Introduction to Complexity Theory*
Lecture 24: Average Case Complexity

Notes taken by Tzvika Hartman and Hillel Kugler

July 6, 1999

Summary: We introduce a theory of average case complexity. We define the notion of a distribution function and the classes P-computable and P-samplable of distributions. We prove that P-computable $\subset$ P-samplable (strict containment). The class DistNP, which is the distributional analogue of NP, is defined. We introduce the definition of polynomial on the average and discuss the weaknesses of an alternative definition. The notion of reductions between distributional problems is presented. Finally, we prove the existence of a problem that is complete for DistNP.

1 Introduction

Traditionally, in theoretical computer science, the emphasis of the research is on the worst case complexity of problems. However, one may think that the more natural (and practical) way to measure the complexity of a problem is by considering its average case complexity. Many important problems were found to be NP-complete and there is little hope to solve them efficiently in the worst case. In these cases it would be useful if we could develop algorithms that solve them efficiently on the average. This is the main motivation for the theory of average case complexity.

When discussing the average case complexity of a problem we must specify the distribution from which the instances of the problem are taken. It is possible that the same problem is efficiently solvable on the average with respect to one distribution, but hard on the average with respect to another. One may think that it is enough to consider the most natural distribution - the uniform one. However, in many cases it is more realistic to assume settings in which some instances are more probable than others (e.g. some graph problems are most interesting on dense graphs, hence, the uniform distribution is not relevant).

It is interesting to compare the average case complexity theory with cryptography theory, since they deal with similar issues. One difference between the two theories is that in cryptography we deal with problems that are difficult on the average, while in the average case theory we try to find problems that are easy on the average. Another difference is that in cryptography we need problems for which it is possible to generate efficiently instance-solution pairs such that solving the problem given only the instance is hard. In contrast, this property is not required in average case complexity theory.

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

2 Definitions

2.1 Distributions

We now introduce the notion of a distribution function. We assume a canonical order of the binary strings (e.g. the standard lexicographic order). The notation $x < y$ means that the string x precedes y in this order and $x - 1$ denotes the immediate predecessor of x.

Definition 1 (Probability Distribution Function) A *distribution function* $\mu : \{0,1\}^* \to [0,1]$ is a non-negative and non-decreasing function (i.e., $\mu(0) \geq 0, \mu(x) \leq \mu(y)$ for each $x < y$) from strings to the unit interval $[0,1]$ which converges to one (i.e., $lim_{x\to\infty}\mu(x) = 1$). The *density function* associated with the distribution function μ is denoted by μ' and defined by $\mu'(0) = \mu(0)$ and $\mu'(x) = \mu(x) - \mu(x-1)$ for every $x > 0$.

Clearly, $\mu(x) = \sum_{y\leq x} \mu'(y)$. Notice that we defined a single distribution on all inputs of all sizes, rather than ensembles of finite distributions (each ranging over fixed length strings). This makes the definition robust under different representations. An important example is the uniform distribution function defined by $\mu'_u \stackrel{\text{def}}{=} \frac{1}{|x|^2} \cdot 2^{-|x|}$. This density function converges to some constant different than 1. A minor modification, defining $\mu'_u \stackrel{\text{def}}{=} \frac{1}{|x|\cdot(|x+1|)} \cdot 2^{-|x|}$, settles this problem:

$$\sum_{x\in\{0,1\}^*} \mu'_u(x) = \sum_{x\in\{0,1\}^*} \frac{1}{|x|\cdot|x|+1} \cdot 2^{-|x|} = \sum_{n\in N} \sum_{x\in\{0,1\}^n} (\frac{1}{n} - \frac{1}{n+1}) \cdot 2^{-n}$$

$$= \sum_{n\in N} (\frac{1}{n} - \frac{1}{n+1}) = 1$$

We will use a notation for the probability mass of a string relative to all strings of equal size: $\mu'_n(x) \stackrel{\text{def}}{=} \frac{\mu'(x)}{\sum_{|y|=|x|} \mu'(y)}$.

2.2 Distributional Problems

Average case complexity is meaningful only if we associate a problem with a specific distribution of its instances. We will consider only decision problems. Similar formulations for search problems can be easily derived.

Definition 2 (Distributional Problem) A *distributional decision problem* is a pair (D, μ), where $D : \{0,1\}^* \to \{0,1\}$ and $\mu : \{0,1\}^* \to [0,1]$ is a distribution function.

