Reliable Computing 1 (3) (1995), pp. 343–359

A bright side of NP-hardness of interval computations: interval heuristics applied to NP-problems

Bonnie Traylor and Vladik Kreinovich

It is known that interval computations are NP-hard. In other words, the solution of many important problems can be reduced to interval computations. The immediate conclusion is negative: in the general case, one cannot expect an algorithm to do all the interval computations in less than exponential running time.

We show that this result also has a bright side: since there are many heuristics for interval computations, we can solve other problems by reducing them to interval computations and applying these heuristics.

Выгодная сторона NP-сложности интервальных вычислений: интервальная эвристика в применении к NP-задачам

Б. Трейлор, В. Крейнович

Известно, что интервальные вычисления NP-сложны. Другими словами, решение многих важных задач может быть сведено к интервальным вычислениям. Первое очевидное следствие этого факта негативно: в общем случае мы не можем построить алгоритм, который выполнял бы все интервальные вычисления быстрее, чем за экспоненциальное время.

Нами показано, что это свойство имеет и свою выгодную сторону: поскольку для интервальных вычислений существует много эвристик, другие задачи могут быть решены сведением их к интервальным вычислениям с дальнейшим применением этих эвристик.

1. Introduction

Before we start talking about the bright side, let us recall what the problem is, what NP-hardness means, and what exactly problem is NP-hard.

1.1. Computing optimal interval estimates is one of the main problems of interval computations

One of the main problems of interval computations is as follows:

Problem 1.

Given:

- an algorithm $f(x_1, \dots, x_n)$ that takes n real numbers and transforms them into a real number;

- n intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

To compute: The range $f(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \{f(x_1, \ldots, x_n) | x_1 \in \mathbf{x}_1, \ldots, x_n \in \mathbf{x}_n\}$ of the function f for $x_i \in \mathbf{x}_i$.

Comment. Usual estimates of interval computations (see, e.g., [14]) do not give the exact range; they give an interval that contains that range. The range itself is called an *optimal* interval estimate [6, 16, 17].

1.2. What does NP-hard mean?

The fact that a problem $\mathcal{P}$ is NP-hard means the following (see, e.g., [4]): If there exists an algorithm that solves all the instances of the problem $\mathcal{P}$ in polynomial time (i.e., whose running time does not exceed some polynomial of the input length), then the polynomial-time algorithm would exist for practically all discrete problems (such as propositional satisfiability problem, discrete optimization problems, etc), and it is a common belief that for at least some of these discrete problems no polynomial-time algorithm is possible (this belief is formally described as P $\neq$ NP). So, the fact that the problem is NP-hard means that no matter what algorithm we use, there will always be some cases for which the running time grows faster than any polynomial, and therefore, for these cases the problem is intractable. In other words: no practical algorithm is possible that would always compute optimal interval estimates.

Theorem [3]. *For polynomial f, the problem of computing optimal interval estimates is NP-hard.*

Comments.

1. Several other problems of interval computations have been proved to be NP-hard, in particular, the problem of finding a solution of an interval linear system [10, 11, 15, 18].
2. The fact that the problem is intractable does not mean that we have to give up: it is well known that many particular cases of NP-hard problems can be solved by polynomial-time algorithms [4]. In particular, many heuristics exist for interval computations (see, e.g., [14]).

The dark side of NP-hardness is what we have already mentioned: one cannot expect an algorithm to do all the interval computations in less than exponential running time.

The bright side of NP-hardness: In this paper, we show that this result also has a bright side: since there are many heuristics for interval computations, we can solve other problems by reducing them to interval computations and applying these heuristics.

2. Main idea

2.1. How are NP-problems reduced to interval computation problems?

Before we start exploiting the reduction to interval computation problems, let us first describe how this reduction is done, i.e., how Gaganov's theorem is proved. We will present a slightly modified version of his proof.

To prove that the problem of computing the range of a polynomial is NP-hard, we will prove that if it were possible to solve it in polynomial time, then it would be possible to solve in polynomial time a problem that is already known to be NP-hard: the so-called satisfiability problem for 3–CNF (see, e.g., [4]). Let us describe this problem in formal terms.

Definition 1.

- *Let an integer $n \geq 1$ be given, and let the finite set $V = \{v_1, \ldots, v_n\}$ with n elements be given. Elements of this finite set V will be called Boolean variables.*
- *By a literal, we mean either a variable v_i, or an expression $\bar{v}_i$ that is called the negation of the variable v_i.*
- *By a 3-disjunction D, we mean an expression of the type $a \vee b$, or $a \vee b \vee c$, where a, b, and c are literals, and corresponding variables are different.*
- *By a propositional formula in 3–CNF (3-conjunctive normal form), we mean an expression of the type $D_1 \& \cdots \& D_d$, where each D_j is a 3-disjunction.*

Comments.

1. In our definition, we excluded disjunctions that contains literals originating from the same variable. The reason for this exclusion is that such disjunctions can be easily handled:
   - the disjunction $v_i \vee v_i \vee \cdots$ is equivalent to $v_i \vee \cdots$;
   - the disjunction $\bar{v}_i \vee \bar{v}_i \vee \cdots$ is equivalent to $\bar{v}_i \vee \cdots$;
   - the disjunction $D = v_i \vee \bar{v}_i \vee \cdots$ is always true and therefore, a formula $F = D \& D_1 \& \cdots$ that contains such a disjunction is equivalent to $D_1 \& \cdots$

   In all three cases, we can easily reduce or eliminate such disjunctions. Therefore, it makes sense to assume that such disjunctions are already excluded.
2. We restricted ourselves to propositional formulas in a 3–CNF from, i.e., to formulas with no more than 3 literals in a disjunction. This restriction is the simplest from which satisfiability problem is still NP-hard: e.g., there exists a polynomial-time algorithm that solves satisfiability problem for so called 2–CNF formulas, i.e., formulas in which every disjunction contains at most 2 literals (see, e.g., [4]).

