6-2011

Is It Possible to Have a Feasible Enclosure-Computing Method Which Is Independent of the Equivalent Form?

Marcin Michalak
Silesian University of Technology

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-11-35

Is It Possible to Have a Feasible Enclosure-Computing Method Which Is Independent of the Equivalent Form?*

Marcin Michalak
Silesian University of Technology, ul. Academicka 16, 44-100 Glivice, Poland
marcin.michalak@polsl.pl

Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, 500 W. University, El Paso, TX 79968, USA
vladik@utep.edu

July 6, 2011

Abstract

The problem of computing the range $\boldsymbol{y}$ of a given function $f(x_1, \ldots, x_n)$ over given intervals $\boldsymbol{x}_i$ – often called the main problem of interval computations – is, in general, NP-hard. This means that unless P = NP, it is not possible to have a feasible (= polynomial time) algorithm that always computes the desired range. Instead, interval computations algorithms compute an enclosure $\boldsymbol{Y} \supseteq \boldsymbol{y}$ for the desired range. For all known feasible enclosure-computing methods – starting with straightforward interval computations – there exist two expressions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ for computing the same function that lead to different enclosures. We prove that, unless P = NP, this is inevitable: it is not possible to have a feasible enclosure-computing method which is independent of the equivalent form.

Keywords: interval computations, enclosure, equivalent form, NP-hard
AMS subject classifications: 65G20, 65G40, 03D15, 68Q17

1 Formulation of the Problem

One of the main problems of interval computations. One of the main problems of interval computations has the following form:

*Submitted: June 28, 2011; Revised: ???; Accepted: ???.

- We are given an algorithm $f(x_1, \ldots, x_n)$ for computing a function of n real variables, and n intervals $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.
- We need to compute the range

$$\boldsymbol{y} = [\underline{y}, \overline{y}] = \{f(x_1, \ldots, x_n) : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n\}$$

  of the function $f(x_1, \ldots, x_n)$ under given intervals.

This problem is NP-hard. It is known (see, e.g., [3]) that the above problem is, in general, NP-hard. For example, the problem of checking whether $\underline{y} \geq 0$ or $\underline{y} < 0$ is NP-hard.

Comment. NP-hardness means that unless P = NP, it is not possible to have a feasible (= polynomial time) algorithm that always computes the desired range.

Need to compute enclosures. Since we cannot always efficiently compute the exact range $\mathbf{y}$, and we want to guarantee that the value $f(x_1, \ldots, x_n)$ is contained in the estimated range, we need to find an *enclosure* $\boldsymbol{Y} \supseteq \boldsymbol{y}$.

Feasible methods for computing enclosures. There exist many feasible techniques for computing enclosures: straightforward interval computations, mean value form, methods combined with bisection, etc.; see, e.g., [4].

Existing feasible methods for computing enclosure are not independent on the equivalent form. In straightforward interval computations, we

- represent the algorithm $f(x_1, \ldots, x_n)$ as a sequence of elementary arithmetic operations such as $+$, $-$, $\cdot$, $/$, min, max, etc. and then
- replace each elementary operation with the corresponding operation of interval arithmetic.

In this method, in general, the resulting enclosure depends on the equivalent form. For example, the algorithms $f(x_1) = x_1 - x_1$ and $g(x_1) = 0$ represent the same function 0 on the interval $[0, 1]$, but:

- for $f(x_1) = x_1 - x_1$ straightforward interval computations leads to an enclosure

$$[0, 1] - [0, 1] = [-1, 1],$$

  while
- for $g(x_1) = 0$, we get the enclosure $\mathbf{Y} = [0, 0] \neq [-1, 1]$.

Similarly, all other known feasible methods for computing enclosure are not independent on the equivalent form: for each of these methods, there exist two algorithms that compute the same function but lead to different enclosures.

Comment. It is worth mentioning that for straightforward interval computations, for each algorithm $f(x_1, \ldots, x_n)$, for each set of intervals $[\underline{x}_i, \overline{x}_i]$, and for each interval $\boldsymbol{Y}$ containing the actual range $\boldsymbol{y}$, there exists an algorithm $g(x_1, \ldots, x_n)$ that compute the same function as $f(x_1, \ldots, x_n)$ and for which straightforward interval computations leads to $\boldsymbol{Y}$; see, e.g., [1, 2].

Natural question. The above fact leads to the following natural question: Is it possible to have a feasible enclosure-computing method which is independent of the equivalent form?

2 Main Result

Definition 1. *By a* feasible enclosure-computing method*, we mean a feasible algorithm A that, given an algorithm $f(x_1, \ldots, x_n)$ and n intervals $\boldsymbol{x}_1$, ..., $\boldsymbol{x}_n$, computes an interval $A(f, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ that contains the range*

$$\boldsymbol{y} = \{f(x_1, \ldots, x_n) : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n\}.$$

Comment. This definition includes non-interesting methods – e.g., a method that always returns very wide intervals $\boldsymbol{Y}$.

Definition 2. *We say that a feasible enclosure-computing method A is* non-trivial *if for every constant-valued algorithm $f(x_1, \ldots, x_n) = c = \text{const}$, this method returns the same constant $A(f, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = [c, c]$.*

Proposition. *If $P \neq NP$, then for every non-trivial feasible enclosure-computing method A, there exist two algorithms $g(x_1, \ldots, x_n)$ and $h(x_1, \ldots, x_n)$ and n intervals $\boldsymbol{x}_1$, ..., $\boldsymbol{x}_n$ for which*

- *$g(x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$ for all $x_1 \in \boldsymbol{x}_1$, ..., $x_n \in \boldsymbol{x}_n$, but*
- *$A(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \neq A(h, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.*

Comment. Thus, it is not possible to have a feasible enclosure-computing method which is independent of the equivalent form.

