Average Case Complexity*

Yuri Gurevich†, University of Michigan.

Abstract. We attempt to motivate, justify and survey the average case reduction theory.

1. Introduction

An NP decision problem may be specified by a set D of strings (*instances*) in some alphabet, another set W of strings in some alphabet and a binary relation $R \subseteq W \times D$ that is polynomial time computable and polynomially bounded (which means that the size $|w|$ of w is bounded by a fixed polynomial of the size $|x|$ of x whenever wRx holds). If wRx holds, w is called a *witness* for x. The decision problem specified by D, W and R, call it $\mathrm{DP}(D, W, R)$, may be stated thus: Given an element x of D, determine if there exists $w \in W$ such that wRx holds. The corresponding search problem, call it $\mathrm{SP}(D, W, R)$ may be stated thus: Given an element x of D, determine if there exists a witness w for x and if so then exhibit such a witness.

Problems of the form $\mathrm{SP}(D, W, R)$ may be called NP search problems even though NP is supposed to be a class of decision problems. It will be convenient for us to use the term "an NP problem" to mean an NP decision problem (a genuine NP problem) or its search counterpart. In this talk, we deal only with NP decision and search problems even though methods of the average complexity theory may be applied to wider classes. NP and the class of NP search problems are sufficiently important.

The restriction to the case when instances and witnesses are strings is standard [GJ], even though it means that one may be forced to deal with string encodings of real objects of interest. The reason is that we need a clear computational model. If instances and witnesses are strings, the usual Turing machine model can be used. The size of a string is often taken to be its length, though this requirement can easily be relaxed.

A solution of an NP problem is a feasible decision or search algorithm. The question arises what is feasible. It is common to adopt the thesis

(1.0) Feasible algorithms are exactly polynomial time ones.

which can be split into two parts:

(1.1) Every polynomial time algorithm is feasible.

(1.2) Every feasible algorithm is polynomial time.

We do not believe in a unique feasibility concept; feasibility depends on application. In real time applications or in the case of huge databases (like that of the US Internal Revenue Service), even linear time may be prohibitively expensive. However, one important feasibility concept fits (1.1) well. It is often reasonable to assume that if your customer had time to produce an input object of some size n, then you can afford to spend time n^2 or n^3 etc. on the

*Springer LNCS 510, 1991, 615–628.

†Partially supported by NSF grant CCR 89-04728. Address: EECS Dept., University of Michigan, 1301 Beal Ave., Ann Arbor, MI 48109-2122. Email: gurevich@eecs.umich.edu

object. Of course, this "etc." should be treated with caution. Probably, you will be unable to spend time n^{64}. Fortunately, in practically important cases, these polynomials tend to be reasonable.

P-time (polynomial time) is a very robust and machine-independent concept closed under numerous operations. If you believe that (i) linear time algorithms are feasible and (ii) every algorithm with feasible procedures running in feasible time (counting a procedure call as one step) is feasible, then you are forced to accept all P-time algorithms as feasible. Finding a polynomial-time algorithm for an NP problem often requires ingenuity, whereas an exponential-time algorithm is there for free. Proving that a given NP problem can be solved in polynomial time, you feel that you made a mathematical advance. It seems that this feeling of mathematician's satisfaction contributed to the popularity of the P-time concept. (Notice however that a superpolynomial bound may require ingenuity and be a source of satisfaction as well.)

We do not know very strong arguments in favor of (1.2). Moreover, there are strong arguments against it. The most important one for our purpose in this talk is that sometimes there are satisfying practical ways to cope with hard NP problems in the absence of P-time algorithms. In the case of an optimization problem, one may have a fast approximating algorithm. In the case of a decision or search problem, one may have a decision or search algorithm that is usually fast or almost always fast or fast on average. Leonid Levin proposed one average case approach in [Le1]. His approach is the topic of this talk.

We hope this account is entertaining. It certainly isn't complete. The references include three student papers: [Gr], [Kn] and [Sc].

2. Polynomial on average

In the average case approach, a decision or search problem is supposed to be given together with a probability distribution on instances. A problem with a probability distribution on instances is called *randomized* (or *distributional* [BCGL]). Determining an appropriate distribution is a part of the formalization of the problem in question and it isn't necessarily easy. (The robustness of the average case approach with respect to probability distributions is of some help.) In this talk, we address only the task of solving a given randomized problem. The task is to devise an algorithm that solves the problem quickly on average with respect to the given distribution. No pretense is made that the algorithm is also good for the worst case.

