The Complexity of Malign Ensembles

Peter Bro Miltersen
Computer Science Department,
Aarhus University,
Ny Munkegade, DK 8000 Aarhus C.
bromille@daimi.aau.dk

Abstract
We analyse the concept of malignness, which is the property of probability ensembles of making the average case running time equal to the worst case running time for a class of algorithms. We derive lower and upper bounds on the complexity of malign ensembles, which are tight for exponential time algorithms, and which show that no polynomial time computable malign ensemble exists for the class of polynomial time algorithms. Furthermore, we show that for no class of superlinear algorithms a polynomial time computable malign ensemble exists, unless every language in \( P \) has an expected polynomial time constructor.

1 Introduction
The average case time complexity of specific algorithms has for a number of years been an active area of research, often showing significant improvement over the worst case complexity when specific distributions of the inputs were assumed. Recently, Li and Vitányi [4] studied the Solomonoff-Levin measure \( m \) and found that when the inputs to any algorithm are distributed according to this measure, it holds that the algorithm's average case complexity is of the same order of magnitude as its worst case complexity. More precisely, for some \( c > 0 \)

\[
\sum_{x \in \mathbb{N}^n} \frac{m(x)}{\sum_{y \in \mathbb{N}^n} m(y)} T_A(x) \geq c \max_{x \in \mathbb{N}^n} T_A(x).
\]

In this paper, we use the term malign for such a measure. This result seems interesting for at least two reasons.

1. The Solomonoff-Levin measure assigns large amounts of probability to strings with lots of pattern, and small amounts to random strings. Therefore, the result seems to imply that worst case strings or strings close to worst case will be easily describable. In [4], the example of quicksort is given, where the worst case strings are the sorted or almost sorted ones. These have short descriptions, and hence high Solomonoff-Levin measure. That worst case strings in general are easily describable seems counterintuitive.

2. Li and Vitányi argue in [5] that the Solomonoff-Levin measure should be used as the a priori probability in Bayesian reasoning, because, in a certain sense, it lies close to any r.e. measure (it dominates them multiplicatively). Similarly, one could argue that if inputs are given from a natural source, the results imply that the average case time will be close to the worst case time, so that no improvement is possible.

However, since the Solomonoff-Levin measure is not even recursive, their result fits poorly with the traditional complexity theoretic view on average case complexity, founded by Levin in [3] and extended in [2] and [1]. In Levin's approach to average case complexity, the distribution function of the input measure is required to be polynomial time computable. Therefore, unless we can derive some kind of time bounded version of Li and Vitányi's result, the two subjects seem quite unrelated, although some of the consequences are similar [4]. In this paper, we analyse from a complexity-theoretic perspective the property of malignness. For this, we restate Li and Vitányi's result and give a simple proof. It seems that the counterintuitive property of malignness is dependent upon an exponential time pattern, which makes the above interpretations less obvious. We present a number of results which support
this. Our results pose a limit on the results achievable in the average case direction by the time bounded versions of the Solomonoff-Levin measure, which are also discussed in [4]. In general, they suggest that if inputs are given by any polynomial time adversary, phenomena like the above will not arise.

2 Notation

- We consider the fixed binary alphabet \( \Sigma = \{0, 1\} \). 
- \( \Sigma^* \) is the set of binary strings with the usual ordering, first by length and then lexicographically, and \( \Sigma^n \) is the set of strings of length \( n \). By \( x-1 \) we denote the string preceding \( x \), and by \( x+1 \) the string following \( x \). The empty string is denoted \( \Lambda \).
- The length of the string \( x \) is denoted by \( |x| \).

A measure \( \mu \) on a finite or countable set \( M \) is a function from \( M \) to the real numbers, with \( \mu(x) \geq 0 \) for all \( x \). Given a subset \( A \subseteq M \), we define

\[
\mu(A) = \sum_{x \in A} \mu(x).
\]

A probability measure is a measure with

\[
\sum_{x \in M} \mu(x) = 1.
\]

Given a measure \( \mu \) on \( \Sigma^* \) or \( \Sigma^n \), we denote by \( \mu^* \) its distribution function \( \mu^*(x) = \mu(\{y | y \leq x\}) \). If \( \mu(\Sigma^n) \neq 0 \), the \( n \)th derived measure of \( \mu \) is the probability measure on \( \Sigma^n \) defined by

\[
\mu_n(x) = \frac{\mu(x)}{\mu(\Sigma^n)}.
\]

- A function \( f : \Sigma^* \to R \) is called polynomial time computable if there exists a polynomial time Turing machine which on input \( <x, 1^t> \) produces the binary notation of an integer \( t \) with

\[
|f(x) - t2^{-i}| \leq 2^{-i}.
\]

In general, if the machine runs in time at most \( g(|x|, i) \), we say that \( f \) is computable in time \( g(|x|, i) \).

