An Analysis of Slow Convergence in Interval Propagation[⋆]

Lucas Bordeaux[1], Youssef Hamadi[1], and Moshe Y. Vardi[2,⋆⋆]

[1]Microsoft Research Cambridge, UK
{lucasb,youssefh}@microsoft.com
[2]Rice University, USA
vardi@cs.rice.edu

Abstract. When performing interval propagation on integer variables with a large range, *slow-convergence* phenomena are often observed: it becomes difficult to reach the fixpoint of the propagation. This problem is practically important, as it hinders the use of propagation techniques for problems with large numerical ranges, and notably problems arising in program verification. A number of attempts to cope with this issue have been investigated, yet all of the proposed techniques only guarantee a fast convergence on specific instances. An important question is therefore whether slow convergence is *intrinsic* to propagation methods, or whether an improved propagation algorithm may exist that would avoid this problem. This paper proposes the first analysis of the slow convergence problem under the light of complexity results. It answers the question, by a negative result: if we allow propagators that are general enough, computing the fixpoint of constraint propagation is shown to be intractable. *Slow convergence is therefore unavoidable unless P=NP.* The result holds for the propagators of a basic class of constraints.

1 Motivation and Results of the Paper

Problems with Large Discrete Ranges. Constraint propagation is probably the most developed component of CP (Constraint Programming) solvers, and the propagation of many constraints has been intensely studied. In this paper we consider variables ranging over a discrete domain and focus on *interval* propagation techniques, which are often used when dealing with numerical constraints. (We assume that the reader is familiar with interval propagation, otherwise see [2,3,4,5].) The question we address, put quickly, is whether interval propagation is effective against variables with a large range. Note that a number of applications require variables with large ranges: the best example is perhaps software verification, an area that heavily relies on constraint solving, but in which CP

⋆ An unabridged version of this paper including the missing proofs is available as a Microsoft Research Report.

⋆⋆ Part of this work was done while this author was visiting Microsoft Research, Cambridge, UK.

© Springer-Verlag Berlin Heidelberg 2007

techniques have so far failed to have a significant impact. In verification problems, the ranges of numerical variables are typically extremely large, because the aim is typically one of the following:

– *To verify a property for all integers.* Typically, if the constraints are simple enough, so-called "small-domain" properties are used to bring the problem down to finite bounds. These properties guarantee that a solution can be found within some finite bounds, *iff* the problem is satisfiable. These bounds are however typically quite large: for instance for purely linear equality constraints, [13] proves that when we have m constraints over n variables, the variables can be restricted to the range $[0, n(ma)^{2m+1}]$, where a is the maximum of the absolute values of the coefficients of the linear constraints. If we have only 10 variables, 10 equalities and coefficients within $[-10, 10]$, this bound already goes as high as $10 \cdot 100^{21} = 10^{43}$. These bounds can be refined [15] but we cannot, in general, avoid the use of large numbers represented in infinite-precision.
– *To reflect machine encoding of numbers.* In this case we typically compute within bounds of about 2^{32} or 2^{64}. Extra care typically has to be taken, so that overflows are correctly handled (which requires "modular arithmetics"). Here the domains are smaller, but nonetheless large enough for the slow propagation problem to become a serious issue.

The Problem of Slow Convergence. When performing interval propagation on numerical variables with a large range, *slow-convergence* phenomena have been reported by many authors, *e.g.,* [8,11]: propagation tends to go on for a prohibitively long number of steps. The problem is easily understood by considering simple examples:

– Consider the problem $X_1 < X_2 \wedge X_2 < X_1$, with X_1 and X_2 ranging over $[0, 2^{30}]$. Bound propagation alone detects the inconsistency. On this example, standard propagation algorithms discover that $X_1 \in [1, 2^{30}]$ because $0 \leq X_2 < X_1$, then $X_2 \in [2, 2^{30}]$ because $1 \leq X_1 < X_2$, and propagation goes ahead narrowing a lower or upper bound by one unit at every step. We ultimately obtain empty intervals, but this requires about 2^{30} operations.
– As mentioned in [11], the problem sometimes even occurs when we have a single constraint. For instance if we take the constraint $2X_1 + 2X_2 = 1$ with X_1 and X_2 ranging over $[-2^{30}, 2^{30}]$, we have a similar problem as before: propagation slowly narrows the bounds of the intervals by a few units until reaching empty intervals.