2.3 Distributional Classes

Before defining classes of distributional problems we should consider classes of distributions. It is important to restrict the distributions, otherwise, the whole theory collapses to the worst case complexity theory (by choosing distributions that put all the probability mass on the worst cases). We will consider only "simple" distributions in a computational sense.

Definition 3 (P-samplable) A distribution μ is in the class *P-samplable* if there is a probabilistic Turing machine that gets no input and outputs a binary string x with probability $\mu'(x)$, while running in time polynomial in $|x|$.

Definition 4 (P-computable) A distribution μ is in the class *P-computable* if there is a deterministic polynomial time Turing machine that on input x outputs the binary expansion of $\mu(x)$.

Interesting distributions must put noticeable probability mass on long strings (i.e., at least $\frac{1}{poly(n)}$ on strings of length n). Consider to the contrary the density function $\mu'(x) \stackrel{\text{def}}{=} 2^{-3|x|}$. An algorithm of exponential running time, $t(x) = 2^{|x|}$, will be considered to have constant on the average running time with respect to this distribution (since $\sum_x \mu'(x) \cdot t(|x|) = \sum_n 2^{-n} = 1$). Intuitively, this distribution does not make sense since usually the long instances are the difficult ones. By assigning negligible probability to these long instances, we can artificially make the average running time of the algorithm small, even though the algorithm is not efficient. Consider, for example, an extreme case in which all instances of size greater than some constant have zero probability. In this case, every algorithm has average constant running time.

We now show that the uniform distribution is P-computable. For every x,

$$\mu_u(x) = \sum_{y \le x} \mu'_u(y) = \sum_{\substack{y \le x \\ |y|<|x|}} \mu'_u(y) + \sum_{\substack{y \le x \\ |y|=|x|}} \mu'_u(y) = \left(1 - \frac{1}{|x|}\right) + \left(N_x \cdot \frac{1}{|x|(|x|+1)} \cdot 2^{-|x|}\right)$$

where $N_x \stackrel{\text{def}}{=} |\{y \in \{0,1\}^{|x|} : y \le x\}| = 1 + \sum_{i=1}^{|x|} 2^{i-1} \cdot x_i$, where $x = x_n \cdots x_1$. Obviously, this expression can be computed in time polynomial in the length of x.

Proposition 2.1 *P-computable $\subset$ P-samplable (strict containment assuming $\#P \neq P$)*

Proof: We prove the proposition in two steps:

1. **Claim 2.2** *For every distribution μ, if $\mu \in$ P-computable then $\mu \in$ P-samplable.*

   Proof: Let μ be a distribution that is P-computable. We describe an algorithm that samples strings according to the distribution μ, assuming that we can compute μ in polynomial time. Intuitively, we can view the algorithm as picking a random real number r in $[0,1)$ and checking which string $x \in \{0,1\}^*$ satisfies $\mu(x-1) < r \le \mu(x)$. Unfortunately, this is not possible since r has infinite precision. To overcome this problem, we select randomly in each step one bit of the expansion (of r) and stop when we are guaranteed that there is a unique x satisfying the requirement above (i.e., every possible extension will yield the same x).

   The algorithm is an iterative procedure with a stopping condition. In each iteration we select one bit. We view the bits selected so far as the binary expansion of a truncated real number, denoted by t. Now we find the smallest n such that $\mu(1^n) \ge t$. By performing a binary search over all binary strings of length n we find the greatest $x \in \{0,1\}^n$ such that $\mu(x) < t$. At this point we check if $\mu(x+1) \ge t + 2^{-l}$, where l is the length of the binary expansion of t. If so, we halt and output $x+1$. Otherwise, we continue to the next iteration. Obviously, this can be implemented in polynomial time. ■

2. **Claim 2.3** *There exists a distribution which is P-samplable but not P-computable, under the assumption $\#P \neq P$.*

   Proof: Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be an NP relation (we assume for simplicity that R contains only pairs of strings with the equal lengths). We define a distribution function:

$$\mu'(x \cdot \sigma \cdot y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } R(x,y) \neq \sigma \\ \frac{1}{|x|^2 \cdot |y|^2 \cdot 2^{|x|} \cdot 2^{|y|}} & \text{otherwise} \end{cases}$$

for every $x, y \in \{0,1\}^*$ of equal length and $\sigma \in \{0,1\}$. The fact that this distribution function converges to a constant can be easily verified using the fact that for every x, y exactly one of the possible values of σ gives a non-zero probability.