- By algorithms we mean the algorithms in a fixed machine model, which takes an input \( x \), and always halts. These can not be effectively enumerated of course, but we will assume that all the machines, including those which do not halt on every input, are enumerated \( A_1, A_2, \ldots \) such that simulation, including stepcounting etc., of \( n \) steps of \( A_i \) on input \( x \) can be performed in time \( p(i, |x|, n) \) on a Turing machine, where \( p \) is a polynomial. We assume that algorithms can simulate Turing machines in real time.

- Given an algorithm \( A_i \), we define \( T_A(x), x \in \Sigma^* \) to be its running time on the binary string \( x \). Given a function \( f : N \to N \), we define \( Alg(f) \) to be the class of algorithms with \( T_A(x) \leq f(|x|) \) for all \( x \), except a finite number. \( T_A^o(n) = \max_{|x| = n} T_A(x) \) is the algorithm's worst case running time. We denote by \( w_A(n) \) the lexicographically least string in \( \Sigma^n \) with \( T(w_A(n)) = T_A^o(n) \). \( T_A^o(\mu, n) = \sum_{|x| = n} \mu_n(x)T_A(x) \) is the algorithm's average case running time with respect to the measure \( \mu \).

3 Malign measures

In this section malignness is defined and Li and Vitányi's result on the Solomonoff-Levin measure is presented. We give a direct proof and skip the conceptual developments of [4]. We consider the class of Turing machines, where each machine has three tapes,

- A binary input tape, infinite in one direction, with the restriction that the head can only move in this direction. Thus, the input tape is one-way, one-way infinite.
- A two-way, infinite, work tape.
- A one-way, one-way infinite binary output tape.

The input tape and the output tape are started with their heads on the first square, and a machine must be able to determine by itself when it has read its input. Now consider an effective enumeration \( M_1, M_2, \ldots \) of the above class of Turing machines, and let \( U \) be a machine universal for the class, i.e. on input \( 1^t0t \), where \( t \) is the infinite tape, \( U \) halts if and only if the machine \( M_i \) on input tape \( t \) halts, and in that case \( U \) outputs whatever \( M_i \) outputs.

We next consider the input tape of \( U \) filled with the results of an infinite sequence of coin tosses. The Solomonoff-Levin measure \( m(x) \) of a string \( x \in \Sigma^n \) is then defined as the probability that \( U \) halts, outputting
a. Since $U$ of course has a non-zero probability of not halting, we have that
\[ \sum_{x \in \Sigma^*} m(x) < 1. \]

The Solomonoff-Levin measure was first defined rigorously in [9]. Intuitively, it gives a large amount of measure to strings with lots of pattern, since these have short programs which have a high probability of appearing. Actually, it is closely tied to self-delimiting Kolmogorov complexity of $x$, $K(x)$, since $m(x) = \Theta(2^{-K(x)})$, but we do not need this result here (see [4] for a proof, and [5] and [6] for general discussions of the properties of $m$).

Definition 3.1 A measure $\mu$ is malign for an algorithm $A$ if and only if there exists a $c > 0$ such that for all sufficiently large $n$
\[ T_A^*(\mu, n) \geq c T_A^*(n). \]

It is malign for a class of algorithms $A$ if it is malign for each $A \in A$.

The following is the main result from [4] on average case complexity.

Theorem 3.1 (Li and Vitányi) The Solomonoff-Levin measure $m$ is malign for all algorithms.

