distillation. *AIChE J.*, **37**, 499-511 (1991).

Kaijaluoto, S., P. Neittaanmäki, and J. Ruhtila, Comparison of different solution algorithms for sparse linear equations arising from flowsheeting problems. *Comput. Chem. Eng.*, **13**, 433-439 (1989).

Kearfott, R. B., Abstract generalized bisection and a cost bound. *Math. Comput.*, **49**, 187-202 (1987a).

Kearfott, R. B., Some tests of generalized bisection. *ACM Trans. Math. Softw.*, **13**, 197-220 (1987b).

Kearfott, R. B., Interval arithmetic methods for nonlinear systems and nonlinear optimization: An outline and status. In *Impacts of Recent Computer Advances on Operations Research*, eds. R. Sharda, B. L. Golden, E. Wasil, O. Balci, and W. Stewart, Elsevier, New York, NY (1989).

Kearfott, R. B., Interval arithmetic techniques in the computational solution of nonlinear systems of equations: Introduction, examples, and comparisons. *Lectures in Applied Mathematics*, **26**, 337-357 (1990a).

Kearfott, R. B., Preconditioners for the interval Gauss-Seidel method. *SIAM J. Numer. Anal.*, **27**, 804-822 (1990b).

Kearfott, R. B. and M. Novoa III, INTBIS, a portable interval Newton/bisection package. *ACM Trans. Math. Softw.*, **16**, 152-157 (1990).

Kearfott, R. B, C. Hu, and M. Novoa III, A review of preconditioners for the interval Gauss-Seidel method. *Interval Computations*, **1**, 59-85 (1991a).

Kearfott, R. B., M. Bayoumi, Q. Yang, and C. Hu, The all-row preconditioned interval Newton method on an MIMD machine. Presented at Second International Conference on Industrial and Applied Mathematics, Washington, D. C. (1991b).

Krawczyk, R., Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, **4**, 187-201 (1969).

Kuno, M. and J. D. Seader, Computing all real solutions to systems of nonlinear equations with a global fixed-point homotopy. *Ind. Eng. Chem. Res.*, **27**, 1320-1329 (1988).

Michelsen, M. L., Phase equilibrium calculations. What is easy and what is difficult? *Comput. Chem. Eng.*, **17**, 431-439 (1993).

Moore, R. E., *Interval Analysis*. Prentice-Hall, Engelwood Cliffs, NJ (1966).

Motard, R. L. and H. M. Lee, *CHESS: Chemical Engineering Simulation System, User's Guide*, 3rd edition. University of Houston, Houston, TX (1971).

Naphtali, L. M. and D. P. Sandholm, Multicomponent separation calculations by linearization. *AIChE J.*, **17**, 148 (1971).

Neumaier, A., *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge (1990).

Powell, M. J. D., A hybrid method for nonlinear equations. In *Numerical Methods for Nonlinear Algebraic Equations*, ed. P. Rabinowitz, Gordon and Breach, London (1970).

Reid, R. C., J. M. Prausnitz, and B. E. Poling, *The Properties of Gases and Liquids*, 4th edition. McGraw-Hill, New York, NY (1987).

Schnepper, C. A. and M. A. Stadtherr, On using parallel processing techniques in chemical process design. Presented at AIChE Annual Meeting, Chicago, November 11-16, 1990.

Schnepper, C. A., *Large Grained Parallelism in Equation-Based Flowsheeting Using Interval Newton/Generalized Bisection Techniques*. Ph.D. Dissertation, University of Illinois, Urbana, IL (1992).

Seader, J. D., W. D. Seider, and A. C. Pauls, *FLOWTRAN Simulation: An Introduction*, 2nd edition. CACHE, Cambridge, MA, (1977).

Smith, J. M., *Chemical Engineering Kinetics*, 3rd edition. McGraw-Hill, New York, NY (1981).

Stadtherr, M. A. and E. S. Wood, Sparse matrix methods for equation-based chemical process flowsheeting - II: Numerical phase. *Comput. Chem. Eng.*, **8**, 19-33 (1984).