We now show that μ is P-samplable. The algorithm samples uniformly a string of length $2n$, denoted $x \cdot y$, where $|x| = |y| = n$ (recall that the uniform distribution is P-computable and thus P-samplable). Next, σ is defined as $R(x, y)$, which is an NP-relation and therefore can be computed efficiently. The algorithm outputs $x \cdot \sigma \cdot y$.

The next step is to show that if μ is P-computable then #P problems can be solved in polynomial time. The number of NP witnesses for x is given by the following expression:

$$\frac{\mu(x \cdot 1 \cdot 1^n) - \mu(x \cdot 0 \cdot 1^n)}{\frac{1}{n^4 \cdot 2^{2n}}} = (\mu(x \cdot 1 \cdot 1^n) - \mu(x \cdot 0 \cdot 1^n)) \cdot n^4 \cdot 2^{2n}$$

The numerator is the probability mass of all strings that end with an NPwitness for x. By normalizing we get the actual number of witnesses, thus solving the #P problem. ■

The proposition follows directly from the two claims. ■

In the sequel we will focus on the P-computable class.

2.4 Distributional-NP

We now define the average-case analogue to the class NP:

Definition 5 (The class DistNP) A distributional problem (D, μ) belongs to the class *DistNP* if D is an NP-predicate and μ is P-computable. *DistNP* is also denoted $\langle \mathcal{NP}, \mathcal{P} - \mathit{computable} \rangle$.

The class $\langle \mathcal{NP}, \mathcal{P} - \mathit{samplable} \rangle$ is defined similarly.

2.5 Average Polynomial Time

The following definition may seem obscure at first glance. In the appendix we discuss the weaknesses of alternative naive formulation.

Definition 6 (Polynomial on the Average) A problem D is *polynomial on the average* with respect to a distribution μ if there exists an algorithm A that solves D in time $t_A(\cdot)$ and there exists a constant $\epsilon > 0$ such that

$$\sum_{x \in \{0,1\}^*} \mu'(x) \cdot \frac{t_A(x)^\epsilon}{|x|} < \infty$$

A necessary property that a valid definition should have is that a function that is polynomial in the worst case should be polynomial on the average. Assume that x^d bounds the running-time of the problem and let $\epsilon = \frac{1}{d+1}$. This function is polynomial on the average (with respect to any μ) according to Definition 6:

$$\sum_{x \in \{0,1\}^*} \mu'(x) \cdot \frac{t_A(x)^\epsilon}{|x|} \leq \sum_{x \in \{0,1\}^*} \mu'(x) \cdot \frac{|x|^{\frac{d}{d+1}}}{|x|} \leq \sum_{x \in \{0,1\}^*} \mu'(x) = 1 < \infty$$

We will now try to give some intuition for Definition 6. A natural definition for the notion of a function $f(\cdot)$ that is "constant on the average" with respect to the distribution μ is requiring

$$\sum_{x\in\{0,1\}^*} \mu'(x)\cdot f(x) < \infty$$

Using this definition, $g(\cdot)$ is called "linear on the average" if $g(x) = O(f(x)\cdot|x|)$ where $f(\cdot)$ is constant on the average. This implies

$$\sum_{x\in\{0,1\}^*} \mu'(x)\frac{g(x)}{|x|} < \infty$$

A natural extension of this definition for the case of polynomial on the average yields Definition 6.

2.6 Reductions

We now introduce the definition of a reduction of one distributional problem to another. In the worst case reductions, the two requirements are efficiency and validity. In the distributional case we also require that the reduction "preserve" the probability distribution. The purpose of the last requirement is to ensure that the reduction does not map very likely instances of the first problem to rare instances of the second problem. Otherwise, having a polynomial time on the average algorithm for the second distributional problem does not necessary yield such an algorithm for the first distributional problem. This requirement is captured by the domination condition.

Definition 7 (Average Case Reduction) We say that the distributional problem (D_1, μ_1) reduces to (D_2, μ_2) (denote $(D_1, \mu_1) \propto (D_2, \mu_2)$) if there exists a polynomial time computable function f such that

1. *validity*: $x \in D_1$ iff $f(x) \in D_2$.
2. *domination*: There exists a constant $c > 0$ such that for every $y \in \{0,1\}^*$,

$$\sum_{x\in f^{-1}(y)} \mu_1'(x) \leq |y|^c \cdot \mu_2'(y)$$

The following proposition shows that the reduction defined above is adequate:

Proposition 2.4 *If $(D_1, \mu_1) \propto (D_2, \mu_2)$ and (D_2, μ_2) is polynomial on the average then so is (D_1, μ_1).*

See proof in Appendix B.