Proof Consider the Turing machine $M$ in figure 1. Assume $M = m_i$ in the above enumeration, and assume that $i$ is the index of an algorithm, i.e. that $A_i$ halts on all inputs. If $U$ is started with the tape $1^k01^k01$, where $U$, started on $t$, outputs a string of length $n$, $U$ will output $w_{A_i}(n)$. The events of reading $1^k01^k0$ and reading $t$ are independent. This means that for all $n$
\[ m(w_{A_i}(n)) \geq 2^{-k-i-2} m(\Sigma^n). \]

But then
\[ T_A^*(\mu, n) \geq \frac{m(w_{A_i}(n))}{m(\Sigma^n)} T_A^*(n) \geq 2^{-k-i-2} T_A^*(n). \]

As mentioned in the introduction, theorem 3.1 seems interesting because it suggests that generally inputs which require a lot of time contain lots of pattern - otherwise the average case complexity with respect to the Solomonoff-Levin measure could not be of the same order as the worst case complexity. In [4], Li and Vitányi use quicksort as an example, where the worst case inputs are the ones already sorted - i.e. inputs with lots of pattern. Theorem 3.1 then suggests that this is a general phenomenon. By examining the proof we see that it indeed holds that the worst case input has a lot of pattern. The proof uses that the worst case input to $A_i$ of length $n$ can be described in the following way: "The worst case input to $A_i$ of length $n"$. Thus $w_{A_i}(n)$ has a short description, i.e. lots of pattern. Of course, this seems to make our interpretation of the theorem a bit less valid: Worst case inputs have a pattern, but it is the very pattern of being a worst case input. Further reflection of the notion of pattern seems to be called for. The problem with the pattern "The worst case input to $A_i$ of length $n"$ is, of course, that it takes exponential time to get from the description to the result. Thus, the pattern is difficult. The main object of this paper is to establish that this is the way it has to be - in general it does not hold that inputs which are slow to process have an easy pattern. To make this more precise, we want to answer questions of the following kind:

Suppose $\mu$ is malign for a class of algorithms $A$. What complexity does $\mu$ have?

If the complexity must be high, no natural source is likely to be malign, which means that we do not always have to give up doing better in the average case than in the worst case. As in the theory of Average-NP [3][2][1], it seems natural to take the computational complexity of the distribution function $\mu^*$ as the complexity of $\mu$. However, this does not seem to be a good idea in this context without further restrictions on $\mu$, as the following theorem shows.

Theorem 3.2 For each general recursive function $f$ there exists a measure $\mu$ which is malign for $ALG(f)$ and whose distribution function $\mu^*$ is polynomial time computable.
Proof Let $T(x, i, t) = \min(T_A_i(x), t)$ and let

$$w(n, i, t) = \min(y \in \Sigma^n | \forall z \in \Sigma^n : T(y, i, t) \geq T(x, i, t)),$$

i.e. $w(n, i, t)$ is the lexicographically least case input of $A_i$ of size $n$, when $A_i$ is restricted to run for at most $t$ time steps. Let $g$ be a time constructible function with $g(n) \geq f(n)$ for all $n$ and let $v(n, i) = w(n, i, g(n))$. Observe that $v(n, i)$ can be computed in time $t(n, i) = q(2^n p(n, i, g(n)))$ where $p$ and $q$ are polynomials, by simulating $A_i$ on all inputs of length $n$. Let $u(n) = t(n, n)$. By choosing $p$ and $q$ appropriately (sufficiently large), $u$ can be made time constructible. We can without loss of generality assume that $u(n + 1) \geq u(n) + n$. Now define

$$b(x, i) = \begin{cases} 1 & \text{if } u(|x|) < i \leq u(|x|) + |x| \\
0 & \text{otherwise} \end{cases}$$

and let $\mu(x) = \sum_{i=0}^{\infty} b(x, i)2^{-i}$. Since $v(n, j) \in \Sigma^n$, we have

$$\mu(\Sigma^n) = \sum_{i=v(n)+1}^{\infty} 2^{-i} < 2^{-u(n)}.$$

Fix an algorithm $A_j \in ALG(f)$. We have that $v(n, j) = w_{A_j}(n)$ for sufficiently large $n$. Then

$$b(w_{A_j}(n), u(n) + j) = 1 \text{ for } n \geq j$$

and therefore

$$\mu(w_{A_j}(n)) \geq 2^{-n(n) - j}.$$
w(n, i) = min\{x \in \Sigma^n | \\
\forall y \in \Sigma^n : \mathcal{T}(y, i) \leq \mathcal{T}(x, i)\}.

