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 Introduction

Constraint propagation is probably the most developed component of CP (Constraint Programming) solvers, and the propagation of many constraints has been intensely studied. In particular, a considerable literature has been devoted to arc-consistency in constraints represented as binary or n-ary arbitrary predicates (*e.g.,* tables), see for instance [5]. Many efficient propagators for complex global constraints have also been proposed [27].

Propagation methods can be roughly classified as *domain* propagation or *interval* propagation methods, depending on whether the set of values allowed for each variable is represented exactly as a list/bit-vector, or approximated by its bounds. The interval representation is typically preferred for numerical variables, because they can have a very high number of values. Interval propagation is therefore used for numerical constraints by most CP solvers. This paper analyzes bound propagation [2, 4, 6, 7] and focuses specifically on the case where the variables take discrete (integer) values. The question we address, put quickly, is whether interval propagation is effective against variables with a large range.

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

When performing interval propagation on numerical variables with a large range, which are common, for instance, in program verification, *slow-convergence* phenomena have been reported by many authors, *e.g.,* [13, 17]: propagation tends to go on for a prohibitively long number of steps. A number of approaches, which are briefly reviewed in the paper, have been proposed to address this problem, but all of these approaches have limitations and none of them completely prevents the risk of slow convergence. In this paper we explain the absence of definitive solution to this problem: slow convergence is not an issue of implementation, nor a problem that could potentially be avoided by algorithmic tricks, but an *intrinsic* problem of interval propagation, in the precise sense that it cannot be avoided unless P = NP. More precisely, our approach is the following:

- In interval propagation, the problem is to compute fixpoints of certain types of operators. We investigate the computational complexity of these fixpoint computations.
- This complexity is *pseudo-polynomial* and, as we explain, the question is whether a *strongly polynomial* algorithm exists for the fixpoint computation problem.
- We answer this question by the negative: we first analyze the complexity of the fixpoint computation problem in its general version, and show that strongly polynomial algorithms for this problem cannot exist if P $\neq$ NP. We then consider particular types of "operators" and, in particular, when we mix the propagators of linear constraints with simple non-linear propagators (one squaring constraint suffices), we still have the hardness result. If we particularize further the problem to purely linear constraints, the problem is open but we conjecture, here again, that the complexity is NP-complete.

The paper introduces required material on fixpoint computation in Section 2, on interval propagation in Section 3, and gives more details on the slow convergence problem in Section 4. Then follows Section 5 with a discussion of strongly polynomial *vs.* pseudo polynomial complexity. Our main results are presented in Section 6. These results are commented and put in perspective in Section 7.

2 Fixpoint Computations

We first review some basic material on fixpoint computations. For more reading on this issue the reader is referred to [1].

2.1 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: $f(f(B)) = B$.*

2.2 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) $B' \subseteq B$, (2) B' is non-empty and (3) B' is a common fixpoint of $f_1, \ldots, f_m$. (An error value should be returned if no such 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 $B' \subseteq B'' \subseteq B$ is a common fixpoint of $f_1, \ldots, f_m$.*

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

2.3 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 $f_1 \dots f_m$ 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 [25], is that this algorithm, although non-deterministic, always converges to the greatest common fixpoint of $f_1 \dots f_m$. (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.)

3 Interval Propagation

Readers are assumed familiar with basics of interval propagation and referred to *e.g.,* [2] otherwise.

3.1 Propagation as Fixpoint Computation.

Interval propagation is the problem of fixpoint computation as described above:

- Each of the n variables has its own interval of values and the state of the variables is seen, altogether, as an n-dimensional box;
- The closure operators are "propagators" which can be either predefined (propagators associated with the standard constraints delivered by a system), or user-defined (many systems, *e.g.,* CHR [10], allow the users to define their own propagators). A propagator is seen as a function which, from the current ranges of the variables, computes ranges that are tighter or equal.

Let us recall briefly why the assumptions made in Def. 2 are reasonable. The property of "narrowing" is, obviously, absolutely desired, since the very aim of a propagator is to remove some values that are inconsistent. Monotonicity reflects the intuition that the more information is available on the ranges, the more deductions the propagator should do. It is very difficult to think of propagators that would not be monotonic, and this is typically not desired: the computed intervals would be dependent on the order in which such propagators are considered, making it very difficult to get a sense of what deductions are made. Idempotence reflects the intuition that "applying the propagator twice in a row does not pay". Some authors, *e.g.,* [23], impose narrowing and monotonicity but see idempotence as optional. We occasionally consider operators that are narrowing and monotonic but not idempotent; we call them *narrowing operators*.