3 DistNP-complete

In this section we state two theorems regarding DistNP-complete problems and prove the first one.

Theorem 8 *There exists a DistNP-complete problem.*

Theorem 9 *Every problem complete for DistNP is also complete for $\langle\mathcal{NP}, \mathcal{P} - sampable\rangle$.*

Proof: (of first theorem) We first define a distributional version of the *Bounded Halting* problem (for a discussion on the *Bounded Halting* problem see Lecture 2.3). We then show that it is in DistNP-complete.

Definition 10 (Distributional Bounded Halting)

1. *Decision:* $BH(M, x, 1^k) = 1$ iff there exists a computation of the non-deterministic machine M on input x which halts within k steps.

2. *Distribution:*
$$\mu'_{BH}(M, x, 1^k) \stackrel{\text{def}}{=} \frac{1}{|M|^2 \cdot 2^{|M|}} \cdot \frac{1}{|x|^2 \cdot 2^{|x|}} \cdot \frac{1}{k^2}$$

Proving completeness results for distributional problems is more complicated than usual. The difficulty is that we have to reduce all DistNP problems with different distributions to one single distributional problem with a specific distribution. In the worst case version we used the reduction $x \to (M_D, x, 1^{P_D(|x|)})$, where D is the NP problem we want to reduce and M_D is the non-deterministic machine that solves D in time $P_D(n)$ on inputs of length n (see Lecture 2.3). A first attempt is to use exactly this reduction. This reduction is valid for every DistNP problem, but for some distributions it violates the domination condition. Consider for example distributional problems in which the distribution of (infinitely many) strings is much higher than the distribution assigned to them by the uniform distribution. In such cases, the standard reduction maps an instance x having probability mass $\mu'(x) \gg 2^{-|x|}$ to a triple $(M_D, x, 1^{P_D(|x|)})$ with much lighter probability mass (recall that $\mu'_{BH}(M_D, x, 1^{P_D(|x|)}) < 2^{-|x|}$). Thus the domination condition is not satisfied.

The essence of the problem is that μ'_{BH} gives low probability to long strings, whereas an arbitrary distribution can give them high probability. To overcome this problem, we will map long strings with high probability to short strings, which get high probability in μ'_{BH}. We will use an encoding of strings which maps a string x into a code of length bounded above by $\log_2 \frac{1}{\mu'(x)}$. We will use the following technical coding lemma:

Lemma 3.1 (Coding Lemma): *Let μ be a polynomial-time computable distribution function (i.e., $\mu \in \mathcal{P} -$ computable). Then there exists a coding function C_μ satisfying the following three conditions:*

1. Efficient encoding: *The function C_μ is computable in polynomial time.*

2. Unique decoding: *The function C_μ is one-to-one.*

3. Compression: *For every x*
$$|C_\mu(x)| \leq 1 + min\{|x|, \log_2 \frac{1}{\mu'(x)}\}$$

Proof: The function C_μ is defined as follows:
$$C_\mu(x) \stackrel{\text{def}}{=} \begin{cases} 0 \cdot x & \text{if } \mu'(x) \leq 2^{-|x|} \\ 1 \cdot z & \text{otherwise} \end{cases}$$

where z is the longest common prefix of the binary expansions of $\mu(x-1)$ and $\mu(x)$ (e.g., if $\mu(1010) = 0.10000$ and $\mu(1011) = 0.10101$ then $C_\mu(1011) = 110$). The intuition behind this definition is that we want to find uniquely a point with certain precision in the interval between $\mu(x-1)$ and $\mu(x)$. As the length of the interval grows, the precision needed is lower. Recall that the length of the interval is exactly the value of $\mu'(x)$, implying the short encoding of high probability strings.

We now show that $C_\mu(x)$ satisfies the conditions of the lemma:

1. *Efficient encoding:* The efficiency of the encoding follows from the fact that μ is a polynomial time computable function.

2. *Unique decoding:* In the first case, in which $C_\mu(x) = 0 \cdot x$, the unique decoding is obvious. In the second case, in which $C_\mu(x) = 1 \cdot z$, since the intervals are disjoint and $\mu(x-1) < 0.z1 \leq \mu(x)$, every z determines uniquely the encoded x, and the unique decoding conditions follows.