One advantage of the average case approach is that it often works. It was possible *a priori* that, whenever you have a fast on the average algorithm, you also have an algorithm that is fast even in the worst case. This is not at all the case. Sometimes a natural probability distribution makes the randomized problem ridiculously easy. Consider for example the 3-coloring search problem when all graphs of the same size have the same probability. The usual backtracking solves this randomized search problem in (surprise!) at most 197 steps on average, never mind the size of the given instance [Wi]. The reason is that there are very simple and probable witnesses to non-colorability, like a clique of 4. The distribution is greatly biased toward the negative answer. The average time can be further cut down if the algorithm starts with a direct search for such witnesses. There are numerous examples of success in the cases of less biased distributions. This is, however, a topic of a different talk.

One may argue that, in many applications, the average case may be more im-

portant than the worst case. Imagine a factory that produces complete graphs. Each edge of a graph is produced separately using the same technology and there is a fixed probability p for an edge to be faulty. Suppose that, for whatever reason, it happens to be important to know whether a given graph has a hamiltonian circuit composed entirely of faulty edges. There is an algorithm A that solves the hamiltonian search problem with any fixed edge-probability distribution and that has an expected running time linear in the number of vertices [GS]. You may want to use that algorithm and open a hamiltonian shop. There may even be several such factories in your area. A customer will bring you a graph G with some number n of vertices. If it is customary that the charge depends only on the number of vertices, you may charge a fee proportional to the expected running time of A on n-vertex graphs, where the proportionality coefficient is chosen to ensure you a fair profit and competitiveness.

For the sake of fairness, we should mention situations where the failure to solve quickly one instance means the failure of the whole enterprise. Even then the worst case approach may be too conservative. In any case, it seems to us that, in many applications where polynomial time is feasible, polynomial on average time is feasible as well. The question arises what does it mean that a function is polynomial (i.e. polynomially bounded) on average.

Following [BG], we define a *domain* to be a set U (the *universe*) with a function from U to natural numbers (the *size function*) and a probability distribution on U satisfying the following technical restrictions. First, the universe is a set of strings in some alphabet (so that the Turing machine model can be used). Second, there are only finitely many elements of positive probability and any given size. It is not required that the size of a string is necessarily its length. Third, the probability distribution is polynomial time computable (P-time). The notion of P-time distributions was introduced in [Le1] and analyzed to some extent in [Gu1]. The requirement that the distribution is P-time will be discussed and relaxed in Section 6. Meantime, view is as some technical restriction that is often satisfied in practice.

Consider a function T from a domain D to the interval $[0..\infty]$ of the real line extended with ∞. Let $E_n(T)$ be the expectation of T with respect to the conditional probability $\mathbf{P}_n(x) = \mathbf{P}\{x \mid |x| = n\}$. The definition of polynomiality on average seems obvious: T is polynomial on average (relative to D) if

(2.1) $\mathbf{E}_n(T)$ is bounded by a polynomial of n.

Unfortunately, this obvious answer is not satisfying. It is easy to construct D and T such that T is polynomial on average but T^2 is not. It seems reasonable to accept that (i) T is polynomial on average if $\mathbf{E}_n(T) = O(n)$, and (ii) if T is bounded by a polynomial of a function which is polynomial on average then T is polynomial on average. These two assumptions imply that T is polynomial on average if

(2.2) there exists $\varepsilon > 0$ such that $\mathbf{E}_n(T^\varepsilon)$ is bounded by a polynomial of n

which is equivalent to

(2.2′) $(\exists \varepsilon > 0)\ \mathbf{E}_n(T^\varepsilon/n) = O(1)$.

The weak point of condition (2.2) is that it is uniform in n. It is possible, for example, that there exists a set Q of natural numbers such that $T(x)$ is large when $|x| \in Q$ but, for $n \in Q$, the probability that $|x| = n$ is very small. The "official" definition requires a slightly weaker condition.

Definition 2.1. A function T from a domain D to $[0..\infty]$ is polynomial on average (in short, AP) if

(2.3) $(\exists \varepsilon > 0)\ \mathbf{E}(T^\varepsilon/|x|) < \infty$.

