Finding the maximum of a polynomial time computable bounded smooth function on an interval is NP-complete

Warren D. Smith
WDSmith@fastmail.fm

date = 5 Feb. 1999

Abstract — **Ker-I Ko in his 1991 book "Complexity theory of real functions" (Birkhauser) defined the notions of polynomial-time computable (P.T.C.) real numbers and real functions, and on p.106 posed as an open question "whether differentiability helps in computing maximum values." We argue the answer is "no."**

But in order to make this argument, Ko's theory needs to be extended to encompass notions of the *code-length* **of a P.T.C. real number, and the** *code transformation time* **for various real arithmetic operations. The question in Ko's unextended theory remains unsolved.**

Our result is: If $f(x)$ is C^∞ and bounded and P.T.C. on $[0,1]$, then the maximum M of f on $[0,1]$ either (1) is not P.T.C., or (2) if it is, then the code of M is not produceable by any polytime algorithm from the code for f, or (3) P=NP.

Keywords — maximization, minimization, polynomial time computable real numbers, NP-completeness, code length.

1 Ko's terminology concerning polynomial time computable reals

A real number x is said to be "computable" if there exists an algorithm which, given an integer $n > 0$, will output a rational number x_n with $|x - x_n| < 2^{-n}$.

A real number x is said to be "polynomial time computable" (P.T.C.) if there exists an algorithm which, given an integer $n > 0$, will output a rational number x_n with $|x - x_n| < 2^{-n}$, in a number of steps bounded by a polynomial in n.

A real-valued function $f(\vec{x})$ is said to be P.T.C. if there exists an algorithm which, given an integer $n > 0$ and given black box access to the polynomial time algorithm box defining any particular P.T.C. real vector $\vec{x}$, will output a rational f_n with $|f(\vec{x}) - f_n| < 2^{-n}$ with cost bounded by a polynomial in n not depending on $\vec{x}$. Here "cost" is defined to be the sum of the compute time plus the number of bits transmitted to and from the black box, where the black box accepts inputs in *unary.*

Note that if $f(x)$ is P.T.C. it is automatically continuous. But it is not necessarily differentiable. A function with a continuous kth derivative is said to be "C^k."

2 Three previous results in the theory of P.T.C. function maximization

Ker-I Ko in his excellent book [4] has discussed the entire area of trying to redo classical real analysis with such computational complexity notions in mind. A typical chapter considers a functional of a P.T.C real function (examples: integration, differentiation, maximization) and asks what can be said about its complexity.

In particular, chapter 3 considers maximization and proves:

1. If MAX_1 is the functional mapping a function $f : [0,1]^2 \to \mathbf{R}$ to $g : [0,1] \to \mathbf{R}$ defined by $g(y) = \max\{f(x,y) | 0 \le x \le 1\}$, then MAX_1 always maps P.T.C. functions f into P.T.C. functions g if and only if P=NP [3].
2. If the operator mapping $f : [0,1] \to \mathbf{R}$ to $g(x) = \max_{t \le x} f(t)$ always takes P.T.C. f to P.T.C. g, then P=NP. Conversely, g is *nondeterministic* polynomial time computable.
3. Further, there exists a P.T.C. $f(x)$ on $[0,1]$ which has a countably infinite number[1] of local maxima, but none of their values are computable reals.

3 Extension of Ko's theory

This paper advocates extending Ko's theory of P.T.C. reals to include notions of the *code length* of the algorithm defining that real (like Kolmogorov complexity) and notions of the *code-transformation time* consumed by various functions and functionals which input and output P.T.C. reals and P.T.C. real functions. This is only natural since P.T.C reals are represented by chunks of computer code.

When we say "code length" of a polynomial time algorithm, we intend for the algorithm to be "self proving,"

[1] Actually "uncountably" but not if a "maximum" means the entire contiguous set of maxima for flat maxima.

i.e. it begins by computing a polynomial giving a runtime bound, and also contains interrupt code (or equivalent counter decrement and test code) which will force it to stop after that polynomial amount of time is used up.

For example, if a and b are P.T.C. reals, then so are $a+b$, ab, a/b, and $a-b$, and with code length bounded by the sum of the code lengths for a and b, plus a constant. If $f(x)$ is P.T.C., then $f(a)$ has codelength bounded by the sum of the codelengths of a and f.

But for *functionals* F acting on $f(x)$ to give other functions $g(y)$ (or just numbers g), the codelengths could change in a more complex manner.

We define F to be a *P.T. functional* if (1) it maps P.T.C functions f to P.T.C. functions g, (2) there is a polynomial time code-transformation algorithm for transforming the input code describing f into the output code describing g.

