The Challenger-Solver game: Variations on the theme of P =?NP

Yuri Gurevich*
Electrical Engineering and Computer Science Department
University of Michigan, Ann Arbor, MI 48109, U.S.A.

Logic in Computer Science Column
The Bulletin of EATCS, October 1989†

- *Quisani:* The article of Juris Hartmanis in the previous issue of this Bulletin [Ha] provoked me to think about the famous P =?NP question. I wonder if the question is right.

- *Author:* What do you mean?

- Q: The question seems to me unbalanced. If it turns out that P = NP then programmers will be very happy. (To simplify the matter, I assume that polynomial-time computations are feasible.) On the other hand, proving P $\neq$ NP may have little practical consequences. Imagine that after 200 years of grappling with the P =?NP question we give up and postulate that P $\neq$ NP. Then what? The postulate seems to be very weak. This is especially troubling since establishing the equality P = NP seems very unlikely.

- A: It seems plausible that whoever proves P $\neq$ NP will prove more; for example, that exponential time is required in the worst case.

- Q: Proving that exponential time is required in the worst case may not mean much either. Consider a game between Challenger and Solver where Challenger repeatedly picks instances of a given NP problem (or the corresponding search

*Partially supported by NSF grant DCR 85-03275

†Republished in Y. Gurevich, "Logic in Computer Science", Chapter in "Current Trends in Theoretical Computer Science", Eds. G. Rozenberg and A. Salomaa, World Scientific, Series in Computer Science, Volume 40, 1993, 223–394.

problem) X and Solver tries to solve them. Even if X has exponentially hard instances, Solver may have easy life: It may take exponential time to find a hard instance.

- A: Are you saying that it is *a priori* possible that P $\neq$ NP, but in practice NP problems are easy?

- Q: Yes. I would like to see an alternative to P =?NP which is better balanced. If you prove it then programmers are happy, and if you disprove it then you have some real evidence that NP problems are hard.

- A: If somebody discovers a decision algorithm for, say, SAT (the satisfiability problem for boolean formulas) which works in time $n^{(\ln \ln n)/3}$, will your programmers be happy?

- Q: I am sure they would. I do not make a religion out of polynomial time. By the way, there are so many issues here: Approximate solutions, algorithms that work well on instances of reasonable size, and so on and on. I would like to concentrate attention on the asymptotic behavior of exact (rather than approximate) algorithms and think in terms of the Challenger-Solver game.

- A: Would you be willing to assume that Challenger draws instances of X with respect to some probability distribution μ?

- Q: OK. Of course, the game will greatly depend of what distribution μ is.

- A: Then I can suggest a more balanced hypothesis. Allow me to impose a slight technical restriction on probability distributions under consideration; I can explain and try to justify the restriction later if you wish. Then the hypothesis is that every randomized NP problem (X, μ) is decidable in time which is polynomial on average. Let me call this hypothesis A. It seems to me that, from the point of view of Solver, A is almost as good as P = NP, and if A turns out to be false then Challenger has a strategy to make Solver miserable. The strategy is to draw instances of X at random with respect to distribution μ.

- Q: Are there interesting NP problems with natural probability distributions that turn out to be easier on average?

- A: I think so. Some problems become ridiculously easy on average. For example, if graphs with the same number of vertices are equally probable, then the obvious backtracking algorithm solves 3-colorability problem in 197 steps on average [Wi].

- Q: You mean $197n$ or $197n^2$ or something like that, don't you?

- A: No, I mean 197, full stop. What happens is that there is an enormous amount of cliques of 4 in a random graph. Actually, you can easily modify the algorithm and make it even quicker on average: Just start with explicit search for a 4-clique.

- Q: What about, say, the Hamiltonian circuit problem?

- A: The following is known. Fix a real α between 0 and 1 and suppose that all events "(u, v) is an edge" are independent and each of them has probability α. Here u and v are different vertices of the random graph. With respect to any such probability distribution, the algorithm of [GS] solves the problem in time linear on average.

- Q: I like your suggestion, but I see some difficulties. For example, how do you draw instances efficiently with respect to a given distribution ?

- A: To explain this, I have to introduce two technical notions. One is the notion of Ptime (i.e. polynomial-time) computable distributions. Order all strings (instances of X) first by length and then lexicographically; for brevity, call the resulting order lexicographical. For every string x, let $E(x)$ be the collection of strings y such that $y \leq x$ in the lexicographical order. Call a probability distribution μ *Ptime computable* if it takes only rational values and the μ-probability $\mathrm{Pr}_\mu[E(x)]$ of $E(x)$ is computable in time polynomial in $|x|$.