Theorem 2.1 [Gu1]. Conditions (2.2) and (2.3) are equivalent if there exists an integer k such that $\mathbf{P}\{x : |x| = n\} > n^{-k}$ for all sufficiently large n with $\mathbf{P}\{x : |x| = n\} > 0$.

A more extensive discussion of the issue of AP functions can be found in [Gu1] and [BCGL]. We will return to that issue in Section 7.

3. A new setting

A *randomized* NP decision or search problem may be specified by a triple (D, W, R) where D, W and R are as above (in Section 1) except that D is a domain now. Randomized decision problems $\mathrm{RDP}(D, W, R)$ and randomized search problems $\mathrm{RSP}(D, W, R)$ play the roles of decision and search problems in the NP theory. The role of NP is played by the class RNP of randomized NP decision problems. The term "an RNP problem" will be used to mean an RNP decision problem or its search counterpart. An algorithm for an RNP problem is considered feasible if it runs in time that is AP relative to D; thus the role of P is played by the class AP of AP-time decidable RNP decision problems. (This is going to be revised.)

It is difficult to exhibit an RNP decision problem that is not AP. (Such a problem exists if $NTime(2^{O(n)}) \neq DTime(2^{O(n)})$ [BCGL].) In particular, the existence of such a problem implies $P \neq NP$. However the reduction theory allows one to exhibit maximally hard RNP problems.

We try to motivate an appropriate reduction notion. Notice that an AP-time function f from a domain D_1 to a domain D_2 need not constitute a reduction of $\Pi_1 = \mathrm{RDP}(D_1, W_1, R_1)$ to $\Pi_2 = \mathrm{RDP}(D_2, W_2, R_2)$ even if $(\exists w)(wR_1x) \leftrightarrow (\exists w)(wR_2(fx))$ for all x in D_1. Why? Suppose that you have a decision algorithm A_2 for Π_2 that runs very fast on average. That decision algorithm can be pulled back by means of f to give the following decision algorithm A_1 for Π_1: Given x, compute $y = f(x)$ and then apply A_2 to y. It may happen unfortunately that the range of f has an unfair share of rare difficult instances of Π_2 and A_1 is not at all AP-time.

It would be natural to say that a function f from D_1 to D_2 *reduces* D_1 to D_2 if

(3.1) f is AP-time, and

(3.2) For every AP function $T : D_2 \to [0..\infty]$, the composition $T \circ f$ is AP.

Because of the universal quantification, however, the requirement (3.2) is not convenient to use. Fortunately, (3.2) can be simplified. Let $\mathbf{P}_i$ be the probability distribution of D_i and let $T_0(y) = \mathbf{P}_1(f^{-1}(y))/\mathbf{P}_2(y)$. Since $\mathbf{E}(T_0) = 1$, we have the following special case of (3.2):

(3.3) $\mathbf{P}_1(f^{-1}(fx))/\mathbf{P}_2(fx)$ is AP.

Theorem 3.1 [BG]. Assume (3.1). Then (3.2) $\leftrightarrow$ (3.3).

Say that D_2 *dominates* D_1 *with respect to* f if (3.3) holds. This is a slight variation on Levin's original definition of domination.

Definition 3.1. A function f from a domain D_1 to a domain D_2 *reduces* D_1 to D_2 if f is AP-time computable and D_2 dominates D_1 with respect to f.

Definition 3.2. $\mathrm{RDP}(D_1, W_1, R_1)$ *reduces* to $\mathrm{RDP}(D_2, W_2, R_2)$ if there exists a reduction f of D_1 to D_2 such that

$(\exists w)(wR_1x) \leftrightarrow (\exists w)(wR_2(fx))$ for all instances $x \in D_1$ of positive probability.

Definition 3.3. $\mathrm{RSP}(D_1, W_1, R_1)$ *reduces* to $\mathrm{RSP}(D_2, W_2, R_2)$ if there exists a reduction f of D_1 to D_2 and a polynomial time computable function g from W_2 to W_1 satistying, for every $x \in D_1$ of positive probability, the following two conditions:

- $(\exists w_1)(w_1R_1x) \longrightarrow (\exists w_2)(w_2R_2(fx))$,
- $w_2R_2(fx) \longrightarrow (gw_2)R_1x$.