To solve these two examples, propagation will typically take several seconds. The problem becomes much more severe whenever similar constraints are not stated by themselves, but together with other constraints. In this case the runtime can become arbitrarily high, as propagation may regularly reconsider many propagators between each reduction of the bounds of X_1 and X_2.

It would be a mistake to consider slow convergence as a mere curiosity arising only in annoying, yet artificial examples. Our experience is that the problem is unavoidable when solving problems in program verification, for instance problems from Satisfiability Modulo Theories[1]. In examples arising from our own experiments in software verification, reaching the fixpoint of one propagation step alone takes seconds or minutes on many instances, and up to 37 hours (in finite precision!) in some of the longer examples where we waited until completion. Note that propagation is supposed to be the fast part of constraint solving: it is done at every node of the branch & prune process, and we are supposed to be exploring many nodes per second.

Attempted Solutions. Several solutions to the slow convergence problem have been investigated in the literature. It was, for instance, suggested to:

– *Detect some cases of slow convergence and find ways to prevent them.* One way would be to use symbolic techniques to get rid of constraints of the form $X_1 < X_2 \wedge X_2 < X_1$ and similar "cycles of inequalities". A related approach was suggested (in the continuous of real-valued intervals) in [12]. Unfortunately, these methods only prevent particular cases of slow convergence.
– *Reinforce interval propagation by other reasoning techniques.* A noticeable recent work on the issue is [11], which use congruence computations in addition to interval propagation. Our experience, however, is that congruence reasoning hardly ever speeds-up propagation in practice, and that it is powerless against very simple cases of slow convergence, *e.g.,* $X_1 < X_2 \wedge X_2 < X_1$.
– *Find a new algorithm* that would avoid the pitfalls of the standard interval propagation algorithm, and that would provably converge quickly. So far no such algorithm has been proposed. (An interesting related work is [10] which uses extrapolation methods to "guess" the possible fixpoint; this is an exciting method but it offers, by definition, no proven guarantee.)
– *Interrupt propagation after a given number of steps*, for instance prevent the propagation of variables whose width have been reduced by less than 5%. This is a pragmatic solution that it easy to implement, its drawback is that it leaves a search space partially reduced, relying on more branching.
– *Do something else than interval propagation.* For instance in [8], the authors note the slow convergence phenomenon and introduce a method for dealing with certain linear constraints between 2 variables; another significant example is that state-of-the-art methods in satisfiability modulo theories use bound reasoning methods that are not based on interval propagation, but on linear relaxations [6].

Our Results. The approaches mentioned previously do not solve the slow convergence problem: no approach allows to compute the fixpoint of interval propagation while being guaranteed to avoid slow convergence. Can this problem be circumvented, or is there an unsuspected, *intrinsic* reason why slow convergence is unavoidable? We believe this is an important question for the field that, surprisingly, has not been studied in the literature on CP theory.

[1] www.smt-lib.org

The authors of [8] (introduction) are perfectly right in their analyzis of slow convergence: it is due to the fact that the number of steps is *proportional to the width of the intervals* (the width of an interval is here defined as the number of integer values in the interval). The question is whether we can reduce this to a number of steps that grows significantly less than linearly in the width. This can be stated precisely by saying that the complexity should be *polynomial in the number of bits of the integer values* encoded in the problem *i.e.,* poly-logarithmic in these integer values. In contrast, existing propagation algorithms are easily seen to be *polynomial in these integer values*, *i.e.,* exponential in the number of bits of their encodings. Following classical terminology we call an algorithm of the first type *strongly polynomial* and an algorithm of the second type *pseudo-polynomial* (formal definitions are to be found in Section 2). The question is therefore whether there exists a strongly polynomial algorithm for interval propagation. We answer this question by the negative, under the P $\neq$ NP assumption.