Definition 2. *By an n-dimensional Boolean vector, or a Boolean array t, we mean a function from the set V into the set of truth values $\{0, 1\}$ (where 1 stands for "true", and 0 stands for "false"). The value $t(v_i)$ is called a truth value of the variable v_i.*

Definition 3. *Let t be a Boolean vector. Then:*

- *The truth value of a literal $\bar{v}_i$ is defined as $\neg t(v_i)$.*
- *The truth value of a disjunction $D = a \vee \cdots \vee c$ is defined as $t(D) = t(a) \vee \cdots \vee t(c)$.*
- *The truth value of a 3–CNF formula $F = D_1 \& \cdots \& D_d$ is defined as $t(F) = t(D_1) \& \cdots \& t(D_d)$.*

If $t(F) = 1$, we say that the Boolean vector t satisfies the formula f.

Definition 4. *We say that a 3–CNF formula is satisfiable if there exists a Boolean vector that satisfies it.*

We can now formulate *satisfiability problem for 3–CNF formulas*:

Problem 2.

Given: a 3−CNF formula F.

To check: whether a formula F is satisfiable, and, if it is satisfiable, *to find* the Boolean vector t that satisfies F.

Comments.

1. This problem is known to be NP-hard.

2. The interval computation problem is reduced to satisfiability as follows:

Definition 5. *Assume that an integer $n \geq 1$ is given. Let us take n real variables $x_1, \ldots, x_n$, and for every variable v_i, literal $\bar{v}_i$, disjunction D, or 3−CNF formula F, let us define a polynomial $M(v_i)$, $M(\bar{v}_i)$, $M(D)$, or $M(F)$ with n variables x_i as follows:*

- $M(v_i) = x_i$;
- $M(\bar{v}_i) = 1 - x_i$;
- *For $D = a \vee \cdots \vee c$, we define $M(D) = M(a) \times \cdots \times M(c)$;*
- *For $F = D_1 \& \cdots \& D_d$, we define $M(F) = M(D_1) + \cdots + M(D_d)$.*

Example. Let's take $F = (v_1 \vee v_2)\&(v_1 \vee \bar{v}_2)$. Here, $n = 2$, $k = 2$, $D_1 = v_1 \vee v_2$, and $D_2 = v_1 \vee \bar{v}_2$. We have $M(v_i) = x_i$, $M(\bar{v}_2) = 1 - x_2$, $M(D_1) = x_1 x_2$, $M(D_2) = x_1(1 - x_2)$, and $M(F) = x_1 x_2 + x_1(1 - x_2)$.

Proposition 1 [3]. *A 3−CNF formula F is satisfiable iff the range $f([0,1], \ldots, [0,1])$ of the polynomial $f = M(F)$ contains 0.*

This proposition, in its turn, is based on the following statement:

Proposition 2. *A Boolean vector t satisfies a 3−CNF formula F iff $f(x_1, \ldots, x_n) = 0$ for $x_i = 1 - t(v_i)$.*

Comment. For reader's convenience, all the proofs are placed in the last section.

2.2. Application of interval heuristics to NP-problems: the main idea

We have just shown how an NP-problem (namely, propositional satisfiability problem for 3−CNF formulas) can be reduced to the problem of computing the range of a polynomial $f(x_1, \ldots, x_n)$ for $x_i \in [0,1]$. So, if we have a heuristic method of solving interval problems, we can apply this method to $M(F)$ and thus get a method of solving satisfiability problem. In the following sections, we will describe several heuristic algorithms based on this idea.

The importance of solving satisfiability problem for 3−CNF formulas follows from the fact that this problem is NP-hard and therefore, many other discrete problems can be reduced to it in the sense that the task of solving these other problem can be reduced to solving several one or several satisfiability problems for 3−CNF formulas (for a description of such reductions, see. e.g., [4]). Therefore, if we have a good algorithm that solves the satisfiability problems for reasonably many 3−CNF formulas, then we automatically get a tool to solve other problems as well.

3. An algorithm based on the simplest possible interval computations heuristic

3.1. Description of the simplest possible interval computation heuristic

In practice, how do people estimate the range of a function $f(x_1, \ldots, x_n)$? This problem often occurs in practice in so-called *indirect measurements*. Suppose that we are interested in the value of some physical quantity y that is difficult or impossible to measure directly. Instead of measuring y, we measure several other quantities x_i, and use the known relationship between x_i and y (i.e., known algorithm f such that $y = f(x_1, \ldots, x_n)$) to transform the results $\tilde{x}_i$ of measuring x_i into an estimate $\tilde{y} = f(\tilde{x}_1, \ldots, \tilde{x}_n)$.

In many cases, the only thing that we know about the measuring devices that we use to measure x_i is their accuracy. In other words, we know the values Δ_i such that the difference $\Delta x_i = \tilde{x}_i - x_i$ between the measurement result $\tilde{x}_i$ and the actual value x_i does not exceed Δ_i: $|\Delta x_i| \le \Delta_i$. So, the only thing we know about the actual values x_i is that they belong to an interval $[\tilde{x}_i - \Delta_i, \tilde{x}_i + \Delta_i]$. The problem is: what are the possible values of y? I.e., what is the range of f for $x_i \in [\tilde{x}_i - \Delta_i, \tilde{x}_i + \Delta_i]$?

Since $x_i = \tilde{x}_i - \Delta x_i$, we can reformulate this problem as follows: what are the possible values of $f(\tilde{x}_1 - \Delta x_1, \ldots, \tilde{x}_n - \Delta x_n)$ when $|\Delta x_i| \le \Delta_i$?

For non-linear f, as Gaganov has proved, we have an NP-hard problem. Usually, however, the errors Δx_i are relatively small. So, we can expand f into a Taylor series, and neglect the terms that are quadratic or of higher order. As a result, we get the following approximate expression: $f(\tilde{x}_1 - \Delta x_1, \ldots, \tilde{x}_n - \Delta x_n) \approx \tilde{y} - f_{,1}\Delta x_1 - \cdots - f_{,n}\Delta x_n$, where we denoted $\tilde{y} = f(\tilde{x}_1, \ldots, \tilde{x}_n)$ and

$$f_{,i} = \frac{\partial f}{\partial x_i}(\tilde{x}_1, \ldots, \tilde{x}_n).$$

In view of this approximate formula, we can estimate the range of the values of

$$f(x_1, \ldots, x_n) = f(\tilde{x}_1 - \Delta x_1, \ldots, \tilde{x}_n - \Delta x_n)$$

by finding the biggest and the smallest possible values of the expression $\tilde{y} - f_{,1}\Delta x_1 - \cdots - f_{,n}\Delta x_n$ when $-\Delta_i \le \Delta x_i \le \Delta_i$.