w can be computed in time \(p_2(2^n, p_2(f(n), i))\) i.e. in time \(p_9(2^n f(n), i)\), where \(p_2\) and \(p_9\) are polynomials.

\[ b(x, i) = \begin{cases} 1 & \text{if } w(|x|, i) = x \\ 0 & \text{otherwise} \end{cases} \]

\[ \mu_{m_1}(x) = \sum_{i=0}^{\infty} b(x, i)2^{-i}. \]

\( \mu \) is a probability ensemble. It can be computed in the required time, because the \( i \)th binary digit of \( \mu_{m_1}(x) \) is 1 if and only if \( w(|x|, i) \leq x \). It only remains to show that it is malign for \( \text{ALG}(f) \). But for this we observe that if \( A_k \) is such an algorithm, \( \mathcal{T}(x, j) = T_{A_k}(x) \) for sufficiently large \(|x|\), and for these \( x \), \( b(x, j) = 1 \) if \( x = \text{the lexicographically least worst case input of size } |x| \) for \( A_j \). But then

\[ T_{A_k}(\mu, n) \geq 2^{-j} T_{A_k}(n) \]

for sufficiently large \( n \).

\[ \square \]

Of course, if \( f \in \Sigma(n) \), the factor \( 2^{\omega(n)} \) can be omitted from the stated time, i.e. for classes of exponential time algorithms, we can compute the ensemble almost as fast as the algorithms run.

Corollary 4.1. There exists an ensemble \( \mu \), computable in time \( p(2^{\omega(n)}, i) \), where \( p \) is a polynomial, so that \( \mu \) is malign for the class of polynomial time algorithms.

Theorem 4.1 and the corollary reflect our intuition from the proof of theorem 3.1: Malignness can be obtained if we are willing to use exponential time. By using the same technique, we can provide a recursive measure that is malign for classes of algorithms with some recursive upper bound on their running time. It won't provide us with a recursive measure for the class of all algorithms, and, as is mentioned below, no such thing exists. We now turn to a negative result, complementing theorem 4.1.

Theorem 4.2. There is an \( \epsilon > 0 \) and a polynomial \( p \), such that for all nondecreasing time constructible function \( f \) with \( f \in \Omega(p) \), there is no ensemble \( \mu \), malign for \( \text{ALG}(f) \) and computable in time \( f(|x|)^{\epsilon} h(i) \), where \( h \) is any function.

Proof. The proof uses our ability to do a binary search for a string of low \( \mu \)-measure. Given a nondecreasing, time constructible function \( g \), \( g \in \Omega(n) \) and any function \( h \), consider an ensemble \( \nu \), computable in time \( g(|x|)h(i) \). We may assume that \( h \) is recursive, since \( h(i) \) otherwise can be replaced with \( \max_n \mathcal{T}(\nu(n), i) \) which is recursive. We may furthermore assume that \( h \) is time constructible, strictly increasing and tends to infinity, since any general recursive function is dominated by such a function. Define

\[ \overline{h}(n) = \min(\max(n, \max(j|h(j) < \log(n)\}), n). \]

By \( h \)'s time constructibility, \( \overline{h} \) can be computed in polynomial time. Furthermore, the polynomial time bound does not depend upon \( h \). Consider the algorithm \( B \) in figure 2. The function \( q \) in the algorithm

\[ \begin{align*}
\text{input } x \\
y := 1 \\
\text{for } i := 1 \text{ to } h(|x|) \text{ do} \\
v_1 := \mu_{|x|}(x < y0(1-\epsilon)) + \epsilon_1 \\
v_2 := \mu_{|x|}(y0(1-\epsilon)) + \epsilon_2 \\
v_3 := \mu_{|x|}(y1(1-\epsilon)) + \epsilon_3 \\
&\quad (\epsilon_j \leq 2^{-i}) \\
&\quad \text{if } v_3 - v_1 \leq v_3 - v_2 \text{ then} \\
&\quad y := y0 \\
&\quad \text{else} \\
&\quad y := y1 \\
&\text{od} \\
y := y0(1-\epsilon) \\
\text{if } x = y \text{ then} \\
\text{idle for } q(g(|x|)\log(|x|))^2 \text{ time-steps.} \\
\end{align*} \]

Figure 2: Algorithm \( B \)

should be a polynomial such that \( q(g(|x|)\log(|x|)) \) is an upper bound for the running time of the algorithm when \( x \neq y \). Observe that \( q \) can be picked independently of \( \nu \) and \( h \). We may assume that \( q \) is of the form \( q(t) = t^q \). Put \( \epsilon = \frac{1}{\log q} \). Putting \( g(n) = f(n)^q \), we have that \( B \) halts on almost all inputs in time \( q(g(|x|)f(|x|)^q = \log(|x|)^2 f(|x|)^q \) which is less than \( f(|x|) \) for almost all \(|x| \). Thus, \( B \in \text{ALG}(f) \).

