On Average Time Hierarchies

Mikael Goldmann Per Grape Johan Håstad

Royal Institute of Technology
Stockholm, SWEDEN

September 26, 1997

Abstract

For a time-constructible function T we give an explicit language L_T which can be recognized in time $T(n)$. We prove that any Turing machine that recognizes L_T requires time close to $T(n)$ for most inputs, thus forming an average time hierarchy. The existence of the average time hierarchy was known, but depended on languages defined by complicated diagonalization. We also give simple proofs for the known stricter hierarchy for functions.

Key words: computational complexity, average time complexity, hierarchy theorems.

1 Introduction

Much effort is spent in the field of computational complexity on showing that there are functions which are hard to compute. It has long been known that diagonalization techniques from recursion theory can be used to obtain very strong lower bounds on, for instance, the time needed to compute some function. Intuitively one uses a time-bounded version of the halting problem: "Given a machine x and an input y, is it true that x halts in $T(|y|)$ steps on input y?" It is now easy to see that this question cannot be answered using much less than $T(|y|)$ steps of computation.

Without formalizing the above argument, let us point out some of its shortcomings.

1. As the "hard" problem was formulated above it is not clear how to solve it computationally at all; for some choices of T the corresponding function is not computable.

2. The results obtained in this way are "worst case" results. That is, there are hard inputs, but they might be few and far between.

3. The functions that are proven difficult in this way are sometimes "unnatural" and/or not explicitly defined.

Hartmanis and Stearns [HS65] proved that there are functions computable in time $T^2(n)$ but not in time $T(n)$, thus exhibiting a hierarchy of complexity classes. One of the referees pointed out to us that these results were known earlier in the Sovietunion (by Tseitin [Tse56], cited in [Yan59]). Since this not is not a historical paper we have not looked deeper into this question of priority. Hennie and Stearns subsequently improved [HS65] to a tighter hierarchy [HS66]. We use a simulation from their paper to prove our results.

The results in [HS65, Tse56, HS66] are for the worst case. The history of "average case" hardness is longer than one might perhaps think. Rabin proved in [Rab60] that given a recursive function T there is a function such that any machine computing it must use time $T(n)$ on all but a finite number of inputs. Rabin's results are actually considerably more general and apply to any complexity measure satisfying a small set of axioms. Rabin did not state any other upper bound for his hard functions than that they were recursive or primitive recursive. Blum, building on [Rab60], was first to show the existence of *compressed* predicates [Blu67]. That is, predicates that can be "sandwiched" between two time bounds T_1 and T_2 in such a way that any machine computing the predicate must use at least $T_2(n)$ steps on almost all inputs, and there is a machine that computes the predicate in $T_1(n)$ steps. In particular, given T one can construct a predicate which requires time $T(n)$ for almost all inputs, but is computable in time $(T(n))^7$. Actually, Blum's results, like Rabin's, hold for a large class of complexity measures.

The next question would be if one could obtain an "average-time" hierarchy as tight as that of Hennie and Stearns, and with only a constant number of easy instances. In [Lev73] Levin shows a result that implies such a hierarchy. Levin does this by constructing languages that have the same worst case and best case complexity. These separating languages are constructed by a complicated diagonalization. We obtain a less tight hierarchy but, on the other hand, our languages are explicitly defined. We show that given a function $T(n)$ and a constant c there are predicates computable in time $O(T(n)\log n(\log\log n)^2)$ that requires time $\Omega(n^{-c}T(n)/\log T(n))$ on a fraction $1 - O(n^{-c}\log n)$ of the inputs of length n.

In this paper we first present two average case time hierarchies, one for functions and one for languages. In both cases we prove that for time bounds T_1 and T_2, which are not too far apart, there are explicit functions which can be computed in time T_1, but any program that computes the function can halt in time T_2 for only a small fraction of the inputs. The result for functions uses a fairly straightforward generalization of the methods of [HS66], while the result for languages uses a different technique and in particular it relies on Kolmogorov Complexity. In spite of this, the language hierarchy is less dense than the corresponding result for functions.

An important problem which is loosely connected to the present paper is the question whether there are problems in **NP** which cannot be solved in expected polynomial time with respect to a probability measure on the input space. Our results have no direct relation to this problem, and hence we refer the reader to [Gur91, Lev86] and their references for a discussion of these questions.