A fixpoint characterizes the state of the problem when propagation has stopped, *i.e.,* no propagator is able to reduce the intervals anymore. More specifically, propagation computes the largest such fixpoint.

3.2 Examples of Propagators.

We now give the precise definitions of the propagators that are used for examples and proofs in the rest of the paper. 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.

The following propagators are used when we have 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 $l_j + 1$.)

The propagators for a constraint $X_i = X_j^2$ are:

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

The propagators for a constraint $aX_i + bX_j = c$, where a, b and c are non-negative integer constants, are:

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

(The constraint $aX_i + bX_j = c$ is, of course, only a particular case of linear constraint, and the way we have expressed the propagator is just one way of expressing what general propagators for linear constraints would do [12].)

One can check that all these operators are narrowing, monotonic and idempotent. They are also correct *w.r.t.* the constraint, in that they never remove a solution.

4 Slow Convergence

This section gives more background on the slow convergence phenomenon, why it is a serious issue, and which approaches have been proposed to tackle it.

4.1 Examples of Slow Convergence.

The problem of slow convergence 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 [17], 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.

Readers are encouraged to run the propagation-based solvers they have at their disposal against these straightforward problems: except if problem-specific optimizations such as those described in [17] are available, propagation alone invariably takes several seconds. More severe is the fact that, if similar constraints are not stated by themselves but together with other constraints, 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.

4.2 Problems with Large Numerical Ranges.

It would be a mistake to consider slow convergence as a mere curiosity arising only in annoying, yet artificial examples. Our own experience is that the problem is unavoidable when solving problems in program verification, for instance problems from Satisfiability Modulo Theories[1].

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, if the problem is satisfiable, a solution can be found within some finite bounds. But these bounds are typically quite large: for instance for purely linear equality constraints, [20] 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 [24] 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.

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

[1] www.smt-lib.org

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.

4.3 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, suggested by colleagues in personal communications, 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 [18] (in the context of real-valued intervals): it dynamically detects "cycles" in the propagation and deals with them separately, for instance by postponing the corresponding operators. 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 [17], which use congruence computations in addition to interval propagation. Congruences capture information of the type "all possible values of the variable have the form $ax + b$ for some x". This allows to directly deduce the inconsistency of constraints like $2X_1 + 2X_2 = 1$ (congruences find out that the sum is even while the right-hand-side is odd). Here again, the technique is however powerless against very simple cases of slow convergence, for instance the example $X_1 < X_2 \wedge X_2 < X_1$.
- *Interrupt propagation after a given number of steps*, or when a given precision is reached (*e.g.,* propagate further a variable only when its width has been reduced by more than 5%). This is a pragmatic solution that it easy to implement. What is frustrating, of course, is that the search space is then left in a state where obvious deductions could be made but are postponed simply by fear of slow convergence: we leave it up to the search mechanism to perform these obvious deductions.
- *Find a new algorithm* that would avoid the pitfalls of the standard "chaotic iteration" presented before, and which provably converge quickly. For instance, when investigating the question, we have struggled in vain to find "dichotomic" algorithms that would compute a fixpoint more quickly. (This investigation has failed, and led us to consider the question whether the problem is intrinsically unsolvable.) An interesting related work is [16] which (also in the context of continuous intervals) uses extrapolation methods to "guess" the possible fixpoint—an exciting method which, by definition, offers no proven guarantee.
- *Do something else than interval propagation.* If interval propagation does not work well for some classes of problems, perhaps the best solution is to use different pruning techniques. For instance, in [13], the authors note the slow convergence phenomenon and introduce a method for dealing with certain linear constraints between 2 variables. Significantly, state-of-the-art methods in satisfiability modulo theories use bound reasoning methods that are not based on interval propagation, but on linear relaxations [9].

All of these approaches are of interest but none of them completely solves the problem: no approach allows to compute the fixpoint of interval propagation while being guaranteed to avoid slow convergence.

5 Pseudo-Polynomial *vs.* Strongly Polynomial Algorithms

Interval propagation exhibits features that could, in some respects, sound unusual: the standard propagation algorithm is reputedly polynomial, yet its runtime can grow unreasonably high for but simple instances. The question, when discussing polynomial *vs.* exponential runtimes, is to be careful of not being polynomial in some features that can grow exponentially.