It is easy to show (see, e.g., [9]) that the smallest possible value of this linear function is attained when $\Delta x_i = -\Delta_i \cdot \mathrm{sign}(f_{,i})$, and this smallest value is equal to $\tilde{y} - \sum |f_{,i}|\Delta_i$. Similarly, the largest value is attained when $\Delta x_i = \Delta_i \cdot \mathrm{sign}(f_{,i})$, and it is equal to $\tilde{y} + \sum |f_{,i}|\Delta_i$.

3.2. How to apply this heuristic to satisfiability problem: an idea

To check whether a 3–CNF formula F is satisfiable, we must check whether 0 belongs to the range $f([0,1], \ldots, [0,1])$ of the polynomial $f = M(F)$. According to our construction, for $x_i \in [0,1]$, the polynomial $f = M(F)$ is always non-negative. So, to check whether its range contains 0 or not, it is sufficient to check whether the minimum value of this polynomial is 0 or not. According to the above-described heuristic, the minimum of this polynomial is attained for $x_i = \tilde{x}_i - \Delta_i \cdot \mathrm{sign}(f_{,i})$.

For our function, the interval $[\tilde{x}_i - \Delta_i, \tilde{x}_i + \Delta_i]$ is equal to $[0, 1]$. Therefore, $\tilde{x}_i = \Delta_i = 0.5$. Hence, the coordinates of the minimizing point can be computed very easily:

- if $f_{,i} < 0$, then $x_i = 0.5 + 0.5 = 1$;
- if $f_{,i} > 0$, then $x_i = 0.5 - 0.5 = 0$.

Since these values are equal to 0 or 1, the value of f for these x_i can be computed using Proposition 2: $f = 0$ iff the vector $t(v_i) = 1 - x_i$ satisfies the formula F.

To apply this idea, we must be able to compute $f_{,i}$. This computation can be done as follows:

Definition 6. *Let F be a 3–CNF formula. Then, for every i from 1 to n, and for every $l = 2, 3$, we use the following denotations:*

- *By $N_l^+(v_i)$, we mean the total number of disjunctions of length l that contain a literal v_i.*
- *By $N_l^-(v_i)$, we mean the total number of disjunctions of length l that contain a literal $\bar{v}_i$.*
- *By $N_l(v_i)$, we mean the difference $N_l^+(v_i) - N_l^-(v_i)$.*

Proposition 3. *For an arbitrary 3–CNF formula F, the derivative*

$$f_{,i} = \frac{\partial f}{\partial x_i}(0.5, \ldots, 0.5)$$

of the function $f = M(F)$ is equal to $(1/4)\big(N_3(v_i) + 2N_2(v_i)\big)$.

Corollary. *$f_{,i} > 0$ iff $N_3(v_i) + 2N_2(v_i) > 0$.*

Now, we are ready to describe the resulting algorithm:

3.3. First heuristic algorithm for solving 3-CNF problems

Given: a 3–CNF formula F.

Do:

- Compute $N_2(v_i)$ and $N_3(v_i)$ for all i from 1 to n.
- If $N_3(v_i) + 2N_2(v_i) > 0$, then choose $t(v_i) = 1$, else $t(v_i) = 0$.
- Substitute the resulting values $t(v_i)$ into the formula F.
  - If for this Boolean vector, F is true (i.e., if $t(F) = 1$), then F is satisfiable, and t is its satisfying vector.
  - If for this Boolean vector, F is false (i.e., $t(F) = 0$), then we have not found a satisfying vector (and therefore, if for some practical purposes, we are required to decide whether the formula should be treated as satisfiable or not, we will treat the formula F as practically unsatisfiable).

Comment.

- If $t(F) = 1$, then, of course, the formula F is satisfiable.
- On the other hand, if $t(F) = 0$, it may mean that we have missed the satisfying vector. The reason for that possibility is that satisfiability problem for 3–CNF is NP-hard, so probably only an *exponential-time* algorithm can always find a solution. Our algorithm is *linear-time* (its number of steps is limited by a linear function of the length of n) and therefore, it cannot be always correct.

Example. For $F = (v_1 \vee v_2) \& (v_1 \vee \bar{v}_2)$, we have:

- $N_2^+(v_1) = 2$, $N_2^-(v_1) = 0$, $N_2(v_1) = 2 - 0 = 2$;
- $N_2^+(v_2) = 1$, $N_2^-(v_2) = 1$, $N_2(v_2) = 1 - 1 = 0$;
- $N_3(v_i) = 0$, because there are no disjunctions of length 3.

Here, $N_2(v_1) + 2N_2(v_1) = 4 > 0$, and $N_2(v_2) + 2N_2(v_2) = 0$, so, $t(v_1) = 1$ and $t(v_2) = 0$. If we substitute v_1 = "true" and v_2 = "false" into F, we get $t(F)$ = "true". So, F is satisfiable.

4. Modified partial derivatives estimation and its application to checking satisfiability

4.1. Main idea

The above-described heuristic was based on the possibility to approximate the expression $f(\tilde{x}_1 - \Delta x_1, \ldots, \tilde{x}_n - \Delta x_n)$ by the sum $\tilde{y} - f_{,1}\Delta x_1 - \cdots - f_{,n}\Delta x_n$ of the first order terms in the Taylor expansion of this expression. In other words, it was based on the possibility to neglect quadratic and higher order terms in the expansion

$$f(\tilde{x}_1 - \Delta x_1, \ldots, \tilde{x}_n - \Delta x_n) =$$

$$f(\tilde{x}_1, \ldots, \tilde{x}_n) - \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \Delta x_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j} \Delta x_i \Delta x_j.$$

For functions of one variable, we only have one quadratic term $(1/2)f''(\tilde{x})\Delta x^2$, and from practical viewpoint, it is either negligible or not. For functions of several variables, it can happen that each second order term is negligible, but there are many of them ($\approx n^2$), and so their sum is not negligible any more. In this cases, we cannot neglect all quadratic terms and approximate the function f by the sum of its linear terms. However, if we can still neglect some of the quadratic terms, then we can still have a reasonable minimization procedure. Such a procedure has been proposed by E. Hansen in [5]. In this Section, we briefly describe the main idea of this procedure, and how it can be applied to satisfiability.