Stadtherr, M. A., C. A. Schnepper, and J. F. Brennecke, Robust phase stability analysis using interval methods. Presented at Fourth International Symposium on Foundations of Computer Aided Process Design (FOCAPD'94), Snowmass Village, Colorado, July 10-15, 1994.

Swaney, R. E. and C. E. Wilhelm, Robust solution of flowsheeting equation systems. In *Proc. Third International Conference on Foundations of Computer-Aided Process Design*, eds., J. J. Siirola, I. E. Grossman, and G. Stephanopoulos, Elsevier, New York (1990).

Wayburn, T. L. and J. D. Seader, Homotopy continuation methods for computer-aided process design. *Comput. Chem. Eng.*, **11**, 7-25 (1987).

Zitney, S. E. and M. A. Stadtherr, Computational experiments in equation-based chemical process flowsheeting. *Comput. Chem. Eng.*, **12**, 1171-1186 (1988).

Table 1. Results for sequential runs on one processor of a BBN TC2000. CPU time is in seconds. These times are estimated to be over an order of magnitude slower than what might be expected on a fast (by current standards) workstation (see text).

Problem	Number of Equations	Number of Roots Found	Number of Root Inclusion Tests	Lowest Level Reached in Binary Tree	Maximum Stack Depth	CPU Time
1	163	1	1	1	1	3.1
2	146	1	7	7	6	21.8
3A	21	3	569	27	21	49.0
3B	23	1	4	3	2	1.2
4A-1	3	3	47	10	7	0.98
4A-2	3	1	40	10	5	0.62
4A-3	3	1	41	9	6	0.70
4B-1	11	3	135	15	9	12.94
4B-2	11	1	109	15	8	8.41
4B-3	11	1	83	12	8	7.24
5-1	50	0†	100001	81	38	21790
5-2	50	1	1	1	1	4.0
6	177	1	108	16	14	1003.5

†Program terminated when limit of 100000 root inclusion tests was exceeded. This problem has at least one solution.

FIGURE CAPTIONS

Figure 1. Unbalanced tree growing in depth only.

Figure 2. Balanced tree growing in both breadth and depth.

Figure 3. Speedup obtained using multiple processors on a BBN TC2000.

