When We Know the Number of Local Maxima, Then We Can Compute All of Them

Olga Kosheleva, Martine Ceberio, and Vladik Kreinovich

University of Texas at El Paso, El Paso, TX 79968, USA
olgak@utep.edu, mceberio@utep.edu, vladik@utep.edu

Abstract. In many practical situations, we need to compute local maxima. In general, it is not algorithmically possible, given a computable function, to compute the locations of all its local maxima. We show, however, that if we know the *number* of local maxima, then such an algorithm is already possible. Interestingly, for global maxima, the situation is different: even if we only know the number of locations where the *global* maximum is attained, then, in general, it is not algorithmically possible to find all these locations. A similar impossibility result holds for local maxima if instead of knowing their exact number, we only know two possible numbers.

1 Locating Local Maxima: an Important Practical Problem

Need for computing local maxima. In many practical situations, we are interested in locating all local optima; see, e.g., [3]. For example:

- in spectral analysis, chemical species are identified by local maxima of the spectrum;
- in radioastronomy, radiosources and their components are identified as local maxima of the brightness distribution; see, e.g., [4];
- elementary particles are identified by locating local maxima of the dependence of scattering intensity on the energy.

In general, no algorithm is possible for computing all local maxima. In general, no algorithm is possible for computing all local maxima of a computable function $f(x)$; this follows, e.g., from our negative result formulated below.

Natural question and what we do in this paper. Since we cannot *always* compute all local maxima, a natural question is: *when* can we compute them? In this paper, we prove that such a computation is algorithmically possible in situations when we know the number of local maxima – and not possible if we only know two possible candidates for this number.

2 What Is Computable: Reminder

Need for a reminder. To formulate our results, we need to recall the main definitions of what is computable: what is a computable number, what is a computable set, and what is a computable function; see, e.g., [1, 2, 5] for more details.

What is a computable number: intuitive idea. In a computer, we can only represent rational numbers – namely, binary-rational ones. Thus, it is reasonable to say that a real number is computable if it can be algorithmically approximated, with any given accuracy, by rational numbers.

What is a computable number: a precise definition. A real number x is called *computable* if there is an algorithm that, given a natural number n, produces a rational number r_n which is 2^{-n}-close to x.

What is a computable set: intuitive idea. In a computer, we can only store finitely many objects – i.e., a finite set, with computable distances. It is therefore reasonable to define a computable set as a set that can be algorithmically approximated, with any given accuracy, by finite sets – approximated in the sense that every element of our set is 2^{-n}-close to one of the elements from the approximating finite set.

Since a computer has a linear memory, it is convenient to place all the elements of these finite sets – which approximate our set with higher and higher accuracy – into a single infinite sequence x_1, x_2, ... Elements from this sequence approximate any element from the given set. Thus, this sequence must be *everywhere* dense in this set.

In practice, we do not know the exact values of the elements, we only have approximations to elements of the set. Based on these approximations, we can never know whether the resulting set is closed or not – i.e., whether a set of real numbers is the interval $[-1, 1]$ or the same interval minus 0 point. To ignore such un-detectable differences, it is reasonable to assume that our set is *complete*, i.e., that it includes the limit of each converging sequence.

Thus, we arrive at the following definition.

What is a computable set: definition. By a *computable set*, we mean a complete metric space with an everywhere dense sequence $\{x_i\}$ for which:

- there is an algorithm that, given i and j, computes the distance $d(x_i, x_j)$ (with any given accuracy), and
- there exists an algorithm that, given a natural number n, returns a natural number $N(n)$ for which every point $x_1, x_2, \ldots$ is 2^{-n}-close to one of the points $x_1, \ldots, x_{N(n)}$.

By a *computable element* x of a computable set, we mean an algorithm that, given a natural number n, returns an integer $i(n)$ for which $d(x, x_{i(n)}) \le 2^{-n}$.

