How Important Is Theory for Practical Problems? A Partial Explanation of Hartmanis' Observation

Vladik Kreinovich and Luc Longpré
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
{vladik,longpre}@cs.utep.edu

Hartmanis' observation. How important is theory for practical problems? In his recent talk [4], J. Hartmanis analyzed several cases when a long-standing open theoretical problem was solved, and came up with a rather unexpected conclusion: that if some theoretical result is very difficult to prove or disprove, then usually, this result has little or no practical usefulness.

This observation seemed troubling to us, because it runs contrary to the usual theoreticians' belief that complex theoretical results are practically useful. Because of this, we decided to formalize this observation and to investigate whether Hartmanis' informal statement is true or not in the resulting formalization. We started with a very simple formalization, and, somewhat to our surprise, discovered that in this simple formalization, Hartmanis' intuition is right and our original intuition was wrong: complex theoretical results are indeed, in some precise sense, of little practical use. Thus, this gives a (partial) explanation of Hartmanis' observation.

In this paper, we describe this result. Our result is simple, but we believe the readers will be interested because it does provide a justification of an otherwise unexplained observation. Of course, the reader should take into consideration that this result is based on a rather simple possible formalization, and more adequate and more sophisticated formalizations are necessary.

Towards formalization: general idea. In order to formalize Hartmanis's observation, we need to formalize two intuitive notions: what it means for a statement S to be "practically useful", and what it means for a statement to be "difficult to prove".

Both terms are difficult to formalize in a way that would best fit our intuitive understanding of the corresponding notions. In view of this difficulty, we will only formalize the features of these notions which are necessary to prove our formal version of Hartmanis' statement. Namely, we will define the following three notions:

- The first is the notion of a *potential application*; this notion will be defined in such a way that every application (in the intuitive sense of this word) is also (hopefully) an application in the sense of this definition.

  As we will see from our definition, the inverse is not necessarily true: a potential application in the sense of our definition is not necessarily an application in the intuitive sense of this word.

- The second notion will be related to the *usefulness* of an application.

- The third is the notion of a *potentially complex* proof; this notion will be defined in such a way that every proof which is complex in the intuitive sense is also complex in the sense of our definition.

  Again, the inverse is not necessarily true: a proof which is potentially complex in the sense of our definition is not necessarily complex in the intuitive sense of this word; in the last section of this paper, we speculate on how to make this definition closer to the intuitive notion of proof complexity.

Our result will then be that whenever we have a potential application of a potentially complex proof, this potentially application is not very useful. As we have just mentioned, we are choosing our definitions in such a way that:

- every application (in the intuitive sense of this word) is a potential application in the formal sense, and

- every proof which is complex in the intuitive sense is also potentially complex in the sense of our definition.

Thus, from our formal (proved) proposition, we can make an informal conclusion: that every application of a complex proof is not very useful.

Before we give formal definitions, let us first provide motivations for them.

Towards defining potential applications. Let us start with the notion of a potential application. A typical expected practical application of a theoretical statement S is that it would lead to a practical algorithm for solving a practically useful problem. For example, one of the remaining long-standing important open problems in mathematics is the generalized Riemann hypothesis; it is known that if Riemann's hypothesis is true, then a certain fast and simple algorithm $\mathcal{U}$ would be a (provable) primality test (see, e.g., [1, 6]).

Without the Riemann's hypothesis, we can only conclude that the algorithm $\mathcal{U}$ is correct if it says that a given number n is composite; if the algorithm $\mathcal{U}$ concludes that n is prime, then, without Riemann's hypothesis, it may be possible that n is actually a composite number $n = n_1 \cdot n_2$ for some integers $n_i > 1$. Thus, the correctness of the algorithm $\mathcal{U}$ means that for every pair (n_1, n_2) of integers $n_i > 1$, the result of applying the algorithm $\mathcal{U}$ to the product $n_1 \cdot n_2$ is "composite". Since the algorithm $\mathcal{U}$ is feasible, we can express its correctness as $\forall n_1 \forall n_2\, A(n_1, n_2)$, where A is a feasible (polynomial-time computable) predicate. This formula can be further simplified as follows: by adding 0's in front of one of n_i, we can always guarantee that the binary descriptions of n_i have the same length (or lengths differing by 1). Thus, the correctness of the algorithm $\mathcal{U}$ can be described as $\forall x\, P(x)$, where x runs over all possible binary input strings, and $P(x)$ is a feasible predicate (namely, the result of applying $\mathcal{U}$ to the product $n_1 \cdot n_2$, where n_1 is the integer whose binary code is the first half of the string x, and n_2 is the integer whose binary code is the second half of x).

Similarly, in general, by a potential application of a statement S, we mean the fact that from S, we can conclude that $\forall x\, P(x)$, where x runs over all binary strings, and $P(x)$ is a feasible predicate.