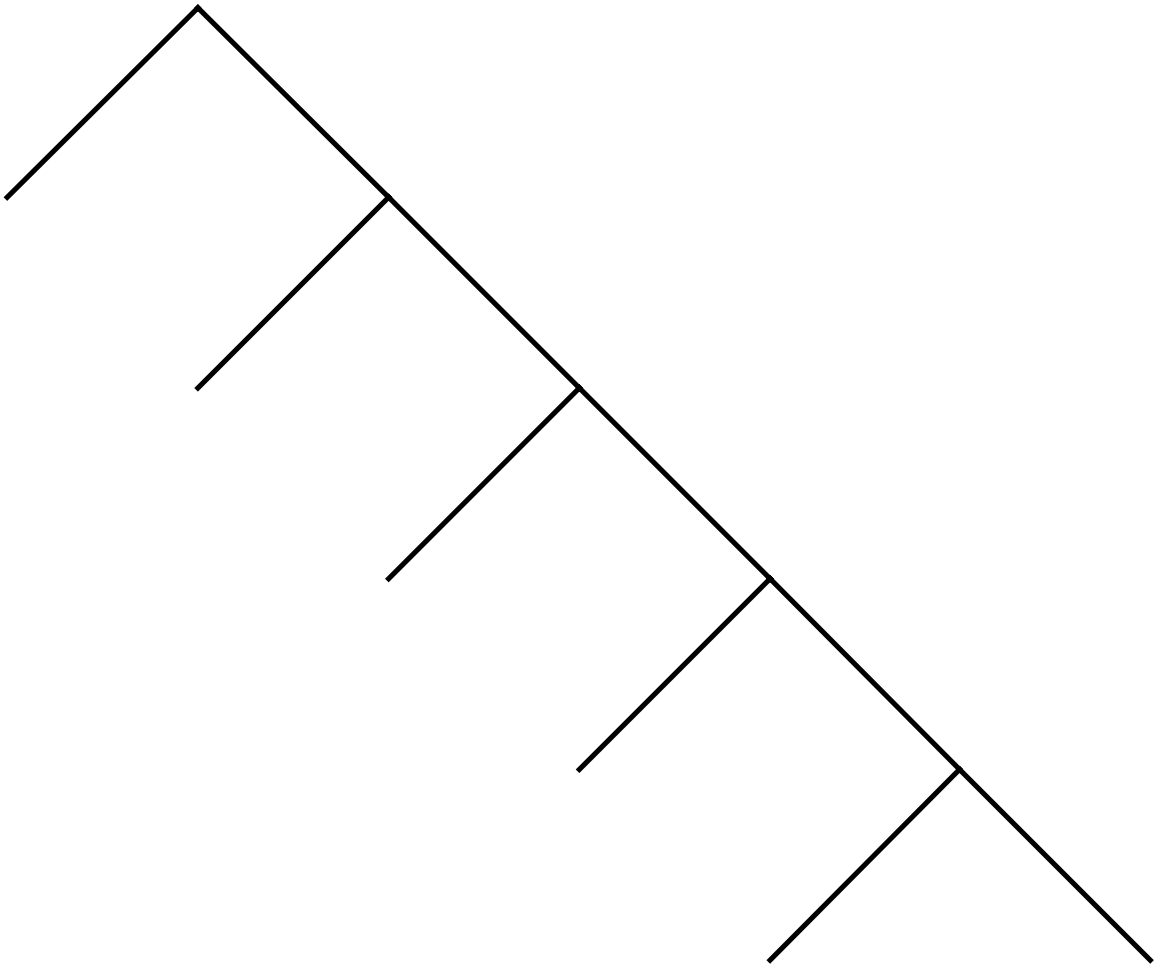
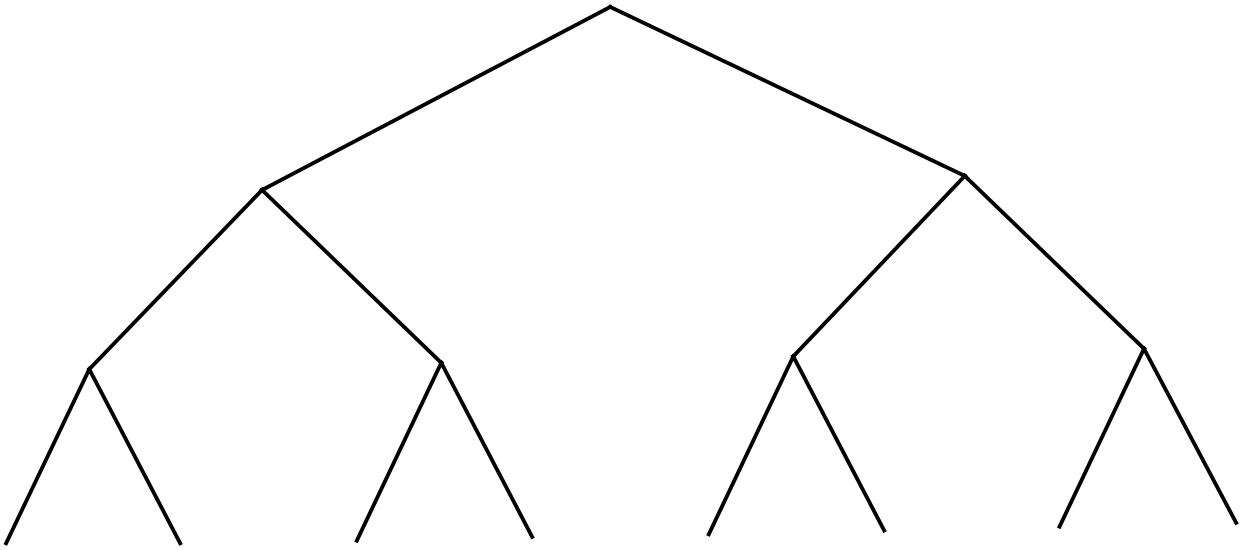