5.1 Definitions.

The authors of [13] (introduction) are perfectly right in their analyzis of the slow convergence of problems of the type $X_1 < X_2 \wedge X_2 < X_1$: the slow convergence 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*.

As stated before, n is the dimension of the considered box (*i.e.,* number of variables), m is the number of operators whose fixpoint we compute. Furthermore, let w denote the maximum of the widths. We use the following definitions, which follow classical terminology [11]:

- 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.

The question we address is whether there exists a strongly polynomial algorithm for interval propagation.

5.2 Pseudo-Polynomiality and Fixed-Parameter Tractability.

As the name suggests, a "pseudo"-polynomial algorithm is in fact exponential: the complexity of an algorithm is, strictly speaking, measured as the function of the length (number of bits) of the machine representation of the problem, and using k bits to represent w we have an upper bound of $P(n, m, 2^k)$ on the runtime. A pseudo-polynomial complexity has nevertheless some advantages compared to a complexity that would be exponential, say, in n (*e.g.,* $\mathcal{O}(mw^n)$). In particular the availability of a pseudo-polynomial algorithm for a problem

shows that the problem is *fixed-parameter tractable* [8], *i.e.,* if we bound the number of bits allowed for the numbers (*i.e.,* we work in fixed precision, in 32 bits), the problem becomes, strictly speaking, polynomial. But what can make the algorithm slow remains the same issue: if the constant is huge, say $w = 2^{32}$, the question is whether we can reduce this constant down dramatically, or if we are forced to run the propagators about w times before reaching the fixpoint.

5.3 Interval Propagation is Pseudo-Polynomial.

It is easy to see that even the naive version of the standard interval propagation algorithm (Algorithm 1, Section 2.3) is pseudo-polynomial: at each loop iteration we reduce at least one interval by one unit, and the number of iterations is therefore bounded by nw. Within each loop we have to go through all operators and see whether applying one of them has an effect, which gives overall a bound of $\mathcal{O}(mnw)$ operator applications. (If we consider a typical basic operator, the cost of its application is typically constant or linear in the number of variables it connects.) Note that these worst-case time complexities are ironically not significantly different from the ones obtained for the *domain propagation* [1] of, say, numerical constraints: even though interval representation is more space efficient and to some extent more adapted to large domains, the worst-case time complexities are essentially the same.

5.4 Pseudo-Polynomial NP-complete Problems.

Some problems can be solved in pseudo-polynomial time, yet are NP-complete: any other NP problem can be reduced to them using a reduction that creates numbers whose size in bits grows polynomially in the size of the original problem The most famous such problem is the knapsack—solving one linear equality of the form $\sum_{i \in 1..n} k_i x_i = r$, where the k_is and r are constants and the xs are variables ranging over $\{0, 1\}$. This is an NP-complete problem which can nonetheless be solved in time $\mathcal{O}(nK)$, where K is the sum of the k_is, using dynamic programming algorithms similar to the one used in a CP context by [26]. Knowing that such problems are NP-complete tells us that the problem cannot be solved in strongly polynomial time, unless P=NP.

6 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.

6.1 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) [10], in which the propagators can be user-defined. The question is whether one can avoid slow convergences in this general context.

We assume that the propagators $f_1 \dots f_m$ 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 $B = \langle B_1 \dots B_n \rangle$ together with a set of closure operators $\{f_1 \dots f_m\}$ 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]

Proof. Membership in NP is due to the fact that a certificate of *yes* answers to the decision problem is simply a box (its being a fixpoint can be checked in polynomial time).

For the hardness we directly reduce from SAT: given a SAT instance in n variables $x_0 \dots x_{n-1}$, we create a 1-dimensional instance described by one initial interval $[l, r]$ and 2 operators f_0, f_1. The interval $[l, r]$ is defined as $[0, 2^n]$. The programs f_0 and f_1 use a common piece of code *check*(v) which, given an n-bit integer v, returns true iff assigning each x_i to the ith bit of v satisfies the formula. (Such a program is straightforward to define: we essentially encode the SAT formula into the instructions of the program.) The operator $f_0([l, r])$ is simply defined as:

$$\mathsf{if}\ \neg \mathit{check}(r) \wedge r \bmod 2 = 0\ \mathsf{then\ return}\ [l, r-1]\ \mathsf{else\ return}\ [l, r]$$