Here is one example. Given an arbitrary $\mathrm{RSP}(D, W, R)$, form the direct product W' of W and (the universe of) D. Let $P(w, x) = x$ be the projection function from W' to D, F be the restriction of P to the set $\{(w, x) : \ wRx$, and R' be the graph $\{((w, x), x) : \ wRx\}$ of F. The new problem $\mathrm{RSP}(D, W', R')$ is the problem of inverting F. Given an element x of D, determine whether $F^{-1}(x)$ is nonempty and if so then exhibit an element of $F^{-1}(x)$. Thus, every RNP search problem reduces to a randomized problem of inverting a polynomial-time computable function.

Corollary 3.1. Let Π_1, Π_2 and Π_3 be RNP decision (resp. search) problems.

- If Π_1 reduces to Π_2 and Π_2 reduces to Π_3 then Π_1 reduces to Π_3.
- If Π_1 reduces to Π_2 and there exists an AP-time decision (resp. search) algorithm for Π_2 then there exists an AP-time decision (resp. search) algorithm for Π_1.

Corollary 3.2. Let D_1 and D_2 be two domains with the same universe and the same size function dominating each the other with respect to the identity function. Then any $\mathrm{RDP}(D_1, W, R)$ (resp. $\mathrm{RSP}(D_1, W, R)$) is solvable in AP-time if and only if $\mathrm{RDP}(D_2, W, R)$ (resp. $\mathrm{RSP}(D_2, W, R)$) is so.

Corollary 3.2 witnesses a certain robustness of the average case approach.

Leonid Levin proved that a randomized version of the (bounded) tiling problem is complete for RNP [Le1]. Some additional problems complete for RNP with respect to reductions as defined above were given in [Gu1,Gu2]; in the next section, we describe one of them.

4. One RNP complete problem

In order to generalize the notion of the uniform probability distribution from finite sample spaces to domains, we need to fix some default probability distribution on natural numbers. Let us agree that the default probability of any $n > 1$ is proportional to $n^{-1} \cdot (\log n)^{-2}$. Call a domain D *uniform* if elements of the same size have the same probability distribution and the probability of $\{x : \ |x| = n\}$ is proportional to the default probability of n. Further, define the direct product $D_1 \times D_2$ of domains D_1 and D_2 to be the domain of pairs (a, b), where $a \in D_1$ and $b \in D_2$, such that $|(a, b)| = |a|+|b|$ and $\mathbf{P}(a, b) = \mathbf{P}(a)\times\mathbf{P}(b)$.

Recall that the *modular group* is the multiplicative group of two-by-two integer matrices of determinant 1. In this section, a matrix is an element of the modular group. Make a uniform domain out of the modular group in a natural way. The size $|A|$ of a matrix A may be defined as the number of bits necessary to write the matrix down (using the binary notation for the entries); alternatively, it may be defined as the log of the maximal absolute value of its entries. (Notice that $|A|$ is the size rather than the determinant of a matrix A.)

A matrix pair (B, C) gives rise to an operator $T_{B,C}(X) = BXC$ over the modular group which is linear (even though the modular group is not closed under addition) in the following sense: If $X = \sum Y_i$

then $T_{B,C}(X) = \sum T_{B,C}(Y_i)$. Andreas Blass proved that an arbitrary operator over the modular group is linear if and only if there exist matrices B and C such that either $T(X) = BXC$ for all X or else $T(X)$ is the transpose of BXC for all X. Moreover, any linear operator T uniquely extends to a linear operator on all two-by-two integer (or even complex) matrices; this gives rise to the standard representation of T by a four-by-four integer matrix. The two presentations are polynomial-time computable each from the other. Thus, an appropriate matrix pair (B, C) with one additional bit, indicating whether the transpose is applied, is a natural representation of the corresponding linear operator T. It is natural to define the domain of linear operators as the direct product of two copies of the matrix domain and the uniform domain of two elements. Let σ be a sufficiently large positive integer.

We are ready to define an RNP decision problem called *Matrix Decomposition*. The domain of Matrix Decomposition is the direct product of the matrix domain, σ copies of the domain of linear operators and the domain of natural numbers where $|n| = n$ and $\mathbf{P}(n)$ is the default probability of n. In other words, an instance of Matrix Decomposition comprises a matrix A, a sequence $S = (T_1, \ldots, T_\sigma$ of σ linear orientation preserving operators and the unary notation for a natural number n. The corresponding question is if there a product $P = T_1 \times \ldots \times T_m$ of $m \leq n$ linear operators $T_i \in S$ such that $A = P(1)$.