  The second notion is the notion of domination. Given two probability distributions μ and ν, say that ν *dominates* μ if ν is equal to or greater than μ modulo a polynomial factor. In other words, ν dominates μ if there exists a polynomial p such that $\mathrm{Pr}_\mu[\{x\}] \leq \mathrm{Pr}_\nu[\{x\}] \cdot p(|x|)$. It can be checked [Le1,Gu1] that (X, μ) is Ptime on μ-average if there exists ν such that ν dominates μ and (X, ν) is Ptime on ν-average.

  There seems to be a consensus that natural probability distributions are dominated by Ptime computable ones. Thus, Challenger may draw instances with respect to a Ptime computable probability distribution ν. (Moreover, it may be assumed without loss of generality that every $\nu(x)$ is a binary rational number with $O(|x|)$ digits.) Challenger can use the following strategy: Draw a real number α between 0 and 1 with respect to the uniform probability distribution, then use the binary search to compute the lexicographically first string x such that $\mathrm{Pr}_\nu[E(x)] \geq \alpha$ and then pick that x.

- Q: Good. Here is another difficulty. For simplicity, let us consider a restriction of the game where Challenger draws binary strings of a given length n. Suppose that the probability distribution is uniform, so that the probability to draw a string with $\geq i$ leading zeroes is 2^{-i}. Finally suppose that Solver spends 4^i steps on any string with exactly i leading zeroes.

  It is easy to see that the expected computation time of Solver is exponential in n because Solver spends 4^n steps on a single string 0^n whose probability is 2^{-n} and therefore you have an exponential summand of the expected time right there. However, as a rule, Challenger will draw strings with few leading zeroes and Solver will work a relatively short time. Challenger will need to make a great many attempts in order to force Solver to work hard. Roughly speaking, Challenger needs $t = 2^i$ attempts to draw a string with $\geq i$ leading zeroes and to force Solver to work $\geq t^2 = 4^i$ steps.

- A: Very good. You force me to "disclose" the "official" definition of polynomiality on average [Le1,Gu1]. The definition is slightly different from what you (most justifiably) assumed it to be. Let μ_n be a function that assigns probabilities to strings of length n. The connection between this case and the case of one distribution over all strings is discussed at length in [Gu1]. Further, let T be a function from strings to nonnegative integers (or reals). You have assumed, I guess, that T is polynomial on average if there exists a polynomial $p(n)$ such that

$$\sum_{|x|=n} T(x) \cdot \mu_n(x) \leq p(n).$$

- Q: I think you are right.

- A: Call T *linear on average* if

$$\sum_{|x|=n} T(x) \cdot \mu_n(x) = O(n).$$

  Say that T *polynomial on average* if it is bounded by a polynomial of a linear on average function. Thus, T is polynomial on average if and only if there exists an $\varepsilon > 0$ such that

$$\sum_{|x|=n} (T(x))^{\varepsilon} \cdot \mu_n(x) = O(n).$$

  According to this definition, the expected computation time of Solver in your example is polynomial on average.

- Q: Makes sense. I understand that you cannot exhibit a provably hard randomized NP problem; this would imply P $\neq$ NP. But is there any average-case analog of NP completeness theory?

- A: Yes. Leonid Levin generalized NP completeness theory in [Le1]. (You may find a much longer paper [Gu1] easier to read.) One can prove certain natural randomized problems to be complete in the average case [Le1, Gu1, VL]. Only few natural randomized problems are known to be complete however.

- Q: I have an idea. There is no need for Challenger to be bound to some natural probability distribution. What he really needs is a simple way to generate hard instances.

- A: Excellent. This gives rise to so called samplable distributions. Ben-David, Chor, Goldreich and Luby [BCGL] call a probability distribution μ *samplable* (or P-samplable) if there exists a polynomial p and a probabilistic algorithm M such that $\Pr_\mu(\{x\})$ is exactly the probability that M outputs x within $p(|x|)$ steps. They prove, assuming the existence of a one-way function, that some samplable distributions are not dominated by Ptime computable ones.

- Q: This brings us I guess to the question what was the "slight technical restriction" that you have mentioned formulating hypothesis A ?

- A: The original restriction on probability distributions was that they are dominated by Ptime computable ones [Le1,Gu1]. Fortunately, Imagliazzo and Levin proved very recently (a couple of days ago as a matter of fact) that any NP problem (or the corresponding search problem) with samplable probability distribution reduces to an NP problem with Ptime computable probability distribution [IL].

- Q: Great. Let me summarize. If hypothesis A is true then Challenger does not have any means to produce hard instances in polynomial time. Challenger may use the most devious probabilistic Ptime generator of instances; still the expected time of Solver will be bounded by a polynomial of the time of Challenger. On the other hand, if hypothesis A is false then there is an NP problem X and a Ptime computable distribution μ such that if Challenger draws instances of X with respect to μ then the expected computation time of Solver will not be bounded by any polynomial of the Challenger's time.

