On Considering an Interval Constraint Solving Algorithm as a Free-Steering Nonlinear Gauss-Seidel Procedure

Frédéric Goualard
Laboratoire d'Informatique de Nantes-Atlantique, FRE CNRS 2729
2, rue de la Houssinière. B.P. 92208
F-44322 NANTES CEDEX 03
frederic.goualard@lina.univ-nantes.fr

ABSTRACT

We show that a classical interval constraint propagation algorithm enforcing box consistency may be interpreted as a free-steering nonlinear Gauss-Seidel procedure. This suggests that the choice of a transversal in the incidence matrix associated with the problem to solve is paramount to the efficiency of the algorithm. We present experimental evidences that it is indeed so, and we suggest an heuristics to compute good transversals. The improved interval constraint algorithm is compared with a classical one and with standard methods such as Hansen-Sengupta on some well-known benchmarks.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*; G.1.0 [**Numerical Analysis**]: General—*Interval arithmetic*

General Terms

Algorithms, Reliability

Keywords

nonlinear system, branch-and-prune method, constraint

1. INTRODUCTION

We consider the problem of finding all the solutions of systems of the form:

$$\begin{aligned} f_1(x_1,\dots,x_n) &= 0 \\ &\ddots \\ f_n(x_1,\dots,x_n) &= 0 \end{aligned} \tag{1}$$

where $f_1,\dots,f_n$ are nonlinear real functions[1].

[1]In the rest of the paper, we also allow for systems where some variables do not occur in all f_is.

A successful approach to this problem combines a *branch-and-prune algorithm* (See Table 1) with interval extensions [11] of classical numerical methods such as the preconditioned Newton-Gauss-Seidel [12] (aka Hansen-Sengupta [6]) method.

However, with large systems and large domains, algorithms that rely on matrix computation and linear approximations such as the Hansen-Sengupta method become computationally expensive and inefficient. As an alternative, Herbort and Ratz proposed a componentwise Newton method [7] that considers each equation $f_i(x_1,\dots,x_n) = 0$ separately, using a unidimensional Newton iteration on a unary projection of f_i onto one of the variables $x_1,\dots,x_n$.

Herbort and Ratz's method avoids the manipulation of matrices. On the other hand, it may create a large search tree whenever the projections of the f_is have a lot of zeroes that are not solutions of the system.

To overcome this, some *interval constraint methods* do not attempt to directly isolate the zeroes of the projections of each f_i. Rather, they try for each projection to shrink the domain of the variable projected onto, so as to "tightly" enclose all the zeroes of the corresponding unidimensional function; The meaning of "tightly" is precisely defined by the local consistency notion considered [10]. Experimental evidences show that these methods usually outperform both Hansen-Sengupta and Herbort-Ratz methods.

In this paper, we analyze algorithms enforcing a particular local consistency called *box consistency* [2], and we show that they correspond to a *free-steering nonlinear Gauss-Seidel procedure* [12]. A consequence is that the choice of the variable x_j to project the function f_i onto is paramount to the efficiency of the overall process. We then propose an heuristics to make that choice and show experimentally that it allows for a speed-up of several orders of magnitude for some large constraint systems.

2. SOLVING NONLINEAR SYSTEMS

Given a constraint system C of the form (1), let $\rho \subseteq \mathbb{R}^n$ be the solution set of C. Let us assume the availability of a prune function that takes as input the vector F of f_is occurring in C and a Cartesian product of n interval domains (*box*) $\boldsymbol{D}$ for the variables, and that returns as output a box $\boldsymbol{D'}$ verifying the two properties $\boldsymbol{D'} \subseteq \boldsymbol{D}$ and $\rho \cap \boldsymbol{D'} = \rho \cap \boldsymbol{D}$. Then, Alg. BaP in Table 1 computes a set *sol* of boxes whose largest dimension has a width smaller than some user-given real parameter ε (see the paragraph below for the notations used). Each box in *sol* is included into the box $\boldsymbol{D}_{in}$, and the union of the boxes in *sol* contains $\rho \cap \boldsymbol{D}_{in}$.