Theorem 4.1 [Gu3]. Matrix Decomposition is RNP complete.

The prove of RNP hardness consists of the following steps. First, a randomized version of the (bounded) halting problem is proved complete for RNP. This result is implicit in [Le1] and explicit in [Gu1]. Second, the randomized halting problem is reduced to a randomized version of the (bounded) Post Correspondence Problem [Gu1]. Third, the randomized PCP is reduced to Matrix Decomposition [Gu3].

A simpler version of Matrix Decomposition is obtained by making S a sequence of matrices rather than linear operators. The question becomes whether A can be represented as a product of at most n S-matrices. We doubt that the modified problem is complete for RNP, but the similar problem for larger matrices, like 20×20 is complete [Ve].

5. Revision 1: Randomized reductions

The setting of Section 3 turns out to be too restrictive. Call a domain D *flat* if $\mathbf{P}(x)$ is bounded by $2^{-|x|^\varepsilon}$ for some $\varepsilon > 0$. Intuitively, a flat domain is akin to a uniform one. No element has a probability that is too big for its size. Many usual domains are flat. For example, any domain of graphs is flat if the size of a graph is the number of vertices (or the number of vertices plus the number of edges, or the size of the adjacency matrix) and the conditional probability distribution on n-vertex graphs is determined by the edge-probability $p(n)$ with $n^{-2+\delta} < p(n) < 1 - n^{-2+\delta}$ for some constant $\delta > 0$.

Lemma 5.1 [Gu1]. If an RNP decision problem on a flat domain is complete for RNP then deterministic exponential time DTime($\exp(n^{O(1)})$ equals nondeterministic exponential time NTime($\exp(n^{O(1)})$.

Proof Sketch. Turn the given nondeterministic exponential time decision problem D_0 into a very sparse RNP problem (D_1, μ_1) whose positive instances x have enormous (for their size) probabilities. Given such an x, a reduction f of (D_1, μ_1) to a flat RNP decision problem (D, μ) produces an instance $f(x)$ of a high

probability and therefore a small size. An exponential time decision procedure for D together with a polynomial time procedure for computing f give a (deterministic) exponential time decision procedure for D_0. QED

This proof contains no evidence that RNP problems with flat distributions are easy; it is natural to view the lemma as evidence that the reductions of Section 3 are not strong enough. Those reductions are many-one reductions and one obvious move is to generalize them to Turing reductions. However, the lemma survives the generalization. Leonid Levin had a fruitful idea: randomize. Allow the reducing algorithm to flip coins and produce a somewhat random output. Now the proof of the lemma fails: Instead of producing one instance $f(x)$ of a high probability and small size, a randomizing reduction produces a multitude of instances of small probabilities and large sizes.

Using randomizing reductions, Levin and his student Ramarathnam Venkatesan constructed a natural randomized graph-coloring problem complete for RNP [VL]. Their paper demonstrates another reason, a very good one, for using randomizing reductions. Randomizing reductions allow us to use the structure of a random instance of the target problem (the one whose completeness one would like to prove). Given an instance x of the randomized halting problem, the reducing machine of Venkatesan and Levin flips coins to produce a random graph of an appropriate size. Then it massages this graph to make sure that x can be coded in with a sufficiently high probability.

Next, following [BG], we generalize the notion of domain reduction by allowing reductions to flip coins. All reductions as in Section 3 will be called deterministic from now on.

Consider a Turing machine M which takes inputs from a domain D and which can flip a fair coin. M can be viewed as a deterministic machine computing a function $f(x, r)$ where $x \in D$ and r is a sequence of (the outcomes of) coin flips. Call such f a *random* function on D. (One may object to the term "random function" on the grounds that a random function on A should be a randomly chosen function on A. The term "a random function" is fashioned in [BG] after well accepted terms like "a real function". A real function assigns real numbers to elements. A random function assigns random objects to (almost all) elements.) Formally, f is an ordinary (or *deterministic*) function on a different domain D_f that is an extension of D. (D_f has a somewhat unusual size function. Notice that the auxiliary input r does not have the status of the real input x because we are interested in measuring the running time of M in terms of the real input x only. The size of a pair (x, r) in D_f is the size of x in D.)