Comment. From the topological viewpoint, a complete metric space which can be approximated by finite sets is a *compact space*. Thus, computable sets are also known as *computable compact sets*.

What is a computable function: intuitive idea. A computable function f should be able, given a computable real number (or, more generally, a computable element of a computable set), to compute the value $f(x)$ with any given accuracy. Computable elements x are given by their approximations. Thus, to compute $f(x)$ with a given accuracy 2^{-n}, we need to:

- first algorithmically determine how accurately we need to compute x to achieve the desired accuracy 2^{-n} in $f(x)$, and then
- use the corresponding approximation to x to actually compute the desired approximation to $f(x)$.

So, we arrive at the following definition.

What is a computable function: definition. We say that a function $f(x)$ from a computable set to real numbers is computable if:

- first, we have an algorithm that, given n, returns m for which $d(x, x') \le 2^{-m}$ implies that $|f(x) - f(x')| \le 2^{-n}$, and
- second, we have an algorithm that, given i, computes $f(x_i)$.

Comment. The existence of m for every n is nothing else but uniform continuity; so, in effect, we want $f(x)$ to be effectively uniformly continuous.

Now, we are ready to formulate our main results.

3 Main Results

Proposition 1. *There exists an algorithm that:*

- *given an integer m and a computable function $f(x)$ with exactly m local maxima,*
- *always computes the locations of all these maxima.*

Comment 1. It is worth mentioning that for *global* maxima, such an algorithm is not possible even for $m = 2$: no algorithm can, given a computable function at which the global maximum is attained at exactly two points, computes these two locations; see, e.g., [2].

Comment 2. Knowing the exact number of local maxima is important: as the following result shows, if we have an *incomplete* information about this number, we can no longer compute all the local maxima.

Proposition 2. *Let $m < m'$ be two natural numbers. Then, no algorithm is possible that:*

- *given a computable function $f(x)$ with either m or m' local maxima,*
- *always returns the locations of all its local maxima.*

4 Proof of Proposition 1

Auxiliary results needed to describe our algorithm. Our algorithm is based on the several known results. The first is that we can algorithmically compute the maximum of a computable function on a computable set; see, e.g., [1, 2, 5].

We will also use an easy-to-prove fact that for every computable element x_0, the function $f(x) \stackrel{\text{def}}{=} d(x, x_0)$ is computable; see, e.g., [1, 5].

Another result that we will use is that for every computable function $f(x)$ on a computable set, and for every four rational numbers $\underline{r}_1 < \overline{r}_1 < \underline{r}_2 < \overline{r}_2$, we can algorithmically find the values $b_1 \in (\underline{r}_1, \overline{r}_1)$ and $b_2 \in (\underline{b}_2, \overline{b}_2)$ for which the set $\{x : b_1 \le f(x) \le b_2\}$ is also a computable set; see, e.g., [1].

We will also use the fact that each positive rational number p/q is simply a pair of natural numbers. Thus, a tuple consisting of natural and positive rational numbers can be viewed simply as a tuple consisting of natural numbers.

We can algorithmically sort the tuples consisting of positive natural numbers: e.g., first we consider all (finitely many) tuples whose sum is 1, then all the tuples whose sum is 2, etc.

Now, we are ready to describe our algorithm.

Our algorithm: a description. We want to locate all the maxima with a given accuracy 2^{-n}. To do that, by using one of above-mentioned sorting, we try, one by one, all possible tuples consisting of two natural numbers i and k and four positive rational numbers for which $\underline{r}_1 < \overline{r}_1 < \underline{r}_2 < \overline{r}_2 \le 2^{-n}$. For each such tuple, we compute $f \stackrel{\text{def}}{=} f(x_i)$ and $s \stackrel{\text{def}}{=} \max\{f(x) : b_1 \le d(x, x_i) \le b_2\}$ (for appropriate $b_i \in (\underline{r}_i, \overline{r}_i)$) with accuracy 2^{-k}. If for the resulting approximations $\widetilde{f}$ and $\widetilde{s}$, we get $\widetilde{f} > \widetilde{s} + 2 \cdot 2^{-k}$, then we can conclude that $f > s$.