By an induction, the invariant

\[ \mu_{|x|}(x \in \Sigma^{|x|-\epsilon}) \leq \left( \frac{3}{4} \right)^{|x|-\epsilon} \]
holds at the end of the i'th cycle of the for loop, so the y found by the for loop has

\[ \mu_{|z|}(y) \leq \left( \frac{3}{4} \right)^{h(|z|)} - 3. \]

We then have

\[ T_{\beta}(n) \leq \left( \frac{3}{4} \right)^{h(n)} - 5T_{\beta}(n) + \sqrt{T_{\beta}(n)}. \]

But this is smaller than cT_{\beta}(n), for any c > 0, for sufficiently large n, i.e. \( \mu \) is not malign for \( B \), which was to be proved.

\[ \square \]

For exponential time algorithms the lower bound matches the upper bound of theorem 4.1 within a polynomial. By a technique similar to the above proof, we can also prove that no recursive measure is malign for the class of all algorithms. This implies that the Solomonoff-Levin measure is not recursive (of course, we could have proven this in a more direct way, see [6]).

For polynomial time algorithms, we have

\[ 9 \]

Corollary 4.2 No ensemble \( \mu \in PE \) is malign for the class of polynomial time algorithms.

5 Malign ensembles for classes of fast algorithms

Corollary 4.2 still leaves something to be desired. After all, most algorithms we are likely to run will be in e.g. \( ALG(n^k) \). It still seems that exponential time is required to compute malign ensembles for such classes. However, we are not likely to be able to prove these intuitions correct, because the following theorem tells us that in order to show that superpolynomial time is necessary, we would have to prove \( P \neq NP \). We are reusing the technique from the previous constructions.

Theorem 5.1 For all \( k \), an ensemble \( \mu \in PE^{2k} \) exists, which is malign for \( ALG(n^k) \).

Proof Put \( f(n) = n^k \) in the proof of theorem 4.1. This makes \( T(x,i) \) polynomial time computable. Observe that the i'th binary digit of \( \mu_{|x|}(x) \) is 1 if and only if

\[ \exists y \in \Sigma^* \forall z \in \Sigma^n : y \preceq z \land T(z,i) \leq T(y,i) \]

and this is a \( \Sigma_2^p \)-problem.

\[ \square \]

It thus seems that we will have to concentrate on merely making the existence of such a \( PE \)-ensemble unlikely, instead of trying to prove that it does not exist. For this, we need a result on sampling.

Definition 5.1 An ensemble \( \mu \) is polynomial time samplable if a polynomial time probabilistic Turing machine exists, which on input \( < 1^n, 1^i > \) produces a string of length \( n \), \( M(n,i) \) such that for all \( x \in \Sigma^n \)

\[ | \text{Pr}(M(n,i) = x) - \mu_n(x) | \leq 2^{-i} \]

A similar definition and an analogy to the following theorem can be found in [1].

Theorem 5.2 Every ensemble \( \mu \in PE \) is polynomial time samplable.

Proof The proof is similar to the one in [1]. First, we construct a probabilistic algorithm \( A \) with no time bound, whose outputs follow the distribution \( \mu_{|x|} \) exactly. Given a finite string \( b = b_1b_2 \ldots b_l \) we denote by \( \text{val}(b) \) the number \( \sum_{i=1}^{l} b_i2^{-i} \). Given a one-way infinite tape \( b = b_1b_2 \ldots \) we denote by \( \text{val}(b) \) the number \( \sum_{i=1}^{\infty} b_i2^{-i} \). That is, \( \text{val}(b) \) is \( b \) read as the binary expansion of a real in the interval \([0, 1] \). Clearly, if \( b \) is a random tape, the random variable \( \text{val}(b) \) is uniformly distributed in \([0, 1] \).