4 Smooth bounded P.T.C. function maximization

The following theorem is a solution to the open problem proposed by Ko [4] at the end of his chapter 3 of "whether differentiability helps in computing maximum values":

Theorem. *If $f(x)$ is C^∞ and bounded and P.T.C. on $[0,1]$, then the maximum M of f on $[0,1]$ either*

1. *is not P.T.C., or*
2. *if it is, then the code for M (which, given $n>0$, outputs a rational M_n with $|M-M_n|<2^{-n}$) is not produceable from the code for f by an algorithm which runs in time bounded by a polynomial in the length of its input, or*
3. *P=NP.*

Proof. Let $f(x)$ be the following function.

$$f(x) = -\exp(-|\lg x|^{-1.1}) \qquad (1)$$
$$+2\sum_{m\geq 1}\phi(8m^2x-8m)\exp(-|\lg m|^{1.1})P(m).$$

Here $\lg \equiv \log_2$, and $\phi(x)=\exp(1-[1-x^2]^{-2})$ for $|x|<1$, $\phi(x)=0$ if $|x|\geq 1$, is a C^∞ "unit height pulse" function. We have superimposed pulses of height $2P(m)\exp(-|\lg m|^{1.1})$ and width $(2m)^{-2}$ centered at $x=1/m$ (all these pulses have disjoint support). Here $P(m)$ is the polynomial time[2] computable function $P:\mathbf{Z}^+\to\{0,1\}$ which outputs 1 if the bits of m constitute a proof of the Riemann Hypothesis. Otherwise it outputs 0.

Note $\exp(|\lg m|^{1.1})$ grows faster than any polynomial function of m as $m\to\infty$. The only real trick (if trick there be) in our proof is the selection of this precise expression; power law dependence on m would have been inadequate because it would have led to "too-fast oscillation" in the pulses preventing infinite differentiability as $x\to 0^+$; exponential dependence on m would have caused the NP-reduction to break (solving an NP-hard problem in exponential time with a maximization oracle – no great feat).

It is easily seen that no matter what $P(m)$ is (provided it is bounded), f is *infinitely differentiable*. This essentially follows from the fact that $\sum_{m\geq 1} m^{2k}\exp(-|\lg m|^{-1.1})$ converges for any fixed $k>0$. Also, f is *computable in polynomial time*. This is easily seen after realizing that (by design) at most 1 term in the sum (EQ 1) could be nonzero[3].

Now, the Riemann Hypothesis RH is true, if and only if $f(x)$ has a maximum M on $[0,1]$ with $M>0$. RH has a proof $\leq k$ bits long, if and only if $M>\exp(-k^{1.1})$.

Now, clearly in place of "the Riemann Hypothesis" we could have put any logical assertion A (including various "for all" and/or "exists" quantifiers). "The Riemann Hypothesis" has been used merely for dramatic angst.

Thus, an oracle for approximating M to within 2^{-n} in polynomial(n) time, would be able to tell if A had a proof $\leq n^{1/1.1}$ bits long in time polynomial in $n+\text{length}(A)$.

Now, for any *specific* question A in place of the Riemann Hypothesis, this is no paradox since a polytime algorithm *does* exist to tell that. The algorithm simply answers yes if $n>L$, where L is the shortest proof's length, otherwise answer no. Admittedly we may not know L, but L exists, so a polytime algorithm exists. The problem is the codelength of that algorithm might be very large because L might be a very large constant.

But suppose M's code were produceable in a number of steps bounded by a polynomial in the length of f's code, *and* the M-oracle above existed. *Now* we can get a contradiction. In that case there would be an algorithm with runtime bounded by a polynomial(length(A), L) which could solve all meta-questions of the form: "Does A have a proof or disproof $\leq L$ bits long?" The algorithm:

1. Produce the function f of EQ 1 from the code for a proof-checker for alleged proofs of A as discussed above. (In common models of computation and proof systems, a simple proof checking algorithm runs in quadratic time to output "yes – proof" or "that was no proof.")
2. Transform the code for f to get code for M. By assumption, this takes polynomial time.
3. Run the code for M (for polynomial(L) steps).

But no such algorithm can exist unless P=NP as shown by Buss [2]. QED.

Remark. Due to Ko's theorem 2 from our §2, we have NP-*completeness*, not just NP-hardness.

Remark. After this report was completed, S.Buss pointed out to me an improved version [1] of his paper [2]. If we use this instead, we can conclude that it is NP-hard even to approximate the *logarithm* of the maximum value in $[0,1]$ of a bounded C^∞ P.T.C. function (given as computer code) to within a constant factor.

[2] In fact, the compute-time is bounded by a certain stateable quadratic arising from the runtime of a proof validity-checking algorithm.

[3] There might be many proofs of RH, but since the pulses have disjoint support, for the purpose of evaluating $f(x)$ at any particular X, at most *one* of those proofs needs to be checked.

References

[1] Misha Alekhnovich, Samuel R. Buss, Shlomo Moran, Toniann Pitassi: Minimum Propositional Proof Length is NP-Hard to Linearly Approximate (Extended Abstract), in *Mathematical Foundations of Computer Science* (1998) 176-184. (Proc. of 23rd int'l sympos. Brno, Czechoslovakia, Aug. 24-28 1998. Springer Lecture Notes in CS #1450.) Full version in Journal of Symbolic Logic 66 (2001) 179-191.

[2] Sam R. Buss: On Godel's Theorems of Lengths of Proofs II: Lower Bounds for Recognizing k Symbol Provability, pp.57-90 in (P.Clote & J.Remmel eds.) Feasible mathematics II. Birkhauser, 1995.

[3] M.R.Garey & D.S.Johnson: Computers and intractability, A guide to the theory of NP-completeness, Freeman 1979, 1991.

[4] Ker-I Ko: Complexity theory of real functions, Birkhauser 1991.