3. *Compression:* In the first case, in which $\mu'(x) \leq 2^{-|x|}$, $|C_\mu(x)| = 1 + |x| \leq 1 + \log_2 \frac{1}{\mu'(x)}$. In the second case, in which $\mu'(x) > 2^{-|x|}$, let $l = |z|$ and $z_1 \cdots z_l$ be the binary representation of z. Then,

$$\mu'(x) = \mu(x) - \mu(x-1) \leq \left(\sum_{i=1}^{l} 2^{-i} z_i + \sum_{i=l+1}^{poly(|x|)} 2^{-i} \right) - \sum_{i=1}^{l} 2^{-i} z_i < 2^{-|z|}$$

So $|z| \leq \log_2 \frac{1}{\mu'(x)}$, and the compression condition follows.

■

Now we use the Coding Lemma to complete the proof of the theorem. We define the following reduction of (D, μ) to (BH, μ_{BH}):

$$x \rightarrow (M_{D,\mu}, C_\mu(x), 1^{P_{D,\mu}(|x|)})$$

where $M_{D,\mu}$ is a non-deterministic machine that on input y guesses non-deterministically x such that $C_\mu(x) = y$ (notice that the non-determinism allows us not to require efficient decoding), and then runs M_D on x. The polynomial $P_{D,\mu}(n)$ is defined as $P_D(n) + P_C(n) + n$, where $P_D(n)$ is a polynomial bounding the running time of M_D on acceptable inputs of length n and $P_C(n)$ is a polynomial bounding the running time of the encoding algorithm.

It remains to show that this reduction satisfies the three requirements.

1. *Efficiency:* The description of $M_{D,\mu}$ is of fixed length and by the coding lemma C_μ is computable by polynomial time. Therefore, the reduction is efficient.

2. *Validity:* By construction of $M_{D,\mu}$ it follows that $D(x) = 1$ if and only if there exists a computation of machine $M_{D,\mu}$ that on input $C_\mu(x)$ halts with output 1 within $P_{D,\mu}(|x|)$ steps.

3. *Domination:* Notice that it suffices to consider instances of *Bounded Halting* which have a preimage under the reduction. Since the coding is one-to-one, each such image has a unique preimage. By the definition of μ_{BH},

$$\mu'_{BH}(M_{D,\mu}, C_\mu(x), 1^{P_{D,\mu}(|x|)}) = c \cdot \frac{1}{P_{D,\mu}(|x|)^2} \cdot \frac{1}{|C_\mu(x)|^2 \cdot 2^{|c_\mu(x)|}}$$

where $c = \frac{1}{|M_{D,\mu}|^2 \cdot 2^{|M_{D,\mu}|}}$ is a constant independent of x. By the compression requirement of the coding lemma,

$$\mu'(x) \leq 2 \cdot 2^{-|C_\mu(x)|}$$

Hence,

$$\mu'_{BH}(M_{D,\mu}, C_\mu(x), 1^{P_{D,\mu}(|x|)}) \geq c \cdot \frac{1}{P_{D,\mu}(|x|)^2} \cdot \frac{1}{|C_\mu(x)|^2} \cdot \frac{\mu'(x)}{2} > \frac{c}{2 \cdot P_{D,\mu}(|x|)^2 \cdot |C_\mu(x)|^2} \cdot \mu'(x)$$

Therefore, the reduction satisfies the requirements and the distributional version of the *Bounded Halting* problem is DistNP-complete.

■

Appendix A : Failure of a naive formulation

We now discuss an alternative definition of the notion of polynomial on the average which seems more natural:

Definition 11 (Naive Formulation of Polynomial on the Average) A problem D is *polynomial on the average* with respect to a distribution μ if there exists an algorithm A that solves D in time $t_A(\cdot)$ and there exists a constant $c > 0$ such that for every n

$$\sum_{x \in \{0,1\}^n} \mu'_n(x) \cdot t_A(x) < n^c$$

There are three main problems with this naive definition:

1. This definition is very dependent on the particular encoding of the problem instance. In this definition, the average is taken over all instances of equal length. Changing the encoding of the problem instances does not preserve the partition of instances according to their length and hence does not preserve the average running time of the algorithm.