Given a random tape \( b \) and an input \( n \), the algorithm \( A \) in figure 3 is easily seen to determine a string \( x \), such that

\[ \text{val}(b) \in (\mu^*_n(x-1), \mu^*_n(x)). \]

We will furthermore show that it succeeds with probability 1, i.e. for all tapes except for a set of measure 0. Observe that any time we enter the repeat loop, there are four equally likely possibilities for extending \( r \). There are therefore also 4 equally likely possibilities for \([v_{min}, v_{max}] \), namely \( \text{val}(r_{min}), \text{val}(r_{max}) + j2^{-l_{max}} \), for \( j \in \{1, 2, 3, 4\} \). But since the values of \( v_{max} \) and \( v_{min} \) are calculated independently of this extension, at most two of the intervals can have \( v_{min} \leq v_{max} \) and \( v_{max} > v_{min} \). Therefore, before each cycle of the repeat loop there is a probability of at least \( \frac{1}{2} \) of leaving the repeat loop after that cycle. We then have that at any time the probability of eventually leaving the current repeat loop is 1, and consequently the probability of halting is 1. Furthermore it is easy to see that a polynomial \( p \) exists so that \( \text{Pr}(T_A(1^n) \geq p(n,i)) \leq 2^{-i} \).

Now consider a probabilistic Turing machine \( M \) which on input \( < 1^n, 1^i > \) simulates \( A \) on input \( 1^n \) for \( p(n,i) \) time steps, outputs whatever \( A \) does if \( A \) halts during this period, and outputs \( 0^n \) otherwise. Clearly, for any \( x \in \Sigma^n \)

\[ | \text{Pr}(M(< 1^n, 1^i >) = x) - \mu_n(x) | \leq 2^{-i}. \]
input $1^n$

for $j := 1$ to $n$
do
repeat
append to $r$ two random bits.

$
\alpha := \mu_\ast(x1^n) \pm \epsilon \\
\mu_{\min} := \alpha - 2^{-n} \\
\mu_{\max} := \alpha + 2^{-n} \\
\nu_{\min} := \nu(r) \\
\nu_{\max} := \nu(r) + 2^{-n} \\

\{ \text{Invariant: } \nu_{\min} \leq \nu(\text{random tape}) \leq \nu_{\max} \}
$
until \mu_{\max} < \nu_{\min} \text{ or } \nu_{\max} \leq \nu_{\min}$

if $\mu_{\max} < \nu_{\min}$ then

$x := x1$

else

$x := x0$

$z1$

{ \text{Invariant: } \nu(\text{random tape}) \\
in (\mu_\ast(y | y < x0^n), \mu_\ast|s1^n-j) \} \\

\text{od}

output $z$

Figure 3: Algorithm $A$

We might note that the converse result holds if and only if $P = \text{Pep}$. 

Definition 5.2 Given a language $L$. An expected polynomial time constructor for $L$ is a probabilistic algorithm which on input $1^n$ produces an $x \in L \cap \Sigma^n$ in expected polynomial time if one exists and otherwise fails to halt.

This is a natural generalisation of the deterministic constructors defined and studied by Sanchis and Fulk in [8]. A useful equivalence is

Lemma 5.1 Every $L \in P$ has an expected polynomial time constructor if and only if every $L \in \text{DTIME}(n)$ has one.

Proof Suppose every $L \in \text{DTIME}(n)$ has a constructor. Let $J$ be a language in $P$. There is a constant $c$, such that $J \in \text{DTIME}(n^c)$. Define $\tilde{J} = \{x10^{[c]} | x \in J \}$. It is easy to see that $\tilde{J} \in \text{DTIME}(n)$, so, by assumption, $\tilde{J}$ has a constructor, $\tilde{C}$. By running $\tilde{C}$ on input $1^n1^n+n$ and extracting the first $n$ digits of the output, we get a string in $L \cap \Sigma^n$.

Theorem 5.3 Suppose a polynomial time samplable
ensemble $\mu$ is malignant for $\text{ALG}(f)$, where $f$ is a time constructible function with $n \in o(f)$. Then every $L \in P$ has an expected polynomial time constructor.