Proof.

1°. We will prove this result by contradiction. Let us assume that there exists a non-trivial feasible enclosure-computing method A for which, if two algorithms $g(x_1, \ldots, x_n)$ and $h(x_1, \ldots, x_n)$ compute the same function on a box $\boldsymbol{x}_1 \times \ldots \times \boldsymbol{x}_n$, then $A(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = A(h, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

Let us show that in this case, we will be able to feasibly check, given an algorithm $f(x_1, \ldots, x_n)$ and n intervals $\boldsymbol{x}_1$, ..., $\boldsymbol{x}_n$, whether the lower endpoint $\underline{y}$ of the range $[\underline{y}, \overline{y}] = \{f(x_1, \ldots, x_n) : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n\}$ is non-negative or negative.

2°. Our main idea is that, based on the algorithm $f(x_1, \ldots, x_n)$, we can compute a new algorithm $g(x_1, \ldots, x_n) = \min(f(x_1, \ldots, x_n), 0)$.

Let us consider two possible cases: $\underline{y} \geq 0$ and $\underline{y} < 0$.

2.1°. If $\underline{y} \geq 0$, this means that $f(x_1, \ldots, x_n) \geq 0$ for all $x_i \in \boldsymbol{x}_i$. Thus, for all values $(x_1, \ldots, x_n)$ from the corresponding box, we have $g(x_1, \ldots, x_n) = \min(f(x_1, \ldots, x_n), 0) = 0$. Hence, in this case, $g(x_1, \ldots, x_n)$ coincides with $h(x_1, \ldots, x_n) \stackrel{\text{def}}{=} 0$ for all the values from the box. So, due to our assumption, $A(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = A(h, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

Since the method A is non-trivial, we have $A(h, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = [0, 0]$, so $A(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = [0, 0]$. Thus, in this case, the lower endpoint $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ of the interval $A(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ is equal to 0.

2.2°. On the other hand, if $\underline{y} < 0$, this means that there exists values $x_i \in \boldsymbol{x}_i$ for which $f(x_1, \ldots, x_n) < 0$. For these values x_i, we have $g(x_1, \ldots, x_n) = \min(f(x_1, \ldots, x_n), 0) = f(x_1, \ldots, x_n) < 0$. Thus, the lower endpoint of the range of the function g on the box $\boldsymbol{x}_1 \times \ldots \times \boldsymbol{x}_n$ is also negative. Since the interval $A(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ is an enclosure for this range, its lower endpoint is also negative: $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) < 0$.

2.3°. Summarizing:

- if $\underline{y} \geq 0$, then $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = 0$;
- if $\underline{y} < 0$, then $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) < 0$.

3°. According to Part 2.3 of this proof, if $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = 0$, then we cannot have $\underline{y} < 0$, because then we would have $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) < 0$. Thus, we have $\underline{y} \geq 0$.

Similarly, if $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) < 0$, then we cannot have $\underline{y} \geq 0$, because then we would have $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = 0$. Thus, we have $\underline{y} < 0$. In other words:

- if $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = 0$, then $\underline{y} \geq 0$;
- if $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) < 0$, then $\underline{y} < 0$.

4°. According to Part 3 of this proof, by applying the method A to the function $g(x_1, \ldots, x_n)$ and the original intervals and by checking whether $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = 0$ or $\underline{A}(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) < 0$, we will be able to detect, in feasible time, whether $\underline{y} \geq 0$ or $\underline{y} < 0$.

However, we know that the problem of detecting whether $\underline{y} \geq 0$ or $\underline{y} < 0$ is NP-hard. Thus, the fact that we can solve this problem in polynomial time means that P = NP – and we assumed that P≠NP.

This contradiction shows that our assumption is wrong, i.e., that for every non-trivial feasible enclosure-computing method A, there exist two algorithms $g(x_1, \ldots, x_n)$ and $h(x_1, \ldots, x_n)$ that compute the same function for given n intervals $\boldsymbol{x}_1$, …, $\boldsymbol{x}_n$ but for which $A(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \neq A(h, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

The proposition is proven.

Acknowledgments. This work was partly supported by the US National Science Foundation grants HRD-0734825 and DUE-0926721, and by Grant 1 T36 GM078000-01 from the US National Institutes of Health.

The authors are thankful to all the participants of the 13th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing RSFDGrC'2011 (Moscow, Russia, June 25–30, 2011) for valuable discussions.

References

[1] Koshelev, M.: *Every Superinterval of the Function Range Can Be An Interval-Computations Enclosure*, The Chinese University of Hong Kong, Department of Mechanical & Automation Engineering, Technical Report CUHK-MAE-99-003, January 1999.

[2] Koshelev, M.: *Every superinterval of the function range can be an interval-computations enclosure*, Reliable Computing 6, pp. 219–223, 2000.

[3] Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.

[4] Moore, R. E., Kearfott, R. B., Cloud, M. J.: *Introduction to Interval Analysis*, SIAM Press, Philadelphia, Pennsylvania, 2009.