Notations: Given a finite set $\mathbb{F}$ of floating-point numbers, let $\mathbb{I}$ be the set of intervals with bounds in $\mathbb{F}$. For any $a \in \mathbb{F}$, let a^+ (resp. a^-) be the smallest element of $\mathbb{F}$ strictly greater than a (resp. greatest element of $\mathbb{F}$ strictly smaller than a). Let $\mathscr{P}(\mathbb{I})$ be the *power set* of $\mathbb{I}$. Given an interval $\boldsymbol{I} = [a, b]$, we denote $\underline{\boldsymbol{I}} = a$ and $\overline{\boldsymbol{I}} = b$; let $\mathsf{w}(\boldsymbol{I}) = \updownarrow (b-a) \updownarrow$ be the *width* of $\boldsymbol{I}$ (where $\updownarrow \gamma \updownarrow$ is the floating-point number closest to the real γ, with the IEEE 754 rule for ties); the width of a box is the width of its largest projection. Let $\mathsf{dist}(\boldsymbol{I_1}, \boldsymbol{I_2})$ be the Hausdorff distance between $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$. Given a real set R, let $\Box\, R$ be the smallest (w.r.t. inclusion) interval containing R.

```
BaP(in $F = (f_1, \dots, f_n)\colon \mathbb{R}^n \to \mathbb{R}^n$; in $\boldsymbol{D}_{in} \in \mathbb{I}^n$;
     out $sol \in \mathscr{P}(\mathbb{I}^n)$ )
begin
  $boxset \leftarrow \{\boldsymbol{D}_{in}\}$ ; $sol \leftarrow \varnothing$
  while $boxset \neq \varnothing$ do
    $\boldsymbol{D} \leftarrow$ extract_box($boxset$)
    $\boldsymbol{D} \leftarrow$ prune($F, \boldsymbol{D}$)
    if w($\boldsymbol{D}$) $\leqslant \varepsilon$ then
      if $\boldsymbol{D} \neq \varnothing$ then
        $sol \leftarrow sol \cup \{\boldsymbol{D}\}$
      endif
    else
      $boxset \leftarrow boxset \cup$ split($\boldsymbol{D}$)
    endif
  endwhile
end
```

Table 1: Branch-and-Prune algorithm

For the special case where all the f_is are linear, the prune function may be implemented by the Gauss-Seidel method. For a nonlinear system, a classical choice is Hansen-Sengupta [6], which iteratively solves with the Gauss-Seidel method the preconditioned linear system arising from the linearization of System (1) induced by a multidimensional Newton step.

The principle at the root of the Gauss-Seidel method may also be used for nonlinear constraints leading to *nonlinear Gauss-Seidel* [12] (NLGS): if it is possible to *normalize* each constraint $f_i(x_1, \dots, x_n) = 0$ into $x_i = g_i(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$, NLGS is similar to the linear version (and indeed falls back to the Gauss-Seidel method for linear f_is).[2] Otherwise, given a box $\boldsymbol{D} = \boldsymbol{I_1} \times \cdots \times \boldsymbol{I_n}$ and $\boldsymbol{f_1}, \dots, \boldsymbol{f_n}$ interval extensions of $f_1, \dots, f_n$, one considers the n unary *projection constraints*:

$$\begin{aligned} \boldsymbol{f_1}^{(1)}(x_1, \boldsymbol{I_2}, \dots, \boldsymbol{I_n}) &= 0 \\ &\ddots \\ \boldsymbol{f_n}^{(n)}(\boldsymbol{I_1}, \dots, \boldsymbol{I_{n-1}}, x_n) &= 0 \end{aligned} \tag{2}$$