In accordance to what we mentioned in describing the general idea of our paper, this formulation only works one way: a "potential application" in the above sense means that we have deduced an implication $S \Rightarrow \forall x\, P(x)$, but the fact that we have this implication for some predicate $P(x)$ does not necessarily mean that S is a real application (because the only thing we require about the predicate $P(x)$ is that it is feasible, and not all feasible predicates come from useful algorithms like $\mathcal{U}$).

Towards defining potential usefulness. The fact that $P(x)$ is a potential (or even real) application of a statement S does not necessarily mean that this is actually a useful application.

For example, if we can easily prove the universal statement $\forall x\, P(x)$ without using the complex statement S, then S is clearly not useful at all in guaranteeing that the algorithm (corresponding to the predicate $P(x)$) works for all inputs x.

Similarly, if we can prove, without using the statement S, that $P(x)$ is true for "almost all" x (i.e., for a large portion of the strings) – and therefore, that for almost all inputs, we can guarantee that the corresponding algorithm works correctly – then, intuitively, for this application, the statement S is not very useful.

We can, therefore, define the usefulness of a statement S in an application $P(x)$ for given input length n as, e.g., the largest portion of the strings for which $P(x)$ is true irrespective of whether S is true or not.

To avoid defining too many new terms, we will not give a new definition for usefulness, but we will incorporate this notion into our main result.

Towards defining a notion of a potentially complex proof. Let us now consider how to formalize the notion of a "potentially complex proof". In mathematical logic, a complexity of the proof is often measured by its length. So, it seems reasonable to define a potential complexity of a proof as its length, and a potential complexity of a statement as the shortest length of its proof.

The reason why we added the word "potentially" is that this definition does not necessarily coincide with the intuitive notion of a complexity of a proof. This is especially true if we measure the length of a traditional proof in, say, first order logic, or any other similar formalism. For example, intuitively, a long proof which consists of a series of long routine computations (e.g., a straightforward proof of an arithmetic equality) is complex in the above sense, but intuitively, it is clearly simpler than a short proof which consists of several difficult-to-grasp innovative ideas. In short, if a certain part of a proof can be easily programmed and implemented by a computer, then this part is (intuitively) simple. To eliminate this discrepancy between the intuitive and formalized complexity of a proof, it makes sense to count each simple application of a feasible program as a single step of the proof.

Since we mentioned a computer, it makes sense to add that a computer can also use a random number generator (we mean a true random number generator, in which random numbers come from the physical process like a resistor noise and not from a pseudo-random algorithm). The reason why this addition is useful is that, e.g., if we have two different functions $f_1(x)$ and $f_2(x)$ from real numbers to real numbers, then it is very easy to prove that they are different by simply plugging in a random number α into both functions: if $f_1(\alpha) \neq f_2(\alpha)$, then the functions $f_i(x)$ are different, and, e.g., for analytical functions, the probability that $\Delta f(\alpha) = f_1(\alpha) - f_2(\alpha) = 0$ is (close to) 0. It therefore makes sense, when defining the proof's complexity, to consider one call to a random number generator as one step of the proof.

As a result, we arrive at the following definitions:

Definition 1.

- *Let $\mathcal{L}$ be a fixed first order language, with axioms and deduction rules (e.g., language of set theory which is used as a basis for mathematics). For this language, we have a natural notion of a proof.*

- *Let S be a statement from the language L. By a* potential application of S, *we mean a feasible predicate $P(x)$ (whose truth value is defined for an arbitrary binary string x), for which we have a proof p (of length $\mathrm{len}(p)$) of the implication $S \Rightarrow \forall x\, P(x)$.*

Definition 2. *By a computer proof of a statement from $\mathcal{L}$, we mean a sequence of steps of the following type:*

- *generating a random binary string of a given length n;*
- *checking whether a given feasible predicate $Q(x)$ holds for a given binary string x;*
- *applying one deduction step (of the original deduction system).*

By the potential complexity of a computer proof, we mean the total complexity of all its steps, where:

- *a generating step has complexity n, and*
- *a checking or a deduction step have complexity 1.*

Let $\varepsilon < 1$ be a positive real number. We say that the computer proof is correct if it leads to a correct proof with probability $\geq 1 - \varepsilon$.

Comment. We defined each checking step as having complexity 1. Alternatively, we could define its complexity as the number of elementary computational steps which are necessary for this checking. Since we only consider feasible (polynomial time) algorithms $P(x)$, with the new definition, we can prove a similar proposition (with a slightly different numerical estimate).