And the operator $f_1([l, r])$ as:

$$\mathsf{if}\ \neg \mathit{check}(r) \wedge r \bmod 2 = 1\ \mathsf{then\ return}\ [l, r-1]\ \mathsf{else\ return}\ [l, r]$$

It is straightforward to see that the resulting instance satisfies our requirements (the operators are narrowing, idempotent, monotonic; their size and runtimes are polynomially bounded) and that it has a fixpoint iff the SAT instance has a solution. (The upper bound of a fixpoint encodes a solution.) □

[2] The hardness part of this result can alternatively be proved 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.*

Proof. The membership in OptP is straightforward. The problem of *maximum satisfying assignment* (computing the lexicographically maximum assignment to a SAT instance) is OptP-complete [15]; the hardness is then proved by the same reduction presented in Prop. 1, where we simply notice that the reduction is a *metric reduction* in the sense of [15]. □

OptP is a class introduced in [15] 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).

6.2 Basic Numerical constraints.

Here our goal is to analyze the complexity of computing the fixpoint of "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 consider the particular set of 3 predefined operators listed in Section 3.2, *e.g.,* the propagators for inequality, squaring and binary linear constraints.

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 are all of one of the 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.

The closure operators represented in each of these three cases are as defined in Section 3.2.

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.*

[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 [21].

Proof. The membership in NP follows from the more general result obtained in Prop. 1. To prove the hardness we use an NP-completeness result due to Manders and Adleman [19], which states that determining whether an equation of the form $aX_1^2 + bX_2 = c$ has solutions, where a, b and c are three non-negative integers (in the sense of finding whether there exist natural values for the variables X_1 and X_2 that satisfy the equation), is NP-complete.

The reduction is simple: given the three constants a, b and c (encoded in binary) describing the input, we simply create an instance of the problem of existence of common fixpoint in dimension 3 where the initial box is $\langle [0,c],[0,c],[0,c]\rangle$ and with two constraints: $X_3 = X_1^2$ and $aX_3 + bX_2 = c$. In other words, we focus on the fixpoint of the following operators:

$$(1) \qquad [l_3, r_3] \leftarrow [\max(l_3, l_1^2), \min(r_3, r_1^2)]$$

$$(2) \qquad [l_1, r_1] \leftarrow [\max(l_1, \lceil\sqrt{l_3}\rceil), \min(r_1, \lfloor\sqrt{r_3}\rfloor)]$$

$$(3) \qquad [l_3, r_3] \leftarrow [\max(l_3, \lceil\tfrac{c-b\cdot r_2}{a}\rceil), \min(r_3, \lfloor\tfrac{c-b\cdot l_2}{a}\rfloor)]$$

$$(4) \qquad [l_2, r_2] \leftarrow [\max(l_2, \lceil\tfrac{c-a\cdot r_3}{b}\rceil), \min(r_2, \lfloor\tfrac{c-a\cdot l_3}{b}\rfloor)]$$

We now show that a common fixpoint to these operators exists iff the original equation has a solution.

- If a (non-empty) common fixpoint $\langle [l_1, r_1], [l_2, r_2], [l_3, r_3]\rangle$ exists, then its bounds satisfy the following:
  - $l_3 = \max(l_3, l_1^2)$ by Eq. (1), from which we deduce $l_3 \geq l_1^2$;
  - $l_1 = \max(l_1, \lceil\sqrt{l_3}\rceil)$ by Eq. (2), from which we deduce $l_3 \leq l_1^2$;
  - $l_3 = \max(l_3, \lceil\frac{c-b\cdot r_2}{a}\rceil)$ by Eq. (3), from which we deduce $l_3 \geq \frac{c-b\cdot r_2}{a}$, and $al_3 + br_2 \geq c$;
  - $r_2 = \min(r_2, \lfloor\frac{c-a\cdot l_3}{b}\rfloor)$ by Eq. (4), from which we deduce $r_2 \leq \frac{c-a\cdot l_3}{b}$, and $al_3 + br_2 \leq c$.