and uses any unidimensional root-finding method to tighten the domain of each variable x_i in turn. Using the unidimensional Newton method leads to the *Gauss-Seidel-Newton method* [12], whose extension to intervals is the Herbort-Ratz method [7]. Let HR be the elementary step performed by one unidimensional Newton step applied to a projection $\boldsymbol{f_i}^{(j)}$, where i and j may be different. As soon as $\boldsymbol{D}$ is moderately large, it is very likely that each projection constraint will have many "solutions" that are not solutions of the original real system, and whose discarding slows down the computation. Another problem lies in that the Newton method will usually fail to narrow down the domain of some x_i if there is more than one solution to the corresponding projection constraint for the current box $\boldsymbol{D}$, thereby demanding more splittings in the branch-and-prune algorithm. Achieving the right balance between the amount of work required by the prune method and the number of splittings performed overall is the key to the maximum efficiency of BaP. In this very situation, experimental evidences show that trying harder to narrow down the domain of x_i pays off. A way to do it is to ensure that the canonical intervals $[\underline{\boldsymbol{I_i}}, \underline{\boldsymbol{I_i}}^+]$ and $[\overline{\boldsymbol{I_i}}^-, \overline{\boldsymbol{I_i}}]$ are solutions of $\boldsymbol{f_i}^{(i)}(\boldsymbol{I_1}, \dots, \boldsymbol{I_{i-1}}, x_i, \boldsymbol{I_{i+1}}, \dots, \boldsymbol{I_n}) = 0$. Let BC be an algorithm that ensures this property (called *box consistency [2] of x_i w.r.t. the constraint $\boldsymbol{f_i} = 0$ and $\boldsymbol{D}$*) for a projection $\boldsymbol{f_i}^{(j)}$, where i may be different from j. A simple method to implement it uses a dichotomic process to isolate the leftmost and rightmost solutions included in $\boldsymbol{D}$ of each projection constraint (Refer to the original paper [2] for a more detailed description).

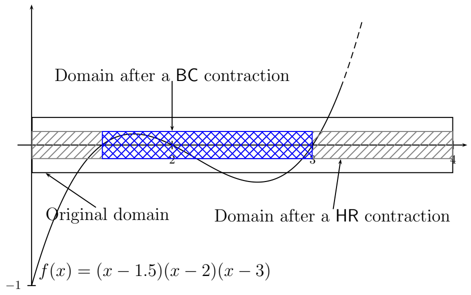
Domain after a BC contraction
2
3
4
Original domain
Domain after a HR contraction
$f(x) = (x-1.5)(x-2)(x-3)$
-1

Figure 1: Comparison of HR and BC

EXAMPLE 1. *Consider the constraint $f(x) = (x-1.5)(x-2)(x-3) = 0$ and the domain $\boldsymbol{I} = [1, 4]$ for x (See Fig. 1). The* HR *method leaves $\boldsymbol{I}$ unchanged because the derivative of f over the initial domain contains* 0*. Alg.* BC *narrows down $\boldsymbol{I}$ to $\boldsymbol{I'} = [1.5, 3]$, which is the smallest interval included in $\boldsymbol{I}$ that contains all the solutions to the interval constraint $\boldsymbol{f}(\boldsymbol{x}) = 0$.*

An implementation of NLGS loops over $\boldsymbol{f_1}^{(1)}, \dots, \boldsymbol{f_n}^{(n)}$ *in order* (primary iteration), applying either HR or BC on each projection (secondary iteration) until reaching some fixed-point. *Free-steering nonlinear Gauss-Seidel* [12] allows to consider the n projection constraints in an arbitrary order, which may even change from one sweep of the primary iteration to the next. Yet another variant makes of each secondary iteration a complete NLGS step [12, p. 224]. It is then possible to consider some projection constraints more often than others.