Random functions compose and give (if you care) a nice category. Say that a random function f on D is AP-time computable if the deterministic function f is AP-time computable with respect to D_f. We say that a domain D' *dominates* a domain D with respect to a random function f from D to D' if D' dominates D_f (in the sense of Section 3) with respect to the deterministic function f from D_f to D'.

Definition 5.1. A *reduction* of a domain D_1 to a domain D_2 is an AP-time random function f from D_1 to D_2 such that D_2 dominates D_1 with respect to f.

Theorem 5.1. Let f be an AP-time random function from a domain D_1 to a domain D_2. Then the following statements are equivalent:

- D_1 is dominated by D_2 with respect to f.
- For every AP-time random function T

from D_2 to $[0..\infty]$, the composition $T \circ f$ is AP.

Corollary. Domain reductions compose.

The question arises when a reduction f of a domain D_1 to a domain D_2 reduces an $\mathrm{RDP}(D_1, W_1, R_1)$ to an $\mathrm{RDP}(D_2, W_2, R_2)$ or reduces an $\mathrm{RSP}(D_1, W_1, R_1)$ to an $\mathrm{RSP}(D_2, W_2, R_2)$. Say that f is (fully) correct if $(\exists w)(wR_1x) \leftrightarrow (\exists w)(wR_2f(x,r))$ for all $x \in D_1$ of positive probability and all r. Should we require that f is fully correct or not? In either case, we must allow – to be consistent – randomizing decision and search algorithms satisfying the corresponding correctness requirement.

It may seem that there is no sense in using fully correct randomizing reductions. Such a reduction can be made deterministic by pretending that all coins come up heads. This may ruin the domination requirement however. Fully correct reductions may be employed to overcome the phenomenon of flatness [Gu1, Gu3].

It is much more fruitful though to allow partially correct reductions [VL, BCGL, IL]. Say that a reduction $f(x.r)$ of D_1 to D_2 *reduces* an $\mathrm{RDP}(D_1, W_1, R_1)$ to an $\mathrm{RDP}(D_2, W_2, R_2)$ with probability guarantee $\alpha(n)$ if, for every $x \in D_1$ of positive probability, the probability of the event $(\exists w)(wR_1x) \leftrightarrow (\exists w)(wR_2f(x,r))$ is at least $\alpha(|x|)$. Define partially correct reductions of RNP search problems, partially correct decision algorithms and partially correct search algorithms in a similar way. In the rest of this section, reducing (resp. solving) an RNP problem Π with correctness guarantee α means reducing (resp. solving) Π by an AP-time randomizing algorithm with correctness guarantee α.

If an RNP decision problem Π is solvable with a constant correctness guarantee $\alpha > 1/2$ then, for any constant $\beta < 1$, there exists k such that running the given algorithm for Π on a given instance k times and taking the majority answer solves Π with correctness guarantee β. The situation is even better for search problems where one needs only one successful attempt; the inequality $\alpha > 1/2$ can be replaced by $\alpha > 0$. Of course, decreasing correctness guarantees can be boosted as well. In the case of search problems, it makes sense to allow inverse polynomial correctness guarantees. Iterating such an AP-time randomizing search algorithm a sufficient number of times gives an AP-time randomizing search algorithm with correctness guarantee close to 1. If an RNP search problem Π_2 is is solvable with an inverse polynomial correctness guarantee and an RNP search problem Π_1 is reducible to Π_2 with an inverse polynomial correctness guarantee then Π_1 is solvable with an inverse polynomial correctness guarantee.

Define the revised counterpart of P in the new setting to be the class RAP of RNP decision problems solvable with a constant correctness guarantee exceeding $1/2$, say, $2/3$.

Notice, however, that the repetition technique for boosting the probability of correctness, which we applied to randomizing decision and search algorithms above, is not directly applicable to many-one randomizing reductions. Repeating a randomizing reduction k times results in k outputs in the target domain, not one as in the definition of reduction. In other words, such a repetition is a version of Turing (or truth-table) reduction, not a many-one reduction. At this point, we refer the reader to papers [VL, BCGL, IL] where partially correct reductions were successfully employed. It is our impression that the notion of reductions of RNP problems requires a little additional cleaning work.