2. This definition is not robust under functional composition of algorithms. Namely, if distributional problem A can be solved in average polynomial time given access to an oracle for distributional problem B and B can be solved in average polynomial time, then it does not follow that A can be solved in average polynomial time.

3. This definition is not machine independent, i.e., an algorithm can be polynomial on the average in one reasonable computational model, but hard on the average in another (e.g. the simulation of a two-tape Turing machine on a one-tape Turing machine).

The two last problems stem from the fact that the definition is not closed under application of polynomials. For example, consider a function $t(x)$ defined as follows:

$$t(x) \stackrel{\text{def}}{=} \begin{cases} 2^n & \text{if } x = 1^n \\ n^2 & \text{otherwise} \end{cases}$$

This function is clearly polynomial on the average with respect to the uniform distribution (i.e., for every $x \in \{0,1\}^n, \mu'_n(x) = 2^{-n}$).

This is true since

$$\sum_{x\in\{0,1\}^n} \mu'_n(x)\cdot t(x) = 2^{-n}\cdot 2^n + (1-2^{-n})\cdot n^2 < n^2+1$$

On the other hand,

$$\sum_{x\in\{0,1\}^n} \mu'_n(x)\cdot t^2(x) = 2^{-n}\cdot 2^{2n} + (1-2^{-n})\cdot n^4 > 2^n$$

which implies that the function $t^2(x)$ is not polynomial on the average. This problem does not occur in Definition 6 since if $t(x)$ is polynomial on the average with the constant ϵ then $t^2(x)$ is polynomial on the average with the constant $\frac{\epsilon}{2}$.

Appendix B : Proof Sketch of Proposition 2.4

In this appendix we sketch the proof of proposition 2.4. We first restate the proposition:

Proposition 2.4 : If $(D_1,\mu_1) \propto (D_2,\mu_2)$ and (D_2,μ_2) is polynomial on the average then so is (D_1,μ_1).

The formal proof of the proposition has many technical details, so it will be only sketched. As a warm-up, we will first prove the proposition under some simplifying assumptions. To our belief, this gives intuition to the full proof. The simplifying assumptions, regarding the definition of a reduction, are:

1. There exists a constant c_1 such that for every x, $|f(x)| \le c_1\cdot|x|$, where f is the reduction function.
2. Strong domination condition: There exists a constant c_2 such that for every $y\in\{0,1\}^*$,

$$\sum_{x\in f^{-1}(y)} \mu'_1(x) \le c_2\cdot\mu'_2(y)$$

Proof: (*of simplified version*) The distributional problem (D_2,μ_2) is polynomial on the average implies that there exists an algorithm A_2 and a constant $\epsilon_2>0$ such that

$$(1)\qquad \sum_{x\in\{0,1\}^*} \mu'_2(x)\cdot\frac{t_{A_2}(x)^{\epsilon_2}}{|x|} < \infty$$

We need to prove that (D_1,μ_1) is polynomial on the average, i.e. that there exists an algorithm A_1 and a constant $\epsilon_1>0$ such that

$$(2)\qquad \sum_{x\in\{0,1\}^*} \mu'_1(x)\cdot\frac{t_{A_1}(x)^{\epsilon_1}}{|x|} < \infty$$

The algorithm A_1, when given x, applies the reduction f on x and then applies A_2 on $f(x)$. Therefore, $t_{A_1}(x) = t_f(x) + t_{A_2}(f(x))$, where t_f denotes the time required to compute f. For the sake of simplicity we ignore $t_f(x)$ and assume $t_{A_1}(x) = t_{A_2}(f(x))$. Taking $\epsilon_1 \stackrel{\text{def}}{=} \epsilon_2$ we obtain:

$$\begin{aligned}
\sum_{x\in\{0,1\}^*} \frac{\mu_1'(x)\cdot t_{A_1}(x)^{\epsilon_1}}{|x|} &= \sum_{y\in\{0,1\}^*}\sum_{x\in f^{-1}(y)} \frac{\mu_1'(x)\cdot t_{A_2}(y)^{\epsilon_1}}{|x|} \\
&\leq c_1 \cdot \sum_{y\in\{0,1\}^*} \frac{t_{A_2}(y)^{\epsilon_1}}{|y|} \sum_{x\in f^{-1}(y)} \mu_1'(x) \\
&\leq c_1\cdot c_2 \cdot \sum_{y\in\{0,1\}^*} \frac{t_{A_2}(y)^{\epsilon_2}\cdot \mu_2'(y)}{|y|} < \infty
\end{aligned}$$

The first inequality uses Assumption 1, and the second uses the simplified domination condition (Assumption 2). ■

Sketch of full proof: First we explain how to deal with the technical problem that arises when considering the running time of the reduction, $t_f(x)$, in the running time of algorithm A_1. This technical problem arises many times in the proof and is solved as described below. The fact that for every $\epsilon \leq 1$, $(a+b)^\epsilon \leq a^\epsilon + b^\epsilon$ is used to bound expression (2) by two sums. The first sum contains the factor $t_f(x)$ and can be easily bounded by choosing the appropriate ϵ. The second sum converges as shown in the proof of the simplified version.