In the linear case, it is well-known that the efficiency of the Gauss-Seidel method may depend heavily on the ordering of the f_is, that is the choice of a *transversal* in the coefficients matrix. The same holds true for nonlinear Gauss-Seidel methods and the choice of the pairs (f_i, x_j) to consider for generating the n projection constraints appears a key factor in their efficiency [7, 1].

The *Generalized Gauss-Seidel* algorithm we introduce in Table 2 is a framework for defining any free-steering nonlinear Gauss-Seidel-like method. The select function chooses

[2]An interesting point to raise here is that, using *relational interval arithmetic*, it is always possible to implicitly normalize a constraint [3].

the pairs (f_i, v_j) to consider in the current primary iteration, while the tighten function implements the secondary iteration narrowing down the domain of v_j w.r.t. the projection constraint $f_i^{(j)}$. The parameter Δ allows the user to decide whether the last primary iteration was effective or not. If not, we prefer reverting to the splitting process, which is less expensive than floundering with GGS.

```
GGS(in $F = (f_1, \ldots, f_n)\colon \mathbb{R}^n \to \mathbb{R}^n$;
    inout $\boldsymbol{D} = \boldsymbol{I_1} \times \cdots \times \boldsymbol{I_n} \in \mathbb{I}^n$)
begin
  $modified \leftarrow$ true; $\boldsymbol{D'} \leftarrow [-\infty, +\infty]^n$
  while $\mathrm{w}(\boldsymbol{D}) > \varepsilon$ and $modified$ do
    $Lfv \leftarrow \mathrm{select}(\{f_1, \ldots, f_n\}, \{x_1, \ldots, x_n\}, \boldsymbol{D'}, \boldsymbol{D})$
    $\boldsymbol{D'} \leftarrow \boldsymbol{D}$
    foreach $(f_i, v_j)$ in $Lfv$ do
      $\boldsymbol{I_j} \leftarrow \boldsymbol{I_j} \cap \mathrm{tighten}(f_i, v_j, \boldsymbol{D})$
    endfor
    $modified \leftarrow (\mathrm{dist}(\boldsymbol{D}, \boldsymbol{D'}) > \Delta)$
  endwhile
end
```

Table 2: Generalized Gauss-Seidel algorithm

In the following, the name of an instantiation of GGS is made of the names of the select and tighten functions chosen.

Let GSSN (resp. GSSN2) be a select function that always creates the same list of n projections (resp. all the—at most n^2—projections) in the same order. For linear f_is, a GSSN function always returning the list $(f_1, x_1), (f_2, x_2), \ldots, (f_n, x_n)$, and a tighten function applying the interval Gauss-Seidel inner step, GGS falls back to the usual Gauss-Seidel method. The Gauss-Seidel-Newton method may be derived accordingly.

For constraint systems such that some variables do not occur in all the constraints, it becomes interesting to use a select function that puts into *Lfv* only the pairs (f_i, v_j) that are such that $f_i^{(j)}$ contains at least one input variable whose domain has been modified during the previous primary iteration (otherwise, it is useless to reconsider this projection constraint). Let AC3N (resp. AC3N2) be such a select function that considers n projections (resp. all the—at most n^2—projections).

If the tighten function uses a BC algorithm, and we choose an AC3N2 select function, we obtain the classical BC3 algorithm (AC3N2.BC in our parlance) used to enforce *box consistency* [2] on nonlinear constraint systems, short of the precise order for the reinvocation of projection constraints.

The *incidence matrix* for a constraint system is a 0/1 matrix: the entry on Line i and Column j contains a 1 if x_j occurs in f_i and 0 otherwise. In the following, we impose on GSSN and AC3N functions the limitations that each constraint shall be used only once, and that each variable shall be used only once as an output variable (This corresponds in effect to the choosing of a *transversal* in the incidence matrix).