We want to find the minimum of a function $f(x_1, \ldots, x_n)$ on a set

$$[\tilde{x}_1 - \Delta_1, \tilde{x}_1 + \Delta_1] \times \cdots \times [\tilde{x}_n - \Delta_n, \tilde{x}_n + \Delta_n].$$

Instead of immediately trying to minimize over all variables, let us first pick one variable x_i, and try to minimize over that particular variable. In this case,

$$f(x_1,\ldots,x_{i-1},\tilde{x}_i-\Delta x_i,x_{i+1},\ldots,x_n)=$$

$$f(x_1,\ldots,x_{i-1},\tilde{x}_i,x_{i+1},\ldots,x_n)-\frac{\partial f}{\partial x_i}(x_1,\ldots,x_{i+1},\tilde{x}_i,x_{i+1},\ldots,x_n)\Delta x_i+\frac{1}{2}\frac{\partial^2 f}{\partial x_i^2}\Delta x_i^2+\cdots$$

Expanding the derivative $\partial f/\partial x_i$ in Taylor series, we get

$$f(x_1,\ldots,x_{i-1},\tilde{x}_i-\Delta x_i,x_{i+1},\ldots,x_n)=$$

$$f(x_1,\ldots,x_{i-1},\tilde{x}_i,x_{i+1},\ldots,x_n)-\frac{\partial f}{\partial x_i}(\tilde{x}_1,\ldots,\tilde{x}_i,\ldots,\tilde{x}_n)\Delta x_i+\frac{1}{2}\sum_{j=1}^{n}\frac{\partial^2 f}{\partial x_i\partial x_j}\Delta x_i\Delta x_j+\cdots$$

So, if we neglect only n quadratic terms (as opposed to n^2 in the simple method, described in the previous Section), we can conclude that

$$f(x_1,\ldots,x_{i-1},\tilde{x}_i-\Delta x_i,x_{i+1},\ldots,x_n)\approx$$

$$f(x_1,\ldots,x_{i-1},\tilde{x}_i,x_{i+1},\ldots,x_n)-\frac{\partial f}{\partial x_i}(\tilde{x}_1,\ldots,\tilde{x}_i,\ldots,\tilde{x}_n)\Delta x_i$$

and therefore, that the minimum of this function is attained when $x_i=\tilde{x}_i-\Delta_i\cdot\mathrm{sign}(f_{,i})$.

We have only found one coordinate of a point in which the minimum is attained. To get other coordinates, we must:

- substitute the value x_i into our function, thus getting the function of $n-1$ variables;
- apply a similar technique to the resulting function of $n-1$ variables, thus finding one more coordinate of the desired minimum point, etc.

The only remaining question is: which variable x_i should we choose? In general, we can neglect quadratic terms if the corresponding linear term is large enough. So, to be on the safe side, we will choose the variable x_i for which the linear term $|f_{,i}|\Delta_i$ is the largest.

4.2. Description of an algorithm

Let us now describe the resulting algorithm:
Given:

- a function f of n real variables;
- n real numbers $\tilde{x}_i$, $1\le i\le n$, and
- n positive real numbers Δ_i.

To estimate: the interval $[y^-,y^+]$ of possible values of f for $x_i\in[\tilde{x}_i-\Delta_i,\tilde{x}_i+\Delta_i]$.
Algorithm: we will explain this algorithm on the example of y^-.

- First, we compute $\tilde{y}=f(\tilde{x}_1,\ldots,\tilde{x}_n)$ and the values $f_{,i}$ of the partial derivatives in the point $(\tilde{x}_1,\ldots,\tilde{x}_n)$.

- Then, we choose i for which $|f_{,i}|\Delta_i$ is the biggest possible.
- For this i only, we take $x_i^{\text{new}} = \tilde{x}_i - \Delta_i \cdot \text{sign}(f_{,i})$. Then, we substitute $x_i = x_i^{\text{new}}$ into f. As a result, we have a new function $f_{\text{new}}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, x_i^{\text{new}}, x_{i+1}, \ldots, x_n)$ with $n-1$ variables.
- For this new function, we repeat the same procedure (i.e., reduce it to a function of $n-2$, $n-3, \ldots$ variables), until we get a constant f.

This constant is the desired estimate for a minimum.

Example. Let's show that this method can lead to better results than the method described in the previous section. Indeed, let us take $f(x_1, x_2) = x_1^2 + x_1 x_2$, $\tilde{x}_1 = \tilde{x}_2 = 1$, $\Delta_1 = \Delta_2 = 0.2$. For this problem, we know the exact value of the lower bound of f: since f is monotonic in both x_1 and x_2, the lower bound is attained when both x_1 and x_2 take the smallest possible values 0.8. For $x_1 = x_2 = 0.8$, $f(x_1, x_2) = 1.28$.

- The method from the previous section leads to the following estimate: $\tilde{y} = 1^2 + 1 \cdot 1 = 2$, $\partial f/\partial x_1 = 2x_1 + x_2$, $\partial f/\partial x_2 = x_1$, hence $f_{,1} = 3$, $f_{,2} = 1$. As a result, we have $\tilde{y} - \sum |f_{,i}|\Delta_i = 2 - 3 \cdot 0.2 - 1 \cdot 0.2 = 1.2$.
- Let us now apply the above-described Hansen's algorithm. Since $|f_{,1}|\Delta_1 = 0.6 > |f_{,2}|\Delta_2 = 0.2$, we first choose $x_1 = 1 - 0.2 = 0.8$. After that, we have a function of one variable $f_{\text{new}}(x_2) = 0.8^2 + 0.8 \cdot x_2$. For this function, $\tilde{y} = 0.8^2 + 0.8 \cdot 1 \doteq 1.44$; the derivative is 0.8, therefore, $f_{,2} = 0.8$, and the resulting estimate is $\tilde{y} - |f_{,2}|\Delta_2 = 1.44 - 0.16 = 1.28$.

On this example, we can see that Hansen's algorithm takes more computation time, but leads (at least sometimes) to a better estimate.