Our results make assumptions on the type of constraints that are propagated. We first state the result in the general case, where arbitrary user-defined "interval propagators" can be defined (Prop. 1 and 2). We next show in Prop. 3 that the result still holds even if we restrict ourselves to a simple class of propagators, namely linear constraints plus one simple non-linear operation (squaring). We leave open the question whether the intractability result still holds when we deal with purely linear constraints. However, it is clear that CP was never meant to deal solely with linear constraints and our results therefore show what we believe is an intrinsic problem of interval propagation methods.

The next section gives more formal definitions of interval propagation which, following a number of authors [1], we see as a form of fixpoint computation; Section 3 will then list our main results. The missing proofs, as well as a more detailed presentation, can be found in the unabridged version of this paper.

2 Interval Propagation and Fixpoint Computations

Closure Operators. Interval propagation is equivalent to the problem of computing certain fixpoints of functions on Cartesian products of intervals. A Cartesian product of intervals will be called *box*, for short:

Definition 1 (Box). *An n-dimensional* box *is a tuple $B = \langle B_1 \cdots B_n \rangle$ where each B_i is an interval. Inclusion over boxes is defined as follows: $B \subseteq B'$ iff $B_1 \subseteq B'_1 \wedge \cdots \wedge B_n \subseteq B'_n$.*

The functions we consider must have the following properties:

Definition 2 (Closure operator). *We call* closure operator *a function f which, given a box B, returns a box $f(B)$, with the following properties (for all B, B'):*

1. *f is "narrowing": $f(B) \subseteq B$;*
2. *f is monotonic: if $B \subseteq B'$ then we have $f(B) \subseteq f(B')$;*
3. *f is idempotent: $f(f(B)) = B$.*

Computational Problems related to Fixpoints. A fixpoint of a closure operator is a box that remains unchanged after application of the operator. We are interested in common fixpoints, defined as follows:

Definition 3 (Common Fixpoint of a set of closure operators). *Given a set of closure operators $\{f_1 \cdots f_m\}$, a common fixpoint of the operators is a box B satisfying $f_1(B) = B \wedge \cdots \wedge f_m(B) = B$.*

We are given an initial n-dimensional box B, and a set of closure operators $\{f_1 \cdots f_m\}$. We consider the following computational problems (the first two are "function problems" in which we aim at computing a result, the third is a "decision problem" in which we just aim at determining whether a certain result exists):

Problem 1 (Computation of Common Interval Fixpoint). *Compute a box B' such that (1) $B' \subseteq B$ and (2) B' is non-empty and (3) B' is a common fixpoint of $f_1, \ldots, f_m$. (a special return value is used to signal the case where no non-empty fixpoint exists.)*

Problem 2 (Computation of the Greatest Common Interval Fixpoint). *Compute a box B' such that (1) $B' \subseteq B$; (2) B' is a common fixpoint of $f_1, \ldots, f_m$ and (3) no box B'' such that $B' \subsetneq B'' \subseteq B$ is a common fixpoint of $f_1, \ldots, f_m$. (An empty box should be returned if no other such fixpoint exists.)*

Problem 3 (Existence of Common Interval Fixpoint). *Determine whether there exists a non-empty box $B' \subseteq B$ which is a common fixpoint of $f_1, \ldots, f_m$.*

Greatest Fixpoint Computation by "Chaotic Iteration". To compute a greatest fixpoint, the standard approach is to run what [1] refers to as a "chaotic iteration" algorithm which, in its simplest and least optimized form, can be presented as follows:

Algorithm 1. Standard Algorithm for Greatest Fixpoint Computation

```
while there exists $f_i$ such that $f_i(B) \neq B$ do
    Choose one such $f_i$
    $B \leftarrow f_i(B)$
```

The functions $f_1 \ldots f_m$ are applied to the box in turn, in any order that is computationally convenient, until we reach a state where nothing changes. A basic result, which directly follows from the Knaster-Tarski theorem [16], is that this algorithm, although non-deterministic, always converges to the greatest common fixpoint of $f_1 \ldots f_m$. (The uniqueness of this fixpoint shows in particular, that the box satisfying the requirements of Problem 2 is unique, and allows us to refer to it as *the* (unique) greatest common interval fixpoint.)