3. ANALYZING BC3-LIKE ALGORITHMS

For the record, all times presented in the rest of the paper have been measured on a computer whose score with the double precision *Whetstone test* [4] (Nov. 1997 version) is 475 MWIPS. The code used to implement all the techniques described here was written entirely in C++ by the author. All the boxes computed have a width smaller than $\varepsilon = 10^{-8}$.

Figures 2 and 3 present computational times obtained when solving Barton's problem [1] with either GSSN2.BC (that is, the BC3 algorithm except that the projections are considered in a fixed order) or GSSN.BC:

$$\begin{array}{l|l} f_1(X) = x_1 + x_4 - 10 = 0 & f_4(X) = x_4 - 3x_1 + 6 = 0 \\ f_2(X) = x_2^2 x_3 x_4 - x_5 - 6 = 0 & f_5(X) = x_1 x_3 - x_5 + 6 = 0 \\ f_3(X) = x_1 x_2^{1.7}(x_4 - 5) - 8 = 0 & \end{array}$$

for $X = (x_1, \ldots, x_5)$ and the initial box $\boldsymbol{D} = [-100, 100]^5$.

Figure 2 corresponds to the solving of Barton's problem with a nonlinear Gauss-Seidel algorithm whose secondary iteration enforces box consistency using both a dichotomic process and interval Newton steps. The order of the projection constraints is fixed at the start and remains the same for all the sweeps of the primary iteration. This figure synthesizes the influence of both the choice of a transversal, and of the order of consideration of the projections: one gray column encloses the computation times for all the transversals that contain the pair written under the abscissa. The position of a column of dots inside a gray column stands for the position of the corresponding projection among the others in the primary iteration. We have solved Barton's problem using all the possible transversals and all the possible positions of each projection in the primary iteration.

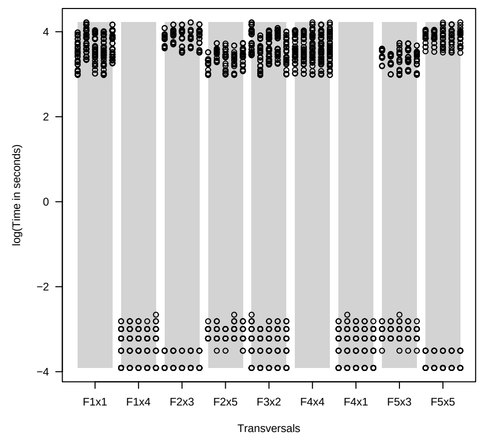

Figure 2: Barton with GSSN.BC

A first observation is that the position of each of the n projections in the primary iteration is not very relevant: for each of the nine gray columns, the results dispersion is almost the same among the five columns of dots corresponding to the five possible positions. On the other hand, the choice of the transversal greatly influences the computation time. In particular, we see that using (f_1, x_1) instead of (f_1, x_4) leads to much larger computation times whatever the choice of the other projections in the transversal. The same holds for (f_4, x_4) and (f_4, x_1). Actually, retaining (f_1, x_1) imposes the choice of (f_4, x_4) and accordingly for (f_1, x_4), if our selection of projections is to be a transversal.

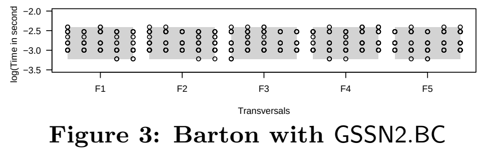

Figure 3: Barton with GSSN2.BC

Picture 3 reports the times obtained when solving Barton's problem with GSSN2.BC: we consider all possible projections (here, there are nine) in a fixed order; the secondary iteration enforces box consistency over each of them. The five leftmost columns correspond to solving the problem with all the projections from f_1 considered first (very first column), in “second position” (that is after all the projections of a single other constraint), and so on. The same holds for the other groups of five columns.