4.3. Application to satisfiability

Comment. To apply Hansen's algorithm to 3−CNF, we have to compare the values $|f_{,i}|\Delta_i$. In our case, $\Delta_i = 0.5$ does not depend on i, therefore, it is sufficient to compare the values $|f_{,i}|$. Then, we must fix the value of the variable x_i for which this value is the largest. This is equivalent to fixing the truth value of the corresponding Boolean variable v_i. Then, we repeat the procedure again, with the new formula F_{new}, etc. So, we arrive at the following algorithm:

Given: a 3−CNF formula F.

Do:

- Compute $N_2(v_i)$ and $N_3(v_i)$ for all i from 1 to n.
- Choose i for which $|N_3(v_i) + 2N_2(v_i)|$ is the largest.
- For this i, take $t(v_i) = 1$ if $N_3(v_i) + 2N_2(v_i) > 0$, and 0 else.
- Substitute this truth value v_i into F. As a result, we get a new 3−CNF formula F_{new} with $n-1$ Boolean variables. Apply the same procedure to F_{new} until we end up with a constant. If this constant is 1 ("true"), then the original formula F is satisfiable. If this constant is 0 ("false"), then we have not found a satisfying vector (and therefore, if for some practical purpises, we are required to decide whether the formula should be treated as satisfiable or not, we will treat the formula F as practically unsatisfiable).

Comment. It can happen that after we substitute the truth value into a formula F, one of the disjunctions will be left with only one literal. These cases are easily dealt with: if $D = a$ is a disjunction of F, then F can only be true when a is true, so, we automatically have a truth value of one more variable. This procedure can be repeated until we get a 3−CNF formula (i.e., a formula in which each disjunction has 2 or 3 literals in it).

Example. For $F = (v_1 \vee v_2)\&(v_1 \vee \bar{v}_2)$, we have:

- $|N_2(v_1) + 2N_2(v_1)| = 4$, and
- $|N_2(v_2) + 2N_2(v_2)| = 0$.

Therefore, the largest value is attained for $i = 1$. For this i, $N_2(v_1) + 2N_2(v_1) > 0$, and therefore, we take $t(v_1) = 1$. The resulting new formula with one Boolean variable v_2 is identically true, so we can choose an arbitrary value of v_2.

4.4. These algorithms are reasonably efficient

The algorithms described in these two sections are similar to the ones proposed first by S. Maslov [12] (see also [2, 7, 8, 13]). The only difference is that Maslov and others use different weights ($\neq 2$) for expressions with only two literals, and they use different heuristics to justify this method. Experimental [12, 13] and theoretical [2, 7, 8] considerations prove that such methods are very efficient for randomly chosen propositional formulas.

We also performed our own experiments (with our weight = 2), and these experiments show that this choice of the weight is not worse than the previous ones.

5. Application of naive interval computations to satisfiability

5.1. Main idea

In order to solve satisfiability problem for a 3−CNF formula F, we must be able to estimate the range of a polynomial $f = M(F)$: if this range contains 0, then the formula is satisfiable.

We showed that the application of the simplest heuristic from interval computations leads to an interesting and non-trivial algorithm for solving satisfiability problems. This makes us believe that applications of other heuristics can lead to even better algorithms for satisfiability (and/or other NP-hard problems).

Let us first try naive interval computations to estimate the range of the polynomial f. After we get an estimate $[y^-, y^+]$, there are two options:

- The lower bound y^- of this estimate is > 0. Since interval computations lead to a guaranteed estimate for the function's range, this means that 0 is not in the range of f and thus, the formula F is not satisfiable.
- $y^- < 0$. In this case, it is possible that 0 belongs to the range of the function f, and thus, it is possible that the formula F is satisfiable. How to find the satisfying vector? Let's find it coordinate after coordinate. To find out whether we can take v_i = "true", we can do the following:

- substitute this value into the formula F;
- find a polynomial $M(F_{\text{new}})$ that corresponds to the new formula (i.e., to the result of this substitution);
- apply naive interval computation to estimate the range of this polynomial.

We can apply the same procedure for v_i = "false".

- If for both cases (v_i = "true" and v_i = "false") the resulting ranges do not contain 0, then the formula F cannot have satisfying vectors neither with v_i = "true", nor with v_i = "false". Thus, the formula F is not satisfiable.
- If in one case (say, for v_i = "true"), the estimate for the range contains 0, and for the other case, it does not contain 0, then we know that v_i must be equal to "true" if we want F to be satisfied. So, we substitute this value of v_i into F, and get a new formula with $n-1$ variables.
- If both estimates for the range contain 0, then it is possible that a satisfying vector will be found in both cases. The lower the estimated y^-, the more reasonable it is to believe that the actual lower endpoint of the range is 0. Therefore, we choose $t(v_i)=1$ or $t(v_i)=0$ depending on which estimate is smaller. As a result, we also get a formula with $n-1$ variables.

Now, we apply the same procedure to the resulting formula with $n-1$ variables, etc.

What i should we choose? In making this choice, we can use the same argument that we used when we decided whether to choose $v_i=1$ or $v_i=0$: we choose i for which the lower bound y^- of the interval estimate is the smallest (and for which, therefore, it is the most reasonable to expect the actual lower bound of the range to be the smallest).

To apply this idea, we must describe the result of applying naive interval computation to $f=M(F)$.

Definition 7. *Let D be a disjunction and v_i be the variable. Let us define the notion of occurrence and a function s as follows:*

- *We say that a variable v_i occurs positively in D if D contains v_i; we will denote it by $s(v_i,D)=+1$.*
- *We say that a variable v_i occurs negatively in D if D contains $\bar{v}_i$; we will denote it by $s(v_i,D)=-1$.*
- *We say that a variable v_i does not occur in D if D contains neither v_i, nor $\bar{v}_i$; we will denote it by $s(v_i,D)=0$.*
- *We say that variables $v_i,\ldots,v_j$ occur positively in D if $s(v_i,D)\times\cdots\times s(v_j,D)=+1$.*
- *We say that variables $v_i,\ldots,v_j$ occur negatively in D if $s(v_i,D)\times\cdots\times s(v_j,D)=-1$.*

Example. Variables v_1 and v_2 occur positively in $v_1\vee v_2\vee v_3$ and $\bar{v}_1\vee\bar{v}_2\vee v_4$. The same set of variables occurs negatively in $v_1\vee\bar{v}_2\vee\bar{v}_4$.