- A: No, this is not quite true. A is formulated in terms of the average of some root of Solver's time. Consider your own example where the expected Solver's time is polynomial on average and suppose that Challenger draws a string by

tossing a fair coin n times. Then he needs $k \cdot n$ steps to draw k strings. The expected time of Solver, needed to solve k strings, is roughly $k \cdot 2^n$ which is an exponential amount in terms of Challenger's time unless k is very large.

- Q: Sorry, I am confused. I had an impression that we do compare the expected Solver's time with the time of Challenger.

- A: I am afraid that the expected computation time of Solver is not an ideal measure after all. It may be misleading if we want to compare the computation times of the two players. As you have pointed out earlier, it is intuitively clear in your example that Solver's time is roughly the square of Challenger's time. It is true that Challenger may be lucky and draw 0^n the very first time and this way make Solver sweat and swear, but such a development is not very likely. I would propose to use the median time of Solver [Le3]. A good case may be made that, in the situations we are interested in, the median represents typical behavior better. See how nicely it works in your example.

- Q: Speaking about the general case (rather than my example), it is not clear to me whether it is possible at all to express the average (in whatever sense) computation time of Solver in terms of the computation time of Challenger. The game was defined loosely and I am not sure how to make things more precise. There are many different procedures for Challenger to draw instances. Each such procedure gives rise to some relation between the average Challenger's time and the average Solver's time. How to deal with all these relations and how to go from relations to functions?

- A: Maybe, we can do something. Consider first a simple special case where the probability distribution μ is such that the expectation of Challenger's time needed to produce one instance is finite.

- Q: Seems a very special case indeed.

- A: Wait. In the special case, the expected Challenger's time needed to produce k instances is proportional to k and, for our purposes, may be identified with k. Consider the median time of Solver as a function of k. For our purposes, the median time may be defined as the maximal integer t such that with probability $\geq 1/2$ Solver spends time $\geq t$ on k instances. It may be checked that the median time of Solver is polynomial in k if and only if the problem X in question is Ptime on μ-average with respect to the official definition above. The median time of Solver [Le3] may be seen as the median rate of growth of Solver's time.

- Q: Somehow I don't have a good feel for the median time.

- A: Alternatively, let $T(x)$ be the computation time of Solver needed to solve a single instance x. We may introduce the average growth rate $R_T(k)$ of T to be the maximal integer t such that $\Pr_\mu[T(x) \geq t] \geq 1/k$. If Solver solves k instances, he is likely to spend time $\geq R_T(k)$ in the worst of the k cases. You can easily convince yourself that the two rates are closely related. The average growth rate $R_T(k)$ is polynomial in k if and only if the randomized problem (X, μ) is polynomial on average.

- Q: Very nice, but the special case is extremely special.

- A: Surprisingly, the general case reduces to the special one. Suppose that Challenger employs some probabilistic procedure M which generates a string x in time bounded by a polynomial of $|x|$. Let $\mu(x)$ be the probability that M generates x. I define a new probabilistic procedure N:

  1. Pick a number 2^i (the limit on computation time) with respect to an auxiliary probability distribution ξ such that $\Pr_\xi(\{2^i\})$ is proportional to $1/(i^2 \cdot 2^i)$. (Notice that the expected value of the number picked is finite.)
  2. Run procedure M. If M outputs a string x in time $\leq 2^i$ then output x; otherwise stop M after 2^i steps and output the empty string (or whatever default you like).

  Procedure N gives rise to a new probability distribution ν on strings. It is easy to see that ν dominates μ, and of course μ dominates ν as well. Thus, X is polynomial on μ-average if and only if it is polynomial on ν-average. It is intuitively clear that computation times of Challenger and Solver do not change that much in the transition to the new model. Meantime, I am afraid, we have exhausted our time.

- Q: Wait, let me bring just one other issue. Hard problems may be very useful. I heard about cryptography, perfect pseudo-random number generators, zero-knowledge interactive proofs, etc. Is the negation of A sufficient to have some or all of those goodies?

- A: This is unknown. Let B be the hypothesis that there is a one-way function. Impagliazzo, Levin and Luby proved recently [ILL] that the existence of perfect pseudo-random number generators follows from B (it obviously implies B), and perfect pseudo-random number generators open a door to all kinds if exciting things.

- Q: I do not know what a one-way function is.

- A: Roughly speaking, a function f is one-way if it is Ptime computable (and the input length is bounded by a polynomial of the output length), but the problem of inverting f is hard.