Once again, the respective positions of the projections in the primary iteration do not appear significant. In addition, since all the projections are used during the solving, we never encounter instances whose solving time are much larger than others. For the case of Barton’s problem, all times are not too far from the best possible time reported in the upper picture. By using all possible transversals, a BC3-like algorithm avoids making a bad choice. On the other hand, it prevents us from achieving the best performances since there are more projections to consider, which is time consuming.

4. CHOOSING A GOOD TRANSVERSAL

For large dense problems (that is, problems with dense incidence matrices), BC3-like algorithms can no longer be used satisfactorily: in the densest case, the number of projections grows as the square of the number of equations, leading to an overwhelming number of projections to consider. Very often, the additional cost their handling incurs largely counterbalances the potential benefit of avoiding the selection of a bad transversal.

To some extent, this situation has already been acknowledged by interval constraint researchers: in a more general settings, Lhomme *et al.* [9] noticed that during the propagation of domain variations (i.e., primary iteration), many applications of narrowing operators (that is, secondary iterations in our parlance) are unnecessary. They suggest a scheme to identify these useless operators by looking for static and dynamic dependencies; they then allocate the bulk of the computing power to but a subset of all the operators. Granvilliers [5] uses both a static selection of the operators relying on a simple heuristics based on the number of occurrences of each variable in the constraints, and a dynamic selection during which he considers all the possible projection constraints and retains only those that tighten variable domains the most.

In the remaining of this paper, we first present the results obtained with an heuristics that helps us select statically and/or dynamically a transversal (that is, the propagation algorithm will deal with only n operators). We then suggest another heuristics based on the structure of the constraints.

For the real linear case, it is well known that a sufficient condition for the Gauss-Seidel method to converge is to have a strictly diagonal dominant coefficient matrix. Hansen and Smith have shown in 1967 that a similar condition exists for the interval Gauss-Seidel method and they suggest to use a preconditioner whose effect is that a variable does not occur prominently in several constraints.

Our guess is that such a situation is beneficial to nonlinear Gauss-Seidel as well. We then propose the following method to achieve some kind of “diagonal dominance” in a nonlinear system. Given the current box $\boldsymbol{D}$, choose for each constraint $f_i(x_1,\dots,x_n)=0$, the variable on which f_i is the most dependent in the domain $\boldsymbol{D}$. This information may be obtained by computing the gradient of f_i on $\boldsymbol{D}$: select the variable x_j with maximal mignitude[3], that is, choose x_j such that:

$$\mathsf{mig}\,\frac{\partial f_i}{\partial x_j}(\boldsymbol{D}) = \max_{k=1\dots n} \mathsf{mig}\,\frac{\partial f_i}{\partial x_k}(\boldsymbol{D})$$

If no mignitude is non-null, then choose instead the variable x_l with maximal magnitude[4].

If different from zero, the largest mignitude corresponds to the variable for which f_i varies the most on $\boldsymbol{D}$. Otherwise, we choose the variable with largest magnitude because it is the one for which f_i *potentially* varies the most on $\boldsymbol{D}$. The use of this heuristics to obtain a transversal for the whole system (1) requires the computation of its Jacobian J. It may be done after each modification of the current box, each time we enter in the prune function, or even only once at the beginning of the solving with the initial box. The first possibility is clearly too expensive; the second is also expensive (of the order of the computational burden required by the Hansen-Sengupta method) but allows to optimize the computation of box consistency: if we find a variable x_j for f_i for which the mignitude of the partial derivative is strictly positive, we may replace BC by HR when considering $f_i^{(j)}$ since there is then at most one solution in the domain of x_j. We thus divide the amount of work performed by two.