Definition 3. *Let C be a positive integer. We say that a statement S is C-potentially complex if any correct computer proof of S or $\neg S$ has potential complexity $\geq C$.*

Proposition. *Let S be a C-potentially complex statement, and let $P(x)$ be its potential application (with a proof p of length $\mathrm{len}(p)$). Then, regardless of whether S is true or not, for every integer n, the portion P_n of strings x of length n for which the predicate $P(x)$ is true satisfies the inequality $P_n \geq \varepsilon^{n/(C-\mathrm{len}(p)-2)}$.*

Comment. When the statement is very complex (i.e., C is large), and the strings are of reasonable length (i.e., n is small), then $n \ll C$, and

$$\varepsilon^{n/(C-\mathrm{len}(p)-2)} = \exp\left(-\frac{\ln(\varepsilon)\cdot n}{C-\mathrm{len}(p)-2}\right) \sim 1 - \frac{\ln(\varepsilon)\cdot n}{C-\mathrm{len}(p)-2}.$$

Since $C \gg n$, this probability is very close to 1. Thus, if, from a complex theoretical statement S, we can deduce a universal practically motivated statement $\forall x\, P(x)$, then even without S, for a reasonable length n, we can guarantee that $P(x)$ is true for "almost all" strings of this length.

In our motivation of the notion of a potential application, we mentioned that the intended meaning of the truth value of a predicate $P(x)$ for a given string x is that a certain practical algorithm works correctly for this input x. In these terms, the above proposition means the following: for "almost all" inputs, we do not need the complex statement S to guarantee that the practical algorithm works for these inputs.

Proof. If S is true, then $\forall x\, P(x)$ is also true, so the portion P_n is equal to 1. Therefore, to prove the proposition, it is sufficient to consider the case when S is false. Let us prove that in this case, if $P_n < \varepsilon^{n/(C-\mathrm{len}(p)-2)}$, then, contrary to our assumption, we will get a computer proof of $\neg S$ whose potential complexity is $< C$.

Indeed, from $S \Rightarrow \forall x\, P(x)$, it follows that for any string z, we have $\neg P(z) \Rightarrow \neg S$. Thus, for any positive integer t, we can consider the following computer proof: t times run a random number generator to generate a random string α of length n, and then check whether $P(\alpha)$ is true. If $P(\alpha)$

is false for one of these strings α, then use the proof p to conclude that $\neg S$. The total potential complexity of this proof is $t \cdot n + \text{len}(p) + 1$ (one step to conclude $\neg S$ from $\neg P(\alpha)$ and $S \Rightarrow \forall x\, P(x)$). Thus, if $t < (C - \text{len}(p) - 1)/n$ (e.g., if $t \leq (C - \text{len}(p) - 2)/n$), then we get a computer proof of $\neg S$ of potential complexity $< C$.

To get a contradiction with our assumption, we must show that this computer proof is correct, i.e., that it leads to an actual proof with a probability $\geq 1 - \varepsilon$. Let us check this inequality. For each α, the probability that $P(\alpha)$ holds is equal to P_n. Since consequent tests are independent, the probability that $P(\alpha)$ will be true for all t tests is equal to P_n^t. Thus, with probability $1 - P_n^t$, we get a string α for which $\neg P(\alpha)$, and hence, a proof of $\neg S$. Thence, if $1 - P_n^t \geq 1 - \varepsilon$ (which is equivalent to $P_n^t \leq \varepsilon$), this computer proof is correct. So, if $P_n \leq \varepsilon^{1/t}$, we get a contradiction. Thus, $P_n > \varepsilon^{1/t}$. Substituting the above formula for t, we get the desired inequality. The proposition is proven.

Where do we go from here. As we have mentioned, we only provide a rather simple formalization of Hartmanis' idea. In particular, our formalization of the proof's complexity is very crude. Just like a more intuitive formalization of the notion of complexity of a string is its Kolmogorov complexity, probably a better formalization of the notion of complexity of a proof p is a (resource-bounded) Kolmogorov complexity of this proof (resource-bounded since we are interested in feasible algorithms only) [2, 5]. For this formalization, we expect a similar result. (Note a similarity with [3], where we showed the relation between Kolmogorov complexity and testing program correctness, i.e., a statement of the similar type $\forall x\, P(x)$.)

Acknowledgments. This work is supported in part by NASA under cooperative agreement NCC5-209, by NSF grants No. DUE-9750858 and CDA-9522207, by United Space Alliance, grant No. NAS 9-20000 (PWO C0C67713A6), by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518, and by the National Security Agency under Grant No. MDA904-98-1-0561.

References

[1] E. Bach, *Analytic methods in the analysis and design of number-theoretic algorithms*, MIT Press, Cambridge, MA, 1985.

[2] C. Calude, *Information and randomness: An algorithmic perspective*, Springer-Verlag, Berlin, 1994.

[3] A. Q. Gates, V. Kreinovich, and L. Longpré, "Kolmogorov Complexity Justifies Software Engineering Heuristics", *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 1998, Vol. 66, pp. 150–154.

[4] J. Hartmanis, *The P vs. NP Problem*, Unpublished talk at New Mexico State University, Las Cruces, NM, October 11, 1999.

[5] M. Li and P. Vitányi, *An introduction to Kolmogorov complexity and its applications*, Springer-Verlag, N.Y., 1997.

[6] G. L. Miller, "Riemann's hypothesis and tests for primality", *Journal of Computer and System Sciences*, 1976, Vol. 13, No. 3, pp. 300–317.