Figure 1



(Figure 2)

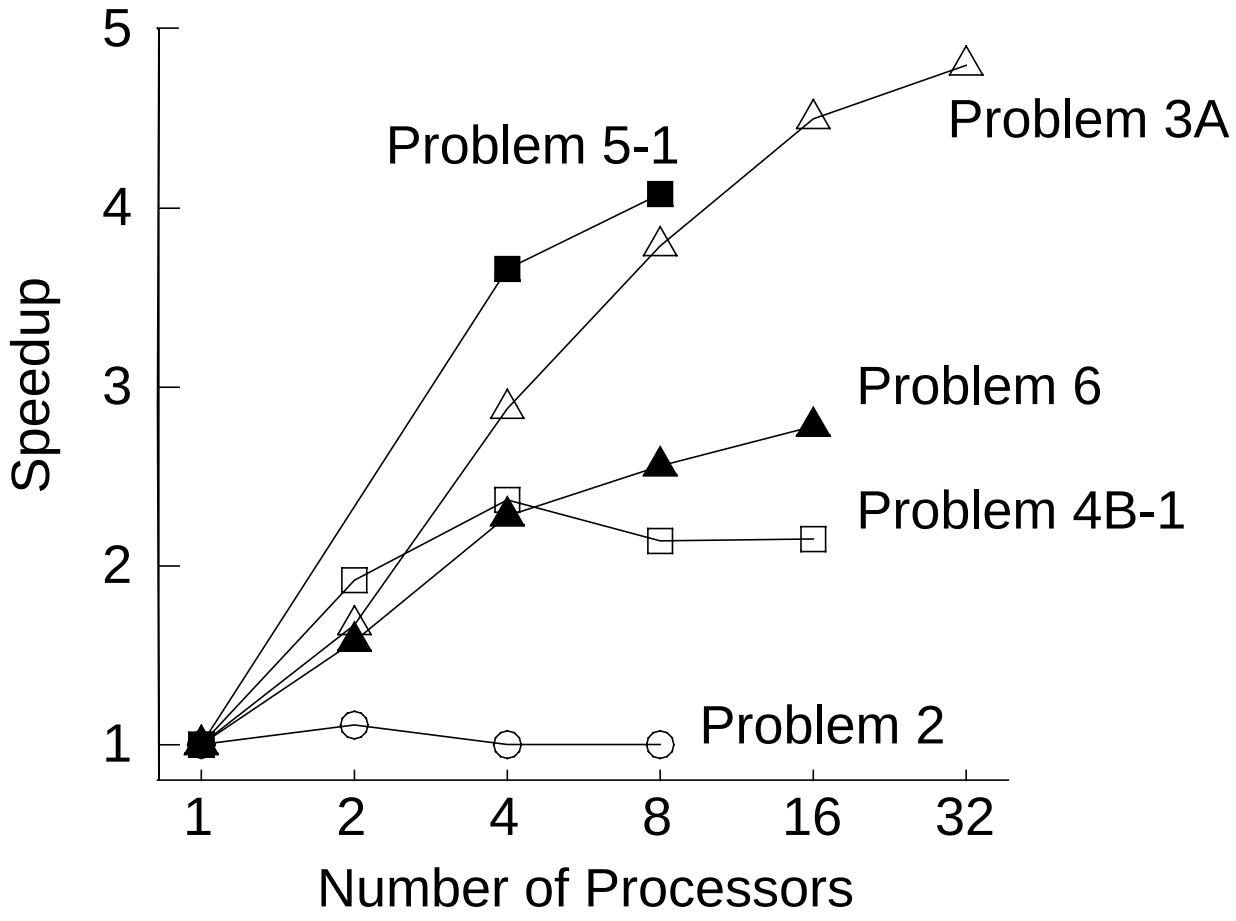

Speedup
5
4
3
2
1
Problem 5-1
Problem 3A
Problem 6
Problem 4B-1
Problem 2
1
2
4
8
16
32
Number of Processors

Figure 3