Definition 8. *Let F be a 3−CNF formula. Then, for every i, j, k from 1 to n, and for every $l=2,3$, we use the following denotations:*

- *By N_l, we mean the total number of disjunctions of length l.*
- *By $N_l^+(v_i, \ldots, v_j)$, we mean the total number of disjunctions of length l in which the variables $v_i, \ldots, v_j$ occur positively.*
- *By $N_l^-(v_i, \ldots, v_j)$, we mean the total number of disjunctions of length l in which the variables $v_i, \ldots, v_j$ occur negatively.*
- *By $N_l(v_i, \ldots, v_j)$, we mean the difference $N_l^+(v_i, \ldots, v_j) - N_l^-(v_i, \ldots, v_j)$.*

Proposition 4. *For an arbitrary 3−CNF formula F, we have*

$$\begin{aligned} M(F) \;=\; & \tfrac{1}{4}N_2 + \tfrac{1}{2}\sum_{i=1}^{n} N_2(v_i)\Delta x_i + \sum_{i<j} N_2(v_i, v_j)\Delta x_i \Delta x_j + \\ & \tfrac{1}{8}N_3 + \tfrac{1}{4}\sum_{i=1}^{n} N_3(v_i)\Delta x_i + \tfrac{1}{2}\sum_{i<j} N_3(v_i, v_j)\Delta x_i \Delta x_j + \\ & \sum_{i<j<k} N_3(v_i, v_j, v_k)\Delta x_i \Delta x_j \Delta x_k \end{aligned} \tag{1}$$

where $\Delta x_i = x_i - 0.5$.

Here, $\Delta x_i \in [-0.5, 0.5]$, so, the result of applying naive interval computation to this formula is as follows:

Corollary. *The result of applying naive interval computation to the formula (1) is $[y^-, y^+]$, where*

$$\begin{aligned} y^- \;=\; & \tfrac{1}{4}N_2 - \tfrac{1}{4}\sum_{i=1}^{n}|N_2(v_i)| - \tfrac{1}{4}\sum_{i<j}|N_2(v_i, v_j)| + \\ & \tfrac{1}{8}N_3 - \tfrac{1}{8}\sum_{i=1}^{n}|N_3(v_i)| - \tfrac{1}{8}\sum_{i<j}^{n}|N_3(v_i, v_j)| - \tfrac{1}{8}\sum_{i<j<k}|N_3(v_i, v_j, v_k)|. \end{aligned} \tag{2}$$

5.2. Resulting algorithm

Given: a 3−CNF formula F.
Do:

- Compute expression (2). If it is > 0, then the formula F is not satisfiable. If it is ≤ 0, then do the following:
- For each i from 1 to n, and for each $t = 0, 1$, do the following:
  - substitute $t(v_i) = t$ into the formula F;
  - compute the value (2) for the resulting formula $F_{|v_i=t}$;

  and then choose i and t for which the resulting lower bound y^- is the smallest.
- Choose $F_{\text{new}} = F_{|v_i=t}$ for the selected i and t as a new formula with $n-1$ Boolean variables. Apply the same procedure to F_{new} until we end up with a constant. If this constant is 1 ("true"), then the original formula F is satisfiable. If this constant is 0 ("false"), then we have not found a satisfying vector (and therefore, if for some practical purposes, we are required to decide whether the formula should be treated as satisfiable or not, we will treat the formula F as practically unsatisfiable).

Example. For $F = (v_1 \vee v_2) \& (v_1 \vee \bar{v}_2)$, we have:

- $N_2 = 2$, $N_3 = 0$;
- $N_2^+(v_1) = 2$, $N_2^-(v_1) = 0$, $N_2(v_1) = 2 - 0 = 2$;
- $N_2^+(v_2) = 1$, $N_2^-(v_2) = 1$, $N_2(v_2) = 1 - 1 = 0$;
- $N_2^+(v_1, v_2) = 1$, $N_2^-(v_1, v_2) = 1$, $N_2(v_1, v_2) = 1 - 1 = 0$;
- $N_3(v_i, \ldots, v_j) = 0$, because there are no disjunctions of length 3.

As a result, the formula (2) gives $(1/4) \cdot 2 - (1/4) \cdot 2 - (1/4) \cdot 0 = 0$. Since the range estimate contains 0, we continue by trying to reduce our problem to the formula with $n - 1$ variables. According to the algorithm, we have to pick i and t. Here, we have 4 options: $i = 1, 2$, and $t = 0, 1$:

- For $i = 1$ and $t = 0$, substituting $v_1 = 0$ makes the formula identically false.
- For $i = 1$ and $t = 1$, substituting $v_1 = 1$ makes the formula identically true.

So, we are done (and we do not have to continue with $i = 2$). We can now add any value of v_2 and get a satisfying vector.

Comments.

1. Our (preliminary) experiments with these formulas did not show convincing average speed-up or any other advantage over algorithms presented in Sections 3 and 4. However, some formulas that were not handled by those algorithms are now handled. So, we hope that this new algorithm is also useful for solving NP-problems.

2. Instead of using naive interval computations, we can apply more sophisticated interval estimation techniques. For example, we can apply a mean-valued form, in which the range of a function f is approximated as

$$[y^-, y^+] = f(\tilde{x}_1, \ldots, \tilde{x}_n) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}\Big([\tilde{x}_1 - \Delta_1, \tilde{x}_1 + \Delta_1], \ldots, [\tilde{x}_n - \Delta_n, \tilde{x}_n + \Delta_n]\Big)[-\Delta_i, \Delta_i].$$

   It turns out, however, that for $f = M(F)$, this form leads to worse range estimates than naive interval computations (see Section 7).

3. We also hope that other, more sophisticated techniques, can help. Polynomials that we are building have a special structure: each of them is a sum of monomials, and each monomial is a multi-linear function of total degree ≤ 1 (i.e., each monomial can only contain the first degree of each variable). This specific structure can be used to design specific (and more efficient) range estimation algorithms.