Strongly Polynomial *vs.* Pseudo-Polynomial Algorithms. Denoting by n the dimension of the considered box (*i.e.,* number of variables), m the number of operators whose fixpoint we compute, and w the maximum of the widths, we use the following definitions, which follow classical terminology:

– A *pseudo-polynomial* algorithm is an algorithm whose runtime is bounded in the worst case by $P(n, m, w)$, for some polynomial P;
– A *strongly polynomial* algorithm is an algorithm whose runtime is bounded in the worst case by $P(n, m, \log w)$, for some polynomial P.

It is straightforward to check that the classical "chaotic iteration" algorithm for interval propagation, as well as all the improved versions derived from it, are only pseudo-polynomial, and can therefore be subject to slow convergence.

3 Intractability of Interval Propagation

In this section we present our main results, which show that, under some well-defined assumptions concerning the operators, computing the fixpoint of these operators cannot be achieved in strongly polynomial time.

General Case. We first consider the "general case", in which the closure operators are defined as arbitrary functions. This captures, for instance, the ability of systems like Constraint Handling Rules (CHR) [7], in which the propagators can be user-defined.

We assume that the propagators $f_1 \ldots f_m$ are defined as programs (written in any appropriate language, like CHR), which have the additional guarantee to run in time polynomial in the length of the problem. This is because we want to show that the problem is intractable even when restricted to simple propagators (a hardness result would hardly be a surprise in the case where the execution of a propagator is itself intractable.) More precisely, the input of the fixpoint representation problem is as follows:

Input Representation 1. *The input is given as a box $B = \langle B_1 \ldots B_n \rangle$ together with a set of closure operators $\{f_1 \ldots f_m\}$ which are defined as programs whose runtime is guaranteed to be worst-case polynomial in the total input size.*

Proposition 1. *If the input is encoded using Representation 1, the problem of existence of a common interval fixpoint (Problem 3) is NP-complete.* [2]

[2] The hardness part of this result can alternatively be proven as a direct consequence of Prop. 3. We prove Prop. 1 separately for 3 reasons: it states the membership in NP under the more general assumptions (the closure operators need be polytime computable); it states the NP-hardness under the least restrictive assumptions (one-dimensional case, two operators); Prop. 2 is best presented by first presenting the proof of Prop. 1.

Note that the result even holds in dimension one and for only two operators. Indeed, if we consider non-idempotent narrowing operators instead of closure operators, it is easy to show that computing the fixpoint of one single operator in one dimension is NP-complete. One can show, by straightforward modifications of the proof, that the corresponding function problem, computing an arbitrary fixpoint (Problem 1), is FNP-complete[3]. Interestingly, propagation algorithms do not compute an arbitrary fixpoint, but the *largest* one (Problem 2). The problem is therefore an optimization problem and, in fact, its complexity is higher than FNP:

Proposition 2. *If the input is encoded using Representation 1, the computation of the greatest common fixpoint (Problem 2) is OptP-complete.*

OptP is a class introduced in [9] to characterize the complexity of optimization problems (many optimization problems are FNP-hard but not in FNP because the optimality of the result cannot be checked in polynomial time).

Basic Numerical constraints. We now refine our analysis to the case where we have "basic propagators": what if the user does not have the possibility to write her own propagators, but can only use a set of predefined propagators for basic constraints? For the sake of concreteness we now focus on a simple set of propagators, that we now define precisely.

Variables are numbered from 1 to n; the kth variable is denoted X_k and the interval that is associated with this variable is denoted $[l_k, r_k]$. The notation "$[l_k, r_k] \leftarrow rhs$" denotes an operator f which, given a box B, returns a box $f(B)$ in which the kth interval has been modified as specified by the right-hand side (rhs), and all other intervals are unchanged.

We consider the following operators for a constraint $X_i < X_j$:

$$\begin{array}{l} [l_i, r_i] \leftarrow [l_i, \min(r_i, r_j - 1)] \\ [l_j, r_j] \leftarrow [\max(l_j, l_i + 1), r_j] \end{array} \tag{1}$$

(For readers who would have trouble with the notation: the upper bound of the ith interval, the one associated with X_i, is updated so that it is at most $r_j - 1$, and the lower bound of the jth interval is updated so that it is at least $l_j + 1$.)