Proof By Lemma 5.1, we only have to show that every $L \in \text{DTIME}(n)$ has a constructor. Consider the Algorithm $A_L$ in figure 4: $A_L$ is an $\text{ALG}(f)$-algorithm, so

input $z$
Decide if $x \in L$.
If it is not, halt immediately.
If it is, wait $f(n)$ time-steps before halting.

Figure 4: Algorithm $A_L$

there exists a $c$ so that $T_M^\ast(\mu, n) \geq cT_M^\ast(n)$. We will show that for sufficiently large $n$,

$L \cap \Sigma^n \neq \emptyset \Rightarrow \mu_\ast(L \cap \Sigma^n) \geq c \frac{n}{2}$

Assume not, i.e. $\mu_\ast(L \cap \Sigma^n) < c \frac{n}{2}$. Then

$\sum_{u \in L \cap \Sigma^n} \mu_\ast(u)T_M(u) < c \frac{T_M^\ast(n)}{2}$

We also have for sufficiently large $n$ that

$\sum_{u \in \Sigma^n \setminus L} \mu_\ast(u)T_M(u) < n$

But then

$c \frac{T_M^\ast(n)}{2} + n \geq T_M^\ast(\mu, n) \geq cT_M^\ast(n)$

If $L \cap \Sigma^n \neq \emptyset$, the algorithm will run for $T_M^\ast(n)$ time-steps for some value of $x$. Therefore

$n \geq c \frac{T_M^\ast(n)}{2} \geq \frac{c}{4}f(n)$

which is a contradiction for sufficiently large $n$. Let $M$ be a polynomial time sampler for $\mu$. By the definition of sampling, if $L \cap \Sigma^n \neq \emptyset$ then $Pr(M(n, f(n) \log f(n)+n) \in L \cap \Sigma^n) \geq \frac{n}{2} - \frac{n}{4} = \frac{n}{4}$. Thus, running $M$ several times on input $<1^n, 1^{f(n)+n^2}>$ until an element of $L$ is produced is a construction of such an element in expected polynomial time.

The following theorem (analog to proposition 4.1 in [8]) makes expected polynomial time constructors for all languages in $P$ unlikely.

170
Definition 6.3 Let \( RE \) be the class of languages \( L \) for which there exist a probabilistic Turing machine, running in time \( 2^{cn} \) on input \( x \), rejects if \( x \not\in L \), and accepts with probability at least \( \frac{1}{2} \) if \( x \in L \).

Thus, \( RE \) is for \( E = \cup_{c \geq 0} \text{DTIME}(2^{cn}) \) what \( RP \) is for \( P \). Hence, \( RE \) should be considered a rather small extension of \( E \), and \( RE = NE \) if and only if there are no sparse languages in \( NP - RP \).

Theorem 6.4 If every \( L \in P \) has an expected polynomial time constructor, then \( RE = NE \).

Proof Let \( L \) be a language in \( NE \), and let \( M \) be a nondeterministic machine, running in time \( 2^{cn} \) and recognizing \( L \). We represent the nondeterministic choices of \( M \) on input \( z \) as binary strings of length \( 2^{cn+1} \). Denote by \( z_i \), the lexicographically \( i \)th string of size \( n \). Define \( f : \Sigma^* \to N \) by \( f(z_i) = 2^{cn} + i \). The function \( f \) is clearly injective, provided \( c \geq 1 \), which we may assume. Now consider \( \tilde{L} = \{ y \mid \exists z : |y| = f(z) \text{ and } y \text{ codes an accepting computation of } M \text{ on } z \} \). Clearly \( \tilde{L} \in P \) and has therefore, by assumption, an expected polynomial time constructor, \( C \). Let \( p(n) \) be an upper bound on \( C \)'s expected running time on inputs of size \( n \). If we simulate \( C \) on \( 1^{f(x)} \) for \( 2p(f(x)) \) time steps and accept if an element has been produced by then and reject otherwise, we have a \( RE \)-acceptor for \( L \).

\( \square \)

Corollary 5.1 If an ensemble \( \mu \in PE \) is malign for \( ALO(f) \), where \( f \) is time constructible and \( n \in o(f) \), then \( RE = NE \).

Acknowledgments

Thanks to Joan Boyar for her helpful supervision on the thesis on which this paper is based and to Sven Skyum for useful comments and suggestions.

References