2 Bounds by diagonalization

We use the alphabet $\{00, 01, 10, 11\}$ to encode Turing machines. We will assume that a description of a Turing machine contains a straightforward encoding of its next-step function. We will also assume that all Turing machines have the same alphabet. The actual description

of a machine uses 00 and 01, while 11 marks the end of the description. Thus, we can say that a string $x \in \{0,1\}^*$ has a (description of a) Turing machine, x', as a prefix if $x = x'11x''$ where $x' \in \{00,01\}^*$.
A standard diagonalization can now be defined.
Let the function F be defined by:

$$F(x) = \begin{cases} 1 & \text{if } x \text{ has a machine } x' \text{ as prefix and } M_{x'} \text{ halts} \\ & \text{with output 0 on input } x \text{ within } T(|x|) \text{ steps} \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 1 *Let M be a Turing machine computing the function F. Then there is a constant c_M depending only on M such that M requires time $T(|x|)$ on a fraction 2^{-c_M} of inputs x with $|x| \geq c_M$.*

Proof: Let M be a machine computing F with description y. By diagonalization, M must use time at least $T(|x|)$ when $x = y11x''$. For inputs x of length at least $c_M = |y| + 2$ the fraction of such inputs is 2^{-c_M}. ∎

There are two major problems with this proposition. First, there are many Turing machines that compute F, and the fraction of hard instances will depend on the actual machine used. The proof only shows that for any machine there is a constant fraction of the inputs that is hard. By choosing machines with long description the constant given by the proof can be made arbitrarily small. In subsequent sections we develop hierarchies of functions where the fraction of hard inputs tends to 1 as the input length grows, but the rate will depend on the machine used. Second, it is not clear how hard it is to compute F. To handle this we need a theorem by Hennie and Stearns.

Theorem 2 ([HS66, Theorem 1]) *The number of operations for a two-tape Turing machine needed to simulate n operations of a k-tape Turing machine is at most $\alpha n \log_2 n$ for $n > 1$, where α is a constant independent of n.*

On the other hand it is not difficult to see that if we are given extra tapes we can simulate a Turing machine with constant over-head.

Lemma 3 *There is a universal Turing machine A_k, that given the description x of a k-tape Turing machine M and and input y to M, simulates m steps of M in time βm. Here $\beta \leq c|x|^2$ and c is a universal constant. The machine A_k uses $k+2$ tapes.*

Proof: Use the two extra tapes of A_k to store x and the state q which the machine currently is in. The other tapes correspond to the tapes of M. A step can now be completed by locating the line of q in x. By a naive algorithm this can be done in time $O(|q||x|)$ Using $|q| \leq |x|$ the result follows. ∎

Remark: Using more careful programming of Turing machines it might be possible to gain up to a factor of $|x|$ in the above lemma. This, however, would not be of any great consequence to the current paper.

We need the following definition:

Definition 1 *A function $U(n)$ is time-constructible if there is a Turing machine that on input x prints $U(|x|)$ ones on the output tape and halts in time $O(U(|x|))$.*

Let $U(n)$ denote an arbitrary time-constructible function and consider the following Turing machine V_U. On an input x of length n

- Write $U(n)$ ones on a special tape.
- If prefix x' of x codes a two-tape Turing machine then try to simulate it on input x for $U(n)$ steps (of the simulating machine). If it halts with output 0 then halt with output 1. If it halts with any output different from 0 then halt with output 0.
- If we run out of time or if x does not encode a Turing machine, halt with output 0.

The simulation is done using Lemma 3. By simulating for $U(n)$ steps we mean that the simulating machine runs for $U(n)$ steps. Since it is easy to use the special tape as a stopwatch, the machine V_U can be made to run in $O(U(n))$ steps on a multitape Turing machine. Clearly the proof of Proposition 1 applies also to this function and we get:

Theorem 4 *Suppose T and U are time-constructible, $T(n) > n$, and*

$$\lim_{n\to\infty} \frac{T(n)}{U(n)} = 0.$$

Then there is a language L_U that can be recognized in time $O(U(n))$ on a multitape Turing machine such that any two-tape Turing machine M that recognizes L_U must use time at least T on a constant fraction c of the inputs. Here $c > 0$ is a constant that depends on M.

Applying Theorem 2 we get an immediate corollary.

Corollary 5 *If T and U are time-constructible, $T(n) > n$, and*

$$\lim_{n\to\infty} \frac{T(n)\log T(n)}{U(n)} = 0,$$

then there is a language L_U which can be recognized in time $O(U(n))$ on a multitape Turing machine such that any Turing machine M that recognizes L_U must use time at least T, on a constant fraction c of the inputs. Here $c > 0$ is a constant that depends on M.

This construction is however unsatisfactory since the fraction c can be arbitrarily small. We remedy this in two steps. First we define a function V that takes binary strings to binary strings. Any machine computing V needs "long time" on almost all inputs.

3 A Function Hierarchy

We need some notation before we can construct functions that require "long time" on many inputs.

Let $l(n)$ be some function such that $l(n) \to \infty$ and $l(n) = o(n)$, for example, $l(n) = \lfloor \log n \rfloor$ or $l(n) = \lfloor \sqrt{n} \rfloor$. Let $m(n) = \lfloor n/l(n) \rfloor$. In what follows $|x| = n$, $l = l(n)$ and $m = m(n)$. For notational simplicity we assume that $n = lm$.