4. The main ideas of this section were proposed by one of the referees, to whom we are most grateful.

6. Bistromathics: a science-fiction case when interval computations help to solve general problems

It is interesting to mention that the idea of using interval computations to solving generic complicated computational problems has already been proposed in science fiction. Namely, in Chapter 5 of [1], a new mathemathics called *bistromathics* is described as a tool to solve such problems. Bistromathics is based on the fact that "numbers written on restaurant checks... do not follow the same mathematical laws as numbers written on any other piece of paper in any other parts of the Universe." So, automata that simulate waiters in the restaurants turn out to be useful in solving complicated problems.

To figure out how bistromathics is related to interval computations, let us recall why and how it is possible to cheat on a restaurant check (and cheating, although not as universal as [1] claims, does happen once in a while). The correct sum on the restaurant check can be obtained by adding, subtracting, and multiplying several numbers, such as the cost of a meal, the restaurant tax, the current exchange rate for a certain currency (if the payment can be done in several different currencies), the discounts that have been promised by this restaurant, the percentage of tips, etc. As a result, the correct sum is a polynomial of several variables $f(x_1, \ldots, x_n)$. The possibility to cheat is based on the fact that customers usually do not remember the exact values of these variables (the exact cost of the meals, the exact exchange rates, etc). At best, they remember the *intervals* $[x_i^-, x_i^+]$ of possible values of these variables. The waiter must announce the value that is *convincing* to the customer, i.e., that belongs to the range $[y^-, y^+] = f([x_1^-, x_1^+], \ldots, [x_n^-, x_n^+])$. His goal is to get as much money from the customer as possible. Therefore, his ideal solution is to request the biggest convincing total, i.e., the *upper bound* y^+ of the range. So, the "perfect" waiter (perfect in the above sense: to cheat as much as possible without being caught) must be able to compute the exact interval range for a polynomial.

So, in view of Gaganov's theorem, this "perfect" waiter will be able to solve an NP-hard problem. By definition of NP-hardness it means that, using this "perfect" waiter as a part of our computations, we will be able to solve an arbitrary complicated discrete problem. This is exactly what bistromathics is about.

7. Proofs

Proof of Propositions 1 and 2. Let us prove that a formula F is satisfiable iff the range of the polynomial $f = M(F)$ for $x_i \in [0,1]$ contains 0. In the course of proving it, we will also prove Proposition 2.

$\rightarrow$ If the formula is true for some values v_i, then for every i, take $x_i = 0$ if $v_i =$ "true" and $x_i = 1$ if $v_i =$ "false". As a result, a literal a is true iff $M(a) = 0$.

Since $F = D_1 \& D_2 \& \cdots \& D_d$ is true, it means that all the expressions D_j are true. So, for every $D_j = a \vee \cdots \vee c$, there exists a literal that is true. For this literal a, we have $M(a) = 0$. Hence, $M(D_j) = M(a) \times \cdots \times M(c) = 0$ for every expression D_j. Hence,

$$M(F) = M(D_1) + M(D_2) + \cdots + M(D_d) = 0 + \cdots + 0 = 0.$$

Thence, 0 belongs to the desired range.

$\leftarrow$ Assume that 0 belongs to the range. This means that $M(F) = M(D_1)+\cdots+M(D_d) = 0$ for some x_i. When $x_i \in [0,1]$, we have $1 - x_i \geq 0$, hence $M(D_j) \geq 0$. The only case when the sum of k non-negative numbers $M(D_j)$ is equal to 0 is when each of them is equal to 0. So, $M(D_j) = 0$ for all j.

By definition of M, $M(D_j) = M(a\vee\cdots\vee c) = M(a)\times\cdots\times M(c)$. So, from $M(D_j) = 0$, it follows that $M(a) = 0$ for one of the literals from D_j. If $a = v_i$, this means that $x_i = 0$. If $a = \bar{v}_i$, this means that $1 - x_i = 0$, and $x_i = 1$.

Let us take v_i ="true" iff $x_i = 0$, and let us show that these values make F true. Indeed, for every j, since $M(D_j) = 0$, we have $M(a) = 0$ for one of the literals from D_j. If $a = v_i$, this means that $x_i = 0$, hence v_i = "true", and D_j is true. If $a = \bar{v}_i$, then $x_i = 1$, so v_i = "false", $\bar{v}_i$ = "true", and D_j is true. So, in both cases, D_j is true. Hence, all expressions D_j are true, so $F = D_1\&D_2\&\cdots\&D_d$ is also true. □

Proof of Proposition 3. By definition of $f = M(F)$, the function f can be represented as $f = f_1 + \cdots + f_d$ for $f_j = M(D_j)$. Therefore,

$$f_{,i} = \sum_{j=1}^{d} \frac{\partial f_j}{\partial x_i}(\tilde{x}_1,\ldots,\tilde{x}_n).$$

Let's calculate the derivatives of f_j. We will consider all possible cases:

- If D_j does not contain v_i, then f_j does not depend on x_i at all, so, $\partial f_j/\partial x_i = 0$.
- If $D_j = v_i \vee b \vee c$, then $f_j = x_i M(b)M(c)$, hence $\partial f_j/\partial x_i = M(b)M(c) = (1/2)^2 = 1/4$.
- If $D_j = \bar{v}_i \vee b \vee c$, then $f_j = (1-x_i)M(b)M(c)$, hence $\partial f_j/\partial x_i = -M(b)M(c) = (1/2)^2 = -1/4$.
- If $D_j = v_i \vee b$, then $f_j = x_i M(b)$, hence $\partial f_j/\partial x_i = M(b) = 1/2$.
- If $D_j = \bar{v}_i \vee b$, then $f_j = (1 - x_i)M(b)$, hence $\partial f_j/\partial x_i = -M(b) = -1/2$.