We stop when we get m different tuples such that:

- each of these m tuples satisfies the inequality $\widetilde{f} > \widetilde{s} + 2 \cdot 2^{-k}$ (thus $f > s$), and
- for every two tuples, the distance $d(x_i, x_j)$ between the corresponding elements x_i and x'_i is larger than the sum of the corresponding values $\overline{r}_2$ and $\overline{r}'_2$.

The corresponding m elements x_i are then returned as the desired 2^{-n}-approximations to the locations of the local maxima.

What we need to prove. To prove the proposition, we need to prove:

- that this algorithm always stops, and that
- that this algorithm is correct, i.e., that the results of this algorithm are indeed 2^{-n}-approximations to the desired locations of local maxima.

Let us first prove that the algorithm always stops. Let $M_1, \ldots, M_m$ be the desired local maxima, and let d_0 be the smallest of all the distances between them.

By definition, a local maximum means that $f(M_j) \ge f(x)$ for all x from some neighborhood of M_j. We can always select this neighborhood of size $\le d_0/3$. This way, we can be sure that there are no other local maxima in this neighborhood – and thus, no values $x \neq M_j$ with $f(M_j) = f(x)$, since otherwise these values x will also be local maxima.

Therefore, $f(M_j) > f(x)$ for all all the values from this neighborhood.

Let δ be a rational number which is smaller than all m radii of these neighborhoods. Then $f(M_j) > f(x)$ for all x for which $\delta/2 \leq d(x, M_j) \leq \delta$. The set $\{x : \delta/2 \leq d(x, M_j) \leq \delta\}$ is a compact, so for a continuous function $f(x)$, the maximum is attained at some element from this set. Since for all the points from this set, we have $f(x) < f(M_j)$, we therefore conclude that $f(M_j) > \max\{f(x) : \delta/2 \leq d(x, M_j) \leq \delta\}$.

Due to continuity, for elements x_i which are sufficiently close to M_j, we also have $f(x_i) > \max\{f(x) : \delta/2 \leq d(x, M_j) \leq \delta\}$. Here, if $d(x_i, M_j) \leq \varepsilon$, then $\delta/2 + \varepsilon \leq d(x, x_i) \leq \delta - \varepsilon$ imply $\delta/2 \leq d(x, M_j) \leq \delta$. Thus:

$$\max\{f(x) : \delta/2 \leq d(x, M_j) \leq \delta\} \geq \max\{f(x) : \delta/2 + \varepsilon < d(x, x_i) \leq \delta - \varepsilon\}$$

and

$$f(x_i) > \max\{f(x) : \delta/2 + \varepsilon < d(x, x_i) \leq \delta - \varepsilon\}.$$

So, when we take $\underline{r}_1 = \delta/2 + \varepsilon$ and $\overline{r}_2 = \delta - \varepsilon$, we will get

$$f(x_i) > \max\{f(x) : b_1 < d(x, x_i) \leq b_2\}.$$

Whenever the above strict inequality is true, we will detect it if we compute both sides of this inequality with sufficient accuracy. Thus, eventually, we will indeed find the tuples for which $\widetilde{f} > \widetilde{s} + 2 \cdot 2^{-k}$ and for which each x_i is desirably close to the corresponding local maximum M_j. Hence, our algorithm will indeed always stop.

Let us now prove that the algorithm is correct. To complete our proof, we need to show that when the algorithm stops, the resulting elements x_i are indeed close to the corresponding local maxima. Indeed, when the algorithm stops, for each of the selected m tuples, we get $f(x_i) > \max\{f(x) : b_1 \leq d(x, x_i) \leq b_2\}$.