Given a box $\boldsymbol{D}$, the computation of a transversal may be performed as follows: create the $n\times n$ real matrix W with entries defined as follows:

$$W_{ij} = \begin{cases} \mathsf{mig}\,\dfrac{\partial f_i}{\partial x_j}(\boldsymbol{D}) + \max\limits_{k,l} \mathsf{mag}\,\dfrac{\partial f_k}{\partial x_l}(\boldsymbol{D}), & \text{if } \mathsf{mig}\,\dfrac{\partial f_i}{\partial x_j}(\boldsymbol{D}) \neq 0 \\ \mathsf{mag}\,\dfrac{\partial f_i}{\partial x_j}(\boldsymbol{D}), & \text{otherwise} \end{cases}$$

Finding a transversal according to the heuristics above then corresponds to the computation of a perfect weighted matching in the bipartite graph associated to the matrix W, which is an $\mathcal{O}(n^2 \log n)$ process. In W, we add to a non-null mignitude the largest magnitude overall to ensure that a mignitude is always preferred over a magnitude.

To avoid the cost of the matching computation, one may lift the requirement of selecting only n projections as long as each variable and each function occurs in at least one pair selected. For our tests, we computed the transversal only once at the beginning of the solving. As experimental evidences show, this choice, though far from perfect, is often sufficient to ensure a significant speed-up.

Fig. 4 and 5 present the results obtained on two standard benchmarks: the Broyden-banded problem and the Moré-Cosnard problem[5]. They both are nonlinear and scalable at will. The matrices in both graphics are graphical depictions of their incidence matrices: a black square at position (i,j) means that x_j occurs in f_i; a white square is used otherwise. As its name suggests, the Broyden-banded is not a dense problem; on the other hand, Moré-Cosnard is a fully dense problem.

AC3N.BC uses our heuristics to compute a transversal once so that the propagation algorithm works with only n projections. AC3N2.BC works with all the possible projections, as expected.

[3] $\mathsf{mig}\,\boldsymbol{I} = \min\{|a| \mid a \in \boldsymbol{I}\}$

[4] $\mathsf{mag}\,\boldsymbol{I} = \max\{|a| \mid a \in \boldsymbol{I}\}$

[5] See, e.g., `http://www.mat.univie.ac.at/~neum/glopt/coconut/benchmark/Library3.html`

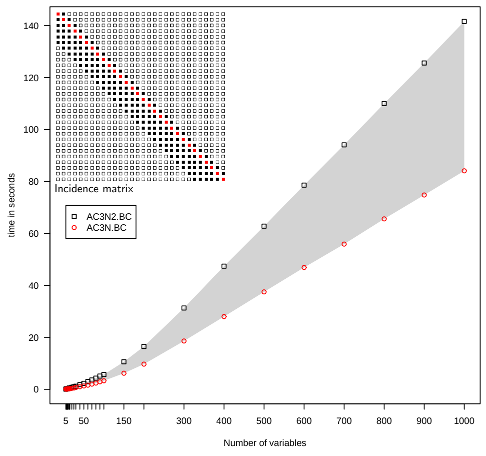

Figure 4: Solving Broyden-banded

For Broyden-banded, the speed-up gained with the heuristics is significant, though not tremendous, for large instances. This is in accordance with the low density of the incidence matrix of the problem. We have also tested the AC3N.HR algorithm. However it was not efficient enough to allow us solving large instances ($n > 50$). It seems that the partial derivatives used in the Newton steps contain 0 most of the time; the HR method is then unable to tighten the domains of the variables and therefore, the AC3N.HR algorithm does not perform much better than a simple dichotomic process.

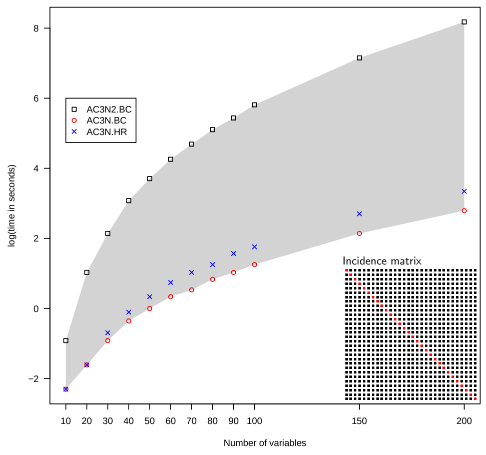

Figure 5: Solving Moré-Cosnard

For Moré-Cosnard, the speed-up obtained with AC3N.BC and our heuristics is impressive (hence the logarithmic scale for the time in Fig. 5). Once again, this is in accordance with the high density of the problem: AC3N2.BC has to handle n^2 projections instead of n well-chosen projections only for AC3N.BC. Contrary to the Broyden-banded problem, the AC3N.HR algorithm fares well here, though its efficiency is still a step behind the one of AC3N.BC.