  Altogether this proves that $al_3 + br_2 = c \wedge l_3 = l_1^2$, therefore $X_1 = l_1, X_2 = r_2$ is a solution to the original equation $aX_1^2 + bX_2 = c$.
- If the equation $aX_1^2 + bX_2 = c$ has a solution $X_1 = m, X_2 = n$, then it is easy to verify that the box $\langle [m,m],[n,n],[p,p]\rangle$ where $p = m^2$ is a common fixpoint of the operators:
  - $p = m^2$ therefore $[p,p] = [\max(p, m^2), \min(p, m^2)]$;
  - $p = m^2$ therefore $\sqrt{p} = m$ is an integer and $\sqrt{p} = \lceil\sqrt{p}\rceil = \lfloor\sqrt{p}\rfloor$, and we have $[m,m] = [\max(m, \lceil\sqrt{p}\rceil), \min(m, \lfloor\sqrt{p}\rfloor)]$;
  - $ap + bn = c$ therefore $\frac{c-b\cdot n}{a} = p$ is an integer and $\lceil\frac{c-b\cdot n}{a}\rceil = \lfloor\frac{c-b\cdot n}{a}\rfloor = \frac{c-b\cdot n}{a}$. As a consequence we have $[p,p] = [\max(p, \lceil\frac{c-b\cdot n}{a}\rceil), \min(p, \lfloor\frac{c-b\cdot n}{a}\rfloor)]$;
  - $ap + bn = c$ therefore $\frac{c-a\cdot p}{b} = n$ is an integer and $\lceil\frac{c-a\cdot p}{b}\rceil = \lfloor\frac{c-a\cdot p}{b}\rfloor = \frac{c-a\cdot p}{b} = n$. As a consequence we have $[n,n] = [\max(n, \lceil\frac{c-a\cdot p}{b}\rceil), \min(n, \lfloor\frac{c-a\cdot p}{b}\rfloor)]$.

  Note, last, that the 3 values m, n and p are in $[0,c]$ since we have $ap+bn = c \wedge p = m^2$ and all values are non-negative integers. The box $\langle [m,m],[n,n],[p,p]\rangle$ is therefore a fixpoint satisfying the conditions given for Problem 3. □

7 Conclusion

7.1 Summary of the Contributions.

We believe that this paper answers an important question concerning one of the key propagation methods used in Constraint Programming. It is well-known that interval propagation suffers from the slow convergence problem when dealing with large intervals, and that this problem hinders its use in some important areas of applications. The question whether this problem can be circumvented is an important one.

We have shown that it cannot, in the precise sense that no algorithm computing the greatest fixpoint of a set of interval propagators can be strongly polynomial, unless P=NP. This result holds under assumptions on the type of propagators available: what Prop. 3 shows is that as soon as the set of propagators includes such basic constructs as linear constraints and squaring, the problem of determining whether the operators have a common fixpoint is NP-complete. This problem might become tractable when we restrict drastically the set of propagators, and we notably leave open the question whether such would be the case 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 show what we believe is an intrinsic problem of interval propagation methods.

This result may come as a surprise to some readers, as it did to us: we discussed the problem with several colleagues and our feeling is that there was a general belief that slow convergence could be tackled and that there was no underlying intractability result to be foreseen.

Our results "put discrete interval propagation on the map" from the perspective of complexity classification. In this respect, it is interesting to notice the following points:

- Interval propagation is the second AI problem we are aware of whose complexity is pseudo-polynomial and that is solved by a "propagation" process whose convergence can be slow: the problem of reaching stable states in Hopfield networks was the first such problem[4]; it has similarities but also differences in that the propagation performed in Hopfield networks is not monotonic in the same sense as interval propagation. See [21] (problem "HAPPYNET") for a textbook presentation of the problem.
- The complexity of computing greatest fixpoints for general propagators (Representation 1) is in a sense higher than the one of the original problem of constraint satisfaction (OptP-complete, as opposed to FNP-complete), which is somehow surprising. This is reminiscent of some phenomena already observed in discrete propagation: notably, [14] prove that computing k-consistency filtering is no less than EXPTIME-complete.

[4] This problem is not NP-complete but it is shown in [22] that no strongly polynomial algorithm exists for it under the weaker assumption that P$\neq$PLS.

7.2 Interpretation of the Results.

The message of the paper is clear: there are intrinsic issues in using discrete interval propagation methods for large domains. The areas where constraint propagation has been most successfully used concern combinatorial problems in which, typically, domains do not get unreasonably large. The recent area of Satisfiability Modulo Theories brought potential venues of new applications for CP in software verification, but our results cast doubts on the use of key CP techniques in this area. It is noteworthy that other propagation methods are currently being used by SMT solvers [9].