The prospect of using Turing reductions raises a hope of bridging the gap between decision and search problems, of reducing

search problems to decision problems. This works in the NP setting. For every NP search problem Π, there exists an obvious polynomial time Turing reduction of Π to an NP decision problem Π'. An instance of Π' comprises an instance x of Π and a string u; the corresponding question is whether there exists a witness w for x (with respect to Π) with an initial segment u. This simple reduction does not work for RNP problems; it violates the domination condition. Using substantially more sophisticated randomizing Turing reductions, Ben-David, Chor, Goldreich and Luby were able to prove that every RNP search problem reduces to an appropriate RNP decision problem [BCGL].

6. Revision 2: P-samplable distributions

We return to the definition of domains in Section 2 and discuss probability distributions. For simplicity, restrict attention to domains where the universe is the set $\{0,1\}*$ of all binary strings and the size of a string is its length. In this case, the domain is completely defined by the probability distribution, and one may speak about the uniform distribution and about one distribution dominating another.

A probability distribution on $\{0,1\}*$ is called P-time computable if there exists exists a polynomial time algorithm that, given a string x, computes (a good approximation to) the probability of the collection of strings y such that $|y| < |x|$ or else $|y| = |x|$ and y precedes x in the lexicographical order. For example, the uniform distributions is P-time. The restriction to P-time distributions was used by Levin to construct the first RNP complete problem [Le1]. This restriction turns out to be too strict.

What distributions are likely to come up in applications? It is natural to assume that some randomizing algorithm is used to generate instances of a problem in question. Then there exists a computable function $h(r)$ from sequences of coin flips to instances of our problem such that the probability $\mu(x)$ of an instance x is proportional to the uniform probability of $h^{-1}(x)$. (We say "proportional" rather than "equal" because the generating algorithm is not required to always terminate.)

Remark. Every such distribution μ, never mind the complexity of the generating algorithm, is dominated by so-called *universal distribution* reflecting the information complexity. Li and Vitani notice that, in the case of the universal distribution, the average-case time complexity is "of the same order of magnitude as the corresponding worst-case complexity" [LV]. The idea is that, in particular, the universal distribution dominates the distribution that concentrates exclusively on the worst-case instances. In practice, of course, the generating algorithms satisfy severe resource bounds and the average-case complexity is often much lower than the worst-case complexity.

It is possible that function h is easily invertible and preserves the order of strings. For example, $h(r)$ may be the concatenation of r and some string $h'(r)$. In such a case, distribution μ is P-time. In general, however, one cannot count on μ being P-time. One may want to distinguish between the cases when μ in question is produced by nature or some other disinterested party and the case when the distribution is produced by an adversary. Even a disinterested party may inadvertently mess things up. Certainly one would expect an adversary to mess things up. The following definition is implicitly in [Le2] and explicitly in [BCGL].

Definition 6.1. A distribution is P-*samplable* if it is generated by a coin-flipping Turing machine M (with no addi-

tional input) such that the length of every terminating computation of M is bounded by a fixed polynomial of the output.

Ben-David, Chor, Goldreich and Luby prove that (i) every P-time distribution is P-samplable, and (ii) if there exists a one-way function (a function which is easy to compute but difficult – in some precise technical sense – to invert) then there is a P-samplable distribution which is not dominated by any P-time distribution [BCGL]. Fortunately, Impagliazzo and Levin were able to prove the following theorem.

Theorem 6.1 [IL]. Every NP search problem with a P-samplable distribution reduces to an NP search problem with the uniform distribution.

Redefine the notion of domains by relaxing the requirement that the probability distribution is P-time and requiring only that it is P-samplable. In particular, this gives a new definition of the class RNP which is the counterpart of NP in the average case approach. By Theorem 6.1, problems complete for old RNP (with only P-time distributions allowed) remain complete for the new RNP.

7. Bounded on average

This section is a little more technical. The reader, not interested in details, can safely skip proofs.

Let T be a random variable with values in the interval $[0..\infty]$ of the extended real line. The function $\mathbf{P}[T > t]$ is continuous from the right, i.e., $\mathbf{P}[T > t] = \inf\{\mathbf{P}[T > s] : s > t\}$. Also $\mathbf{P}[T \leq t]$ is continuous from the right. $\mathbf{P}[T < t]$ and $\mathbf{P}[T \geq t]$ are continuous from the left.