In order to prove that expression (2) converges we partition the sum in expression (2) into two sums and show separately that each sum converges. Formally, $y \in \{0,1\}^*$ is called a *bad* y if $\mu_2'(y)\cdot t_{A_2}(y)^{\epsilon_2} \geq |y|$ and is called a *good* y otherwise. We partition the x's according to the "goodness" of their images under f. This induces the following partition into two sums.
Denote

$$B = \sum_{bad\ y}\sum_{x\in f^{-1}(y)} \frac{\mu_1'(x)\cdot t_{A_2}(y)^{\epsilon_1}}{|x|} \qquad\qquad G = \sum_{good\ y}\sum_{x\in f^{-1}(y)} \frac{\mu_1'(x)\cdot t_{A_2}(y)^{\epsilon_1}}{|x|}$$

Then

$$\sum_{x\in\{0,1\}^*} \frac{\mu_1'(x)\cdot t_{A_1}(x)^{\epsilon_1}}{|x|} = \sum_{y}\sum_{x\in f^{-1}(y)} \frac{\mu_1'(x)\cdot t_{A_2}(y)^{\epsilon_1}}{|x|} = B+G$$

The intuition behind this partition is that each bad y contributes large weight (at least one) to the sum in expression (1). The fact that expression (1) converges implies that there is a finite number of bad y's. This is used to show that B converges. For the second sum we again partition it into two sums, G_1 and G_2. The first sum, G_1, consists of good y's for which $t_{A_2}(y)$ is bounded by $p(|x|)$ for every $x \in f^{-1}(x)$ and some polynomial p (that depends on ϵ_2 and the domination constant c). This sum can be bounded by choosing an ϵ_1 so that $t_{A_2}(y)^{\epsilon_1} < |x|$ for any $x \in f^{-1}(y)$. That is,

$$\sum_{y: t_{A_2}(y) < \min_{x\in f^{-1}(y)}\{p(|x|)\}}\ \sum_{x\in f^{-1}(y)} \frac{\mu_1'(x)\cdot t_{A_2}(y)^{\epsilon_1}}{|x|} < \sum_x \mu_1'(x) = 1$$

The second sum, G_2, consists of the rest of the good y's. For each y in this sum, we have $t_{A_2}(y) \geq q(y) \stackrel{\text{def}}{=} \min_{x\in f^{-1}(y)}\{p(|x|)\}$. Note that q grows at least as a power of p (depending on the relation of $|f(x)|$ to $|x|$); that is for some $\epsilon > 0$ we have $q(y) \geq p(|y|^\epsilon)$. By a suitable choice of p and ϵ_1, we have

$$|y|^c \cdot t_{A_2}(y)^{\epsilon_1} = \frac{|y|^c\cdot t_{A_2}(y)^{\epsilon_2}}{p(|y|^\epsilon)^{\epsilon_2-\epsilon_1}} < \frac{t_{A_2}(y)^{\epsilon_2}}{|y|}$$

Thus,

$$\begin{aligned}
\sum_{y:t_{A_2}(y)\geq q(y)} \sum_{x\in f^{-1}(y)} \frac{\mu_1'(x)\cdot t_{A_2}(y)^{\epsilon_1}}{|x|} &\leq \sum_{y:t_{A_2}(y)\geq q(y)} |y|^c \mu_2'(y)\cdot t_{A_2}(y)^{\epsilon_1} \\
&\leq \sum_{y:t_{A_2}(y)\geq q(y)} \frac{\mu_2'(y)\cdot t_{A_2}(y)^{\epsilon_2}}{|y|}
\end{aligned}$$

which converges being a partial sum of (1).

Finally, we choose ϵ_1 to be the minimum of all the ϵ_1's used in the proof to obtain the convergence of the original sum. ■