We consider the following operators for a constraint $X_i = X_j^2$:

$$\begin{array}{l} [l_i, r_i] \leftarrow [\max(l_i, l_j^2), \min(r_i, r_j^2)] \\ [l_j, r_j] \leftarrow [\max(l_j, \lceil \sqrt{l_i} \rceil), \min(r_j, \lfloor \sqrt{r_i} \rfloor)] \end{array} \tag{2}$$

We consider the following operators for a constraint $aX_i + bX_j = c$, where a, b and c are non-negative integer constants:

$$\begin{array}{l} [l_i, r_i] \leftarrow [\max(l_i, \lceil \frac{c - b \cdot r_j}{a} \rceil), \min(r_i, \lfloor \frac{c - b \cdot l_j}{a} \rfloor)] \\ [l_j, r_j] \leftarrow [\max(l_j, \lceil \frac{c - a \cdot r_i}{b} \rceil), \min(r_j, \lfloor \frac{c - a \cdot l_i}{b} \rfloor)] \end{array} \tag{3}$$

[3] FNP, or "functional" NP, is closely related to NP, the difference being that instead of being asked whether a solution exists (say, to a SAT instance), we are asked to produce a solution [14].

Input Representation 2. *The problem is given as a box $B = \langle B_1 \dots B_n \rangle$ together with a set of constraints $\{c_1 \dots c_m\}$ which include the following 3 forms:*

1. $X_j < X_j$*, for some* $i, j \in 1..n$*;*
2. $X_i = X_j^2$*, for some* $i, j \in 1..n$*; or*
3. $aX_i + bX_j = c$*, for some* $i, j \in 1..n$ *and some constants* a*,* b *and* c*.*

Proposition 3. *If the input is encoded using Representation 2, the problem of existence of a common interval fixpoint (Section 2, Problem 3) is NP-complete.*

References

1. Apt, K.R.: The essence of constraint propagation. Theoretical Computer Science (TCS) 221(1-2), 179–210 (1999)
2. Apt, K.R., Zoeteweij, P.: An analysis of arithmetic constraints on integer intervals. Constraints (to appear)
3. Benhamou, F., Older, W.J.: Applying interval arithmetic to real, integer, and boolean constraints. J. of Logic Programming (JLP) 32(1), 1–24 (1997)
4. Cleary, J.G.: Logical arithmetic. Future Computing Systems 2(2), 125–149 (1987)
5. Davis, E.: Constraint propagation with interval labels. Artificial Intelligence 32(3), 281–331 (1987)
6. Dutertre, B., De Moura, L.M.: A fast linear arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
7. Fruehwirth, T.W.: Theory and practice of constraint handling rules. J. of Logic Programming (JLP) 37(1-3), 95–138 (1998)
8. Jaffar, J., Maher, M.J., Stuckey, P.J., Yap, R.H.C.: Beyond finite domains. In: Borning, A. (ed.) PPCP 1994. LNCS, vol. 874, pp. 86–94. Springer, Heidelberg (1994)
9. Krentel, M.W.: The complexity of optimization problems. In: Proc. of ACM Symp. on Theory of Computing (STOC), pp. 69–76. ACM Press, New York (1986)
10. Lebbah, Y., Lhomme, O.: Accelerating filtering techniques for numeric CSPs. Artificial Intelligence 139(1), 109–132 (2002)
11. Leconte, M., Berstel, B.: Extending a CP solver with congruences as domains for program verification. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 22–33. Springer, Heidelberg (2006)
12. Lhomme, O., Gottlieb, A., Rueher, M., Taillibert, P.: Boosting the interval narrowing algorithm. In: Proc.of Joint Int. Conf. and Symp. on Logic Programming (JICSLP), pp. 378–392. MIT Press, Cambridge (1996)
13. Papadimitiou, C.: On the complexity of integer programming. J. of the ACM 28(4), 765–768 (1981)
14. Papadimitriou, Ch.H.: Computational Complexity. Addison Wesley, Reading (1994)
15. Seshia, S.A., Bryant, R.A.: Deciding quantifier-free presburger formulas using parametrized solution bounds. Logical Methods in Computer Science, vol. 1(2) (2005)
16. Tarski, A.: A lattice-theoretic fixpoint theorem and its applications. Pacific J. of Mathematics 5, 285–309 (1955)