The function V is defined as follows:

$$V : \{0,1\}^n \to \{0,1\}^l$$
$$V(y_1 \dots y_l) = z_1 \dots z_l$$

where

$$\begin{aligned} x &= x_1 \dots x_n \\ y_i &= x_{(i-1)m+1} \dots x_{im} \\ z_i &= \begin{cases} 1 & \text{if } y_i \text{ has a legal two-tape machine } x' \text{ as prefix and } M_{x'} \text{ halts} \\ & \text{within } T(n) \text{ steps with 0 as its } i\text{th output bit on} \\ & \text{input } x \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Like before, we will instead define the function by the output of a simulating program. Instead of running $M_{x'}$ for $T(n)$ steps we simulate it for $U(n)/l(n)$ steps of the simulating machine. Otherwise we use the above definition.

Now consider a machine M that computes V. Let us call its description x' and let $c_M = |x'| + 2$. Suppose

$$\lim_{n\to\infty} \frac{l(n)T(n)}{U(n)} = 0.$$

By diagonalization M must use at least $T(n)$ steps on input x if x' appears as a prefix of any string y_i and x is sufficiently long. Otherwise M makes an error in the ith output bit. When $m \geq c_M$ the probability that this will happen is $1 - (1 - 2^{-c_M})^l$. This proves the following theorem.

Theorem 6 *Suppose $T(n) > n$ and $U(n)$ are time-constructible and*

$$\lim_{n\to\infty} \frac{l(n)T(n)}{U(n)} = 0.$$

Suppose furthermore that $l(n)$ is computable in time $U(n)$, then there is a function f computable in time $O(U(n))$ on a multitape Turing machine such that any two-tape machine that computes f must use time at least T on a fraction $1 - c^{-l(n)}$ of the inputs. Here $c < 1$ is a constant that depends on the machine computing f.

Again using Theorem 2 we get an immediate corollary:

Theorem 7 *Suppose $T(n) > n$ and $U(n)$ are time-constructible and*

$$\lim_{n\to\infty} \frac{l(n)T(n)\log T(n)}{U(n)} = 0.$$

Suppose furthermore that $l(n)$ is computable in time $U(n)$, then there is a function f computable in time $O(U(n))$ on a multitape Turing machine such that any Turing machine that computes f must use time at least T on a fraction $1 - c^{-l(n)}$ of the inputs. Here $c < 1$ is a constant that depends on the machine computing f.

4 Bounds by Kolmogorov complexity

To get a stronger hierarchy for languages we will need time bounded Kolmogorov complexity.

Definition 2 *A string x has* time bounded Kolmogorov complexity $s = K^T(x)$ *if the smallest two-tape Turing machine that gives output x in at most $T(|x|)$ steps is of length s. We also say that x has conditional complexity $s = K^T(x|y)$ if s is the length of the smallest two-tape Turing machine that produces x in $T(|x| + |y|)$ steps when given y as input.*

We will show that if a certain function can be computed fast on more than a small fraction of the inputs, then we can write a short and fast program that outputs a string with high complexity. Thus, we get a contradiction. First we define what we mean by high complexity.

Definition 3 *A string x is T-y-*slightly random *if the complexity $K^T(x|y) > \log |x|$.*

Remark: For the proof to work, the lower bound on the Kolmogorov complexity need just go to infinity with the length of x.

We fix a time function T and a constant c. In what follows we break the input into two parts and refer to it as a pair (x, y), with the input length being $n = |x| + |y|$, and x being the first $\lfloor c \log n \rfloor$ bits of the input. Let us look at the following language:

$$L_c^T = \{ (x, y) \mid x \text{ is } T\text{-}y\text{-slightly random} \}.$$

Our goal is to show that the characteristic function of L_c^T requires long time on most pairs (x, y).

Definition 4

$$\chi_{L_c^T}(x, y) = \begin{cases} 1 & \textit{if } x \textit{ is } T\textit{-}y\textit{-slightly random,} \\ 0 & \textit{otherwise.} \end{cases}$$

We observe that if T is time-constructible then $\chi_{L_c^T}(x, y)$ can be computed on a multitape Turing machine in time $O(T(n) \log n (\log \log n)^2)$. Just run all programs of length at most $\log |x|$ for $T(n)$ steps and see if one of them produces x. There are $c \log n$ such programs and the factor of $(\log \log n)^2$ comes from Lemma 3. Please note that we require the machine in the definition of the Kolmogorov complexity to have two tapes which makes the simulation possible.