- Q: And what is exactly the problem of inverting f?

- A: An instance of the problem is a string y, and the corresponding task is to find a string x such that $f(x) = y$. The probability distribution may be called f-induced: The probability assigned to a given string y is the uniform probability of the event $\{x : \ f(x) = y\}$. Notice that if y does not belong to the range of f then its probability is zero. Thus, without loss of generality, one may restrict attention to instances in the range of f.

- Q: How probable is it that the negation of A implies B?

- A: I do not know. But one-way functions are closely related to the Challenger-Solver game. (This is an observation of Impagliazzo [Le3].) A one-way function f allows Challenger to produce instances of a hard problem (the f-inversion problem) for which he knows solutions. The converse is also true. Suppose that a function h produces pairs (x, w) where x is an instance of some problem X and w is a solution for x. For example, if x is an instance of SAT then w is a satisfying assignment for the boolean formula x. Further suppose that the problem X with the h-induced probability distribution is hard. Let f be the result of composing h with the projection function $\mathrm{Proj}_1(x, w) = x$. Then f is one-way. For, inverting f allows us to solve X.

- Q: I see. Before we were speaking about Challenger producing hard instances for Solver, but they also could be hard for Challenger. Now we speak about Challenger producing instances which are hard for Solver but easy for Challenger.

- A: That's right. *A priori* it seems quite possible that Challenger is able to produce hard instances but not hard solved instances.

  Notice that producing hard instances together with their solutions immediately allows some primitive cryptography. Imagine that, when you check a valuable thing into a bank, you generate a boolean formula x together with a password w which is a satisfying assignment for x. The clerk will know only the formula x but not the password w and therefore will be unable to give w to anybody. When you want to take you thing out, you give the clerk the assignment w and he checks that w satisfies x. He does not check whether w is the assignment that was generated originally; he checks only that the w satisfies x.

- Q: B seems to be a great alternative to P $\neq$ NP.
- A: In this connections, you may be interested to read [Le2]; it is hard to read though. Our time is up.
- Q: This is a pity. I wanted to ask about things closer to logic, in particular, about the possibility that the equality P = NP is independent from Peano Arithmetic.
- A: What do you mean "closer to logic" ? I thought we were talking straight logic all along. You know, logicians have this imperial claim on foundations of anything.

Acknowledgement and Warranty. First, the references cited contain all the material in various degrees of implicitness. The degree varies from very much implicit (in [Le1]) to quite explicit (in [Le3]). Second, nobody else saw the final version of this paper before it went to the publisher, though Andreas Blass and Leonid Levin commented on a very preliminary (several full days before the deadline) version of it for which I am very grateful to them. (Have you noticed how quickly deadlines roll on you?) It follows (doesn't it?) that the only responsibility this author can take is the responsibility for all distortions, misrepresentations, typos and errors.

References

[BCGL] Shai Ben-David, Benny Chor, Oded Goldreich and Michael Luby. "On the Theory of Average Case Complexity". 21st Annual ACM Symposium on Theory of Computing, ACM, 1989, 204–216.

[GL] Oded Goldreich and Leonid A. Levin. "A hard-core predicate for all one-way functions". 21st ACM Symposium on Theory of Computing, May 1989, ACM, 25–32.

[Gu1] Yuri Gurevich. "Average case complexity". To appear in a special issue of J. Computer and System Sciences with selected papers of FOCS'87. [Available to be emailed on request in the form of a latex file.]

[GS] Yuri Gurevich and Saharon Shelah. "Expected computation time for Hamiltonian Path Problem", SIAM J. on Computing 16:3 (1987), 486–502.

[Ha] Juris Hartmanis. "Gödel, von Neumann and the P =?NP Problem". This Bulletin, nu. 38, June 1989, 101–107.

[IL] Russell Impagliazzo and Leonid A. Levin. Private communication.

[ILL] Russell Impagliazzo, Leonid A. Levin and Michael Luby. "Pseudo-random generators from one-way functions", 21st ACM Symp. on Theory of Computing, ACM, 1989, 12–24.

[Le1] Leonid Levin. "Average Case Complete Problems". SIAM Journal of Computing, 1986.

[Le2] Leonid A. Levin. "One-way functions and pseudorandom generators". Combinatorica 7:4 (1987), 357–363.

[Le3] Leonid A. Levin. Private communication.

[VL] Ramarathnam Venkatesan and Leonid Levin. " Random Instances of a Graph Coloring Problem are Hard". 20th Symp. on Theory of Computing, ACM, 1988.

[Wi] Herbert S. Wilf. "Some Examples of Combinatorial Averaging". American Math. Monthly 92 (1985), 250–261.