Adding all these expressions, we get the desired formula

$$f_{,i} = \tfrac{1}{4}\Big(N_3^+(v_i) + 2N_2^+(v_i) - N_3^-(v_i) - 2N_2^-(v_i)\Big).$$

□

Proof of Proposition 4. The function $M(F)$ is a sum of the terms

$$M(D) = M(a)\times\cdots\times M(b)$$

where $M(v_i) = (1/2) + \Delta x_i$ and $M(\bar{x}_i) = (1/2) - \Delta x_i$. Therefore, if a corresponds to v_i or $\bar{v}_i$, and b is v_j or $\bar{v}_j$, then

$$\begin{aligned} M(a\vee b) &= \Big((1/2) + (-1)^{s(v_i,D)}\Delta x_i\Big)\Big((1/2) + (-1)^{s(v_j,D)}\Delta x_j\Big) \\ &= (1/4) \pm (1/2)\Delta x_i \pm (1/2)\Delta x_j \pm \Delta x_i \Delta x_j \end{aligned}$$

where signs depend on whether D contains a variable or its negation. Adding up these equalities and similar equalities for disjunctions of length 3, we get the desired formula. □

Proof of Comment 2 from Section 5. By differentiating formula (1), we get the formula for the i-th partial derivative of $f = M(F)$:

$$\begin{aligned}\frac{\partial f}{\partial x_i} &= \tfrac{1}{2}N_2(v_i) + \sum_{j=1}^{n} N_2(v_i, v_j)\Delta x_j + \\ &\tfrac{1}{4}N_3(v_i) + \tfrac{1}{2}\sum_{j=1}^{n} N_3(v_i, v_j)\Delta x_j + \sum_{j<k} N_3(v_i, v_j, v_k)\Delta x_j \Delta x_k.\end{aligned} \tag{3}$$

Therefore, the result of applying naive interval computations to this formula leads to the interval

$$\frac{\partial f}{\partial x_i}\Big([\tilde{x}_1 - \Delta_1, \tilde{x}_1 + \Delta_1], \ldots, [\tilde{x}_n - \Delta_n, \tilde{x}_n + \Delta_n]\Big) =$$
$$\left[\tfrac{1}{2}N_2(v_i) + \tfrac{1}{4}N_3(v_i) - d_i, \tfrac{1}{2}N_2(v_i) + \tfrac{1}{4}N_3(v_i) + d_i\right]$$

where

$$d_i = \tfrac{1}{2}\sum_{j=1}^{n} |N_2(v_i, v_j)| + \tfrac{1}{4}\sum_{j=1}^{n} |N_3(v_i, v_j)| + \tfrac{1}{4}\sum_{j<k} |N_3(v_i, v_j, v_k)|. \tag{4}$$

Substituting these values into the above formula, and applying interval computations with $\Delta_i = 0.5$, we get an expression that is similar to (2), but in which each term $N_2(v_i, v_j)$ occurs twice. As a result, for these polynomials, mean value form leads to worse estimates than naive interval computations. □

8. Acknowledgments

This work was partially sponsored by NSF grant No. CDA−9015006 and NASA grant No. NAG 9−757. One of the authors (V.K.) is thankful to Yu. V. Matiyasevich for valuable discussions. The authors are thankful to J. Rohn, S. Shary, and to anonymous referees for their attention to this work and for valuable suggestions.

References

[1] Adams, D. *Life, the Universe and everything*. Pocket Books, N.Y., 1983.

[2] Dubois, O. *Counting the number of solutions for instances of satisfiability*. Theoretical Computer Science **81** (1991), pp. 49−64.

[3] Gaganov, A. A. *Computational complexity of the range of the polynomial in several variables*. Cybernetics (1985), pp. 418−421.

[4] Garey, M. and Johnson, D. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.

[5] Hansen, E. R. *On solving systems of equations using interval arithmetic*. Math. Comp. (1968), pp. 374−384.

[6] Kolacz, H. *On the optimality of inclusion algorithms*. In: Nickel, K. (ed.) "Interval Mathematics 1985", Lecture Notes in Computer Science **212**, Springer-Verlag, Berlin, Heidelberg, N.Y., 1986, pp. 67−79.

[7] Koutsoupias, E. and Papadimitriou, C. H. *On the greedy algorithm for satisfiability*. Information Processing Letters **43** (1992), pp. 53–55.

[8] Kreinovich, V. *Maslov's increasing freedom of choice strategy: on the example of the satisfiability problem*. Leningrad Center for New Informational Technology "Informatika", Technical Report, 1989 (in Russian).

[9] Kreinovich, V., Bernat, A., Villa, E., and Mariscal, Y. *Parallel computers estimate errors caused by imprecise data*. Interval Computations 2 (1991), pp. 31–46.

[10] Kreinovich, V., Lakeyev, A. V., and Noskov, S. I. *Optimal solution of interval linear systems is intractable (NP-hard)*. Interval Computations 1 (1993), pp. 6–14.

[11] Lakeyev, A. V. and Noskov, S. I. *A description of the set of solutions of a linear equation with interval defined operator and right-hand side*. Russian Acad. Sci. Dokl. Math. **47** (1993), pp. 518–523.

[12] Maslov, S. Yu. and Kurenkov, Yu. N. *The increasing freedom of choice strategy for the satisfiability problem*. In: "Proceedings of the USSR National Conference on Methods of Mathematical Logic in Artificial Intelligence and Programming", Palanga, 1980 (in Russian).

[13] Maslov, S. Yu. *Theory of deductive systems and its applications*. MIT Press, Cambridge, MA, 1987.

[14] Moore, R. E. *Methods and applications of interval analysis*. SIAM, Philadelphia, 1979.

[15] Poljak, S. and Rohn, J. *Checking robust non-singularity is NP-hard*. Mathematics of Control, Signals, and Systems **6** (1993), pp. 1–9.

[16] Rall, L. B. *Optimization of interval computations*. In: Nickel, K. (ed.) "Interval Mathematics 1980", Academic Press, N.Y., 1980, pp. 489–498.

[17] Ratschek, H. and Rokne, J. *Optimality of the centered form*. In: Nickel, K. (ed.) "Interval Mathematics 1980", Academic Press, N.Y., 1980, pp. 499–508.

[18] Rohn, J. and Kreinovich, V. *Computing exact componentwise bounds on solutions of linear systems with interval data is NP-hard*. SIAM Journal on Matrix Analysis and Applications (SIMAX) **16** (2) (1994), pp. 415–420.

Received: June 25, 1993
Revised version: January 3, 1995

B. Traylor
M/S 301–270
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
USA

V. Kreinovich
Computer Science Department
University of Texas at El Paso
El Paso, TX 79968
USA