We have the following result:

Theorem 8 *Suppose $T'(n) > n$ is time-constructible and let c be an arbitrary constant. Let T be a function such that*

$$\lim_{n \to \infty} \frac{n^c T'(n) \log T'(n)}{T(n)} = 0.$$

Then any multitape Turing machine that computes the characteristic function $\chi_{L_c^T}(x, y)$ must use more than $T'(n)$ steps on a fraction $1 - 2n^{-c} c \log n$ of all inputs.

Theorem 8 follows immediately from the following lemma:

Lemma 9 *Let c, T', and T be as in the statement of Theorem 8. Fix a multitape Turing machine M that computes $\chi_{L_c^T}(x, y)$. For any fixed sufficiently long y_0 M must take more than $T'(n)$ steps for a fraction $\geq 1 - 2n^{-c} c \log n$ of the x's.*

Proof: Assume that we have some y_0 for which the lemma does not hold. That is, we can compute $\chi_{L_c^T}(x, y_0)$ in time $T'(n)$ for more than a fraction $2n^{-c}c\log n$ of the x's. Consider the following program.

$P(y)$:

1. Find l so that $l = \lfloor c\log(l + |y|)\rfloor$
2. For all $x, |x| = l$, simulate M on input (x, y) and stop after $T'(n)$ steps of the simulated machine ($n = |x| + |y|$).
3. Output the smallest x that gives $\chi_{L_c^T}(x, y) = 1$ in step 2 or if no such x is found, then output $000\ldots0$

Let us see what P does on input y_0. First observe that, using Lemma 3, it runs in time $O(n^cT'(n))$ (since M is a fixed machine there is only constant overhead in the simulation). Thus, P can be computed by a two-tape Turing machine in time $O(n^cT'(n)\log T'(n))$. The output will be some x_0 such that $\chi_{L_c^T}(x_0, y_0) = 1$. Why is that?

By Theorem 2 and the assumption on the running time, we get, for sufficiently long y_0, the value of $\chi_{L_c^T}(x, y_0)$ for more than $2^{c\log n}2n^{-c}c\log n = 2c\log n$ of the x's. One of these must be T-y_0-slightly random. This follows since we have only $2c\log n$ short programs and more than $2c\log n$ strings. Thus, we always end up in the first case of step 3 (at least for sufficiently large n). So on input y_0 the program P outputs a T-y_0-slightly random string. Since P has a description of constant length and runs in time less than $T(n)$ on a two-tape Turing machine (for n sufficiently large), we have a contradiction. ∎

This gives the following hierarchy result:

Theorem 10 *Let c be a constant and let $T(n)$ be a time-constructible time bound. Then there is a language L_c^T which can be recognized in time $O(T(n)\log n(\log\log n)^2)$ on a multitape Turing machine. On the other hand any multitape Turing machine that recognizes L_c^T must use time $\Omega\left(T(n)/(\log T(n)n^c)\right)$ on a fraction $1 - 2n^{-c}c\log n$ of all inputs.*

5 Conclusions

If we require most inputs to be "hard" neither of the two hierarchies is as tight as the standard worst case hierarchy. Since Levin gives an average case hierarchy that is as tight as the worst case hierarchy [Lev73], our results are lacking in this respect. However, we feel that the fact that our functions are fairly standard and quite explicit has given us a greater understanding of these problems. Naturally one would like to obtains as strong results as Levin's by more explicit methods. We believe this is an important open problem.

Acknowledgment: We are grateful to Leonid Levin for telling us about his results and pointing us to the reference [Lev73]. We also thank the referees for their helpful comments and observations. Many thanks are due to Lane Hemachandra who, with his own comments and by relating to us comments form Joel Seiferas and Albert Meyer, helped clarify the history of average-case hierarchies.

References

[Blu67] M. Blum. A machine-independent theory of the complexity of recursive functions. *J. ACM*, 14(2):322–336, 1967.

[Gur91] Y. Gurevich. Average case completeness. *J. of Computer and System Sciences*, 42:346–398, 1991.

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[HS66] F. C. Hennie and R. E. Stearns. Two tape simulation of multitape machines. *Journal of the Association for Computing Machinery*, 13(4):533–546, 1966.

[Lev73] L. Levin. On storage capacity for algorithms. *Soviet Mathematics, Doklady*, 14(5):1464–1466, 1973.

[Lev86] L. Levin. Average case complete problems. *SIAM Journal of Computing*, 15:285–286, 1986.

[Rab60] M. O. Rabin. Degree of difficulty of computing a function and partial ordering of recursive sets. Technical Report 2, Hebrew U., Jerusalem, Israel, 1960.

[Tse56] G.S. Tseitin. seminr on math. logic, moscow university, 11/11, 11/21, 1956.

[Yan59] S. A. Yanovskaya. Math. logic and foundations of math. In *Math. in the USSR for 40 years*, pages 1:13–120, 1959. (in Russian).