5. CONCLUSION

We have tested the mignitude/magnitude-based heuristics on a dozen of standard benchmarks with various structures (polynomial or not) and various densities. The speed-ups obtained range from "real" to "impressive" even when computing the Jacobian only once. We suspect however that for very difficult problems it may be necessary to recompute it more than once during the solving process. As a side note, it is worthwhile contrasting our use of the Jacobian during the *narrowing process* with the "smear value" method introduced by Kearfott [8] that uses it to direct the *search process*.

Herbort and Ratz [7] suggest some other heuristics to select projection constraints for the Gauss-Seidel-Newton method. They do not restrict themselves to transversals and allow one variable to be tightened by several projections. According to our tests of their method implemented in CXSC 2.0, the results we obtain with our method are much better, though it is unclear what is contributed by the surrounding framework (interval library, propagation algorithm, . . .) and what by the heuristics proper.

In a different context, Sotiropoulos *et al.* [13] propose some heuristics to compute good transversals for polynomial systems based on the structure of the constraints. To assess their methods in our framework and to devise more advanced structure-based heuristics are our goals for future researches. In particular, symbolic manipulation of a system to achieve a "structural diagonal dominance" where only one variable is "preeminent" per constraint (e.g., with a much higher degree) seems a promising approach.

6. REFERENCES

[1] P. I. Barton. The equation oriented strategy for process flowsheeting. Dept. of Chemical Eng. MIT, Cambridge, MA, Mar. 2000.

[2] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Proc. of ILPS '94*, pages 124–138, Ithaca, NY, Nov. 1994. The MIT Press.

[3] M. Ceberio and L. Granvilliers. Solving nonlinear systems by constraint inversion and interval arithmetic. In *Proc. AISC '2000*, volume 1930 of *LNAI*, pages 127–141. Springer, 2000.

[4] H. J. Curnow and B. A. Wichmann. A synthetic benchmark. *Comput. J.*, 19(1):43–49, 1976.

[5] L. Granvilliers. On the combination of interval constraint solvers. *Rel. Comp.*, 7(6):467–483, 2001.

[6] E. R. Hansen and S. Sengupta. Bounding solutions of systems of equations using interval analysis. *BIT*, 21:203–211, 1981.

[7] S. Herbort and D. Ratz. Improving the efficiency of a nonlinear-system-solver using a componentwise newton method. RR 2/1997, Inst. für Angewandte Mathematik, Karslruhe, 1997.

[8] R. B. Kearfott and M. N. III. INTBIS, a portable interval newton/bisection package (algorithm 681). *ACM TOMS*, 16:152–157, 1990.

[9] O. Lhomme, A. Gotlieb, and M. Rueher. Dynamic optimization of interval narrowing algorithms. *J. of Logic Prog.*, 37(1–3):165–183, 1998.

[10] A. K. Mackworth. Consistency in networks of relations. *AI*, 1(8):99–118, 1977.

[11] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[12] J. M. Ortega and W. C. Rheinboldt. *Iterative solutions of nonlinear equations in several variables*. Academic Press Inc., 1970.

[13] D. G. Sotiropoulos, J. A. Nikas, and T. N. Grapsa. Improving the efficiency of a polynomial system solver via a reordering technique. In *Proc. 4th GRACM Congress on Computational Mechanics*, 2002.