On the compact set $\{x : d(x, x_i) \leq b_2\}$, the maximum of the continuous function $f(x)$ is attained at some element from this set. Due to the above inequality, this maximum cannot be attained at distances between b_1 and b_2. Thus, this maximum is attained when $d(x, x_i) \leq b_1 < b_2$. So, this maximum is a local maximum of the function $f(x)$, and a local maximum which is (due to $b_2 < \overline{r}_2 \leq 2^{-n}$) 2^{-n}-close to the corresponding element x_i.

On each "zone" $\{x : d(x, x_i) \leq b_2\}$ we thus have a local maximum of the given function $f(x)$. Since $d(x_i, x_j) > \overline{r}_2 + \overline{r}'_2 > b_2 + b'_2$, these zones do not intersect. Thus:

- all m corresponding local maxima are different,
- there are no local maxima outside these zones, and
- within each zone, we have exactly one local maximum which is 2^{-n}-close to x_i.

The correctness is proven, and so is the proposition.

5 Proof of Proposition 2

Our proof-by-contradiction is based on the fact that no algorithm is possible that, given a non-negative real number a, checks whether $a = 0$. This result can be easily proven based on the halting problem result. Indeed, for each Turing machine, we can define a computable real number a for which:

- $r_n = 2^{-n}$ if this Turing machine did not halt by moment n and
- $r_n = 2^{-t}$ if it halted at moment $t \le n$.

As a result:

- If the Turing machine does not halt, then the resulting number is equal to 0.
- Otherwise, if the Turing machine halts at some time t, then we have $a = 2^{-t} > 0$.

The impossibility to check whether a Turing machine halts implies that we cannot check whether $a = 0$.

For each m and m', let us define the following function $f(x)$ on the interval

$$[0, 2m + 2(m' - m) \cdot a] :$$

For $x \in [0, 2m]$, we have:

- $f(x) = 1 - |x - 1|$ for $0 \le x \le 2$,
- $f(x) = 1 - |x - 3|$ for $2 \le x \le 4$,
- $\ldots$,
- $f(x) = 1 - |x - (2m - 1)|$ when $2m - 2 \le x \le 2m$.

For $x \in [2m, 2m + 2(m' - m) \cdot a]$, we have:

- $f(x) = a - |x - (2m + a)|$ when $2m \le x \le 2m + 2a$,
- $f(x) = a - |x - (2m + 3a)|$ when $2m + 2a \le x \le 2m + 4a$,
- $\ldots$,
- $f(x) = a - |x - (2m + (2(m' - m) - 1) \cdot a)|$ when

$$2m + (2(m' - m) - 2) \cdot a \le x \le 2m + 2(m - m) \cdot a.$$

Here:

- When $a = 0$, this function has m local maxima, at points 1, 3, $\ldots$, $2m - 1$.
- When $a > 0$, this function has $m + (m' - m) = m'$ local maxima:
  - m local maxima at points 1, 3, $\ldots$, $2m - 1$, and
  - $m' - m$ local maxima at points $2m + a$, $2m + 3a$, $\ldots$, $2m + (2(m' - m) - 1) \cdot a$.

If we could always return all the local maxima, then by checking whether there is a local maximum close to $2m$, we would be able to check whether $a > 0$ or $a = 0$, and we have already shown that this is not possible. This proves Proposition 2.

Acknowledgments

This work was supported in part by NSF grants HRD-0734825, HRD-1242122, and DUE-0926721, and by an award from Prudential Foundation.

References

1. E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, New York, 1967.
2. V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.
3. K. Villaverde and V. Kreinovich, "A linear-time algorithm that locates local extrema of a function of one variable from interval measurement results," *Interval Computations*, 1993, No. 4, pp. 176–194.
4. G. L. Verschuur and K. I. Kellermann, *Galactic and Extra-Galactic Radio Astronomy*, Springer Verlag, Berlin, Heidelberg, New York, 1974.
5. K. Weihrauch, *Computable Analysis*, Springer Verlag, Berlin, 2000.