Our feeling is nevertheless that it would be a mistake to keep an overly negative message from this paper. First, our results are *worst-case* results and there are many instances where interval propagation just works fine, in which case it can prove very helpful. Second, it is very easy to control the slow convergence problem by interrupting propagation if it takes an excessive amount of time. What has to be given-up, in our opinion, is the idea that we should absolutely reach the fixpoint of interval propagation, as well as the idea that interval propagation can be used on large domains without being complemented by other relaxation techniques.

7.3 Future Works.

- An interesting open question is: if we restrict ourselves to the propagator of linear constraints, can we compute their greatest fixpoint in strongly polynomial time? (We suspect a negative answer.)
- It would, more generally, be interesting to exhibit useful classes of propagators whose fixpoint can be computed in strongly polynomial time.
- The paper has focused on variables ranging over the integers. Another area of CP considers interval propagation techniques for real-valued variables. It would be interesting to provide a similar complexity analysis in this context and to see, in particular, whether the good behaviour of some key techniques like Box-consistency can be explained by a better ability to avoid slow convergence [3].

References

1. K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science (TCS)*, 221(1-2):179–210, 1999.
2. K. R. Apt and P. Zoeteweij. An analysis of arithmetic constraints on integer intervals. *Constraints*, To appear.
3. F. Benhamou and L. Granvilliers. Continuous and interval constraints. In *Handbook of Constraint Programming*, chapter 14. Elsevier, 2006.
4. F. Benhamou and W. J. Older. Applying interval arithmetic to real, integer, and boolean constraints. *J. of Logic Programming (JLP)*, 32(1):1–24, 1997.
5. C. Bessière, J.-C. Régin, R. H. C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.

6. J. G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.
7. E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331, 1987.
8. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
9. B. Dutertre and L. M. De Moura. A fast linear arithmetic solver for DPLL(T). In *Proc. of Int. Conf. on Computer-Aided Verification (CAV)*, pages 81–94. Springer, 2006.
10. T. W. Fruehwirth. Theory and practice of constraint handling rules. *J. of Logic Programming (JLP)*, 37(1-3):95–138, 1998.
11. M. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
12. W. Harvey and P. J. Stuckey. Improving linear constraint propagation by changing constraint representation. *Constraints*, 8(2):173–207, 2003.
13. J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap. Beyond finite domains. In *Proc. of Int. Workshop on Principles and Practice of Constraint Programming (PPCP)*, pages 86–94. Springer, 1994.
14. P. G. Kolaitis and J. Pantajja. On the complexity of existential pebble games. In *Proc. of Int. WS. on Computer Science Logic (CSL)*, pages 314–329. Springer, 2003.
15. M. W. Krentel. The complexity of optimization problems. In *Proc. of ACM Symp. on Theory of Computing (STOC)*, pages 69–76. ACM, 1986.
16. Y. Lebbah and O. Lhomme. Accelerating filtering techniques for numeric CSPs. *Artificial Intelligence*, 139(1):109–132, 2002.
17. M. Leconte and B. Berstel. Extending a CP solver with congruences as domains for program verification. In *CP Workshop on Software Testing, Verification and Analysis*, pages 22–33, 2006.
18. O. Lhomme, A. Gottlieb, M. Rueher, and P. Taillibert. Boosting the interval narrowing algorithm. In *Proc.of Joint Int. Conf. and Symp. on Logic Programming (JICSLP)*, pages 378–392. MIT Press, 1996.
19. K. Manders and L. Adleman. NP-complete decision problems for quadratic polynomials. In *Proc. of ACM Symp. on Theory of Computing (STOC)*, pages 23–29. ACM, 1976.
20. C. Papadimitiou. On the complexity of integer programming. *J. of the ACM*, 28(4):765–768, 1981.
21. Ch. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
22. A. A. Schaeffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM J. on Computing*, 20(1):56–87, 1991.
23. C. Schulte and M. Carlsson. Finite domain constraint programming systems. In *Handbook of Constraint Programming*, chapter 14. Elsevier, 2006.
24. S. A. Seshia and R. A. Bryant. Deciding quantifier-free presburger formulas using parametrized solution bounds. *Logical Methods in Computer Science*, 1(2), 2005.
25. A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific J. of Mathematics*, 5:285–309, 1955.
26. M. A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. In *Proc. of Int. Conf. on Integration of AI and OR Techniques in CP for Combinatorial Optimisation Problems (CP-AI-OR)*, 2001.
27. W.-J. Van Hoeve and I. Katriel. Global constraints. In *Handbook of Constraint Programming*, chapter 6. Elsevier, 2006.