Numbers t such that $\mathbf{P}[T \leq t] \geq 1/2$ and $\mathbf{P}[T \geq t] \geq 1/2$ are called medians of T. They form an interval. It is easy to see that the interval of medians is not empty. For example, it contains

$$M(T) = \sup\{t : \mathbf{P}[T \geq t] \geq 1/2\}.$$

If $M(T) = 0$ then $\mathbf{P}[T \geq M(T)] = 1 > 1/2$; otherwise, by the continuity from the left, $\mathbf{P}[T \geq M(T)] = inf_{s<M(T)}\mathbf{P}[T \geq s] \geq 1/2$. If $M(T) = \infty$ then $\mathbf{P}[T \leq M(T) = 1 > 1/2$; otherwise, by the continuity from the right, $\mathbf{P}[T \leq M(T)] = sup_{s>M(T)}\mathbf{P}[T \leq s] \geq 1/2$. The number $M(T)$ may be called the *upper median* for T. For brevity, we will say that $M(T)$ is *the median* for T.

Notice that $M(T) \leq 2\mathbf{E}(T)$. For, T is a function on some sample space X. Define another function

$$T'(x) = [\text{ if } x < M(T) \text{ then } 0, \text{ else } M(T)]$$

on X. Clearly, $M(T)/2 \leq \mathbf{E}(T') \leq \mathbf{E}(T)$.

Definition 7.1. Let $0 \leq \varepsilon \leq 1$. The *$\varepsilon$-quantile* of T is

$$Q(T, \varepsilon) = \sup\{t : \mathbf{P}[T \geq t] \geq \varepsilon\}.$$

Check that $\mathbf{P}[T \geq Q(T, \varepsilon)] \geq \varepsilon$ and $\mathbf{P}[T \leq Q(T, \varepsilon)] \geq 1 - \varepsilon$.

Definition 7.2. $M(T, k) = M(T_1 + \ldots + T_k)$ where $T_1, \ldots, T_k$ are independent random variables with the same distribution as T.

There is a close relation between $M(T, k)$ and $Q(T, 1/k)$.

Lemma 7.1.

- $Q(T, 1/k) \leq M(T, k)$.
- If $a < \log_e 2$ then, for all sufficiently large k, $M(T, k) \leq kQ(T, a/k)$.

Proof. Let $s = Q(T, 1/k)$ and $t = Q(T, a/k)$. Recall that T is a function on some sample space X. Define functions A and B from X to $[0..\infty]$ as follows:

$$\begin{aligned} A(x) &= [\text{ if } T(x) < s \text{ then } 0 \text{ else } s], \\ B(x) &= [\text{ if } T(x) \leq t \text{ then } t \text{ else } \infty] \end{aligned}$$

so that $A(x) \leq T(x) \leq B(x)$ for all x. Let $A_1, \ldots, A_k$ (resp. $B_1, \ldots, B_k$) be independent random variables with the same distribution as A (resp. B).

To establish the first inequality, it suffices to prove that $M(A,k) \geq s$. Let $p = \mathbf{P}[A = s]$ so that $p = \mathbf{P}[T \geq s] \geq 1/k$ and $(1-p) \leq e^{-p} \leq e^{-1/k}$. Clearly $A_1 + \ldots + A_k \geq s$ if and only if some $T_i \geq s$. Thus,

$$\mathbf{P}[A_1 + \ldots + A_k] \geq s = \\ 1 - (1-p)^k \geq 1 - 1/e > 1/2$$

and therefore $M(A,k) \geq s$.

We prove the second inequality. Since $a < log_e 2$, $e^{-a} > 1/2$. Let k be sufficiently large so that $(1 - a/k)^k > 1/2$. It suffices to prove that $M(B,k) < kt + \varepsilon$ for any $\varepsilon > 0$. Let $q = \mathbf{P}[B \leq t]$ so that $q = \mathbf{P}[T \leq t] \geq 1 - a/k$. Clearly, $B_1 + \ldots + B_k < kt + \varepsilon$ if and only every $B_i \leq t$. Hence

$$\mathbf{P}[B_1 + \ldots + B_k < kt + \varepsilon] = \\ q^k \geq (1 - a/k)^k > 1/2$$

so that $\mathbf{P}[M(B,k) \geq kt + \varepsilon] < 1/2$ and therefore $M(B,k) < kt + \varepsilon$. QED

Definition 7.3. A random variable T is *bounded on average* if there exists $\varepsilon > 0$ such that $\mathbf{E}(T^\varepsilon < \infty$.

Theorem 7.1. T is bounded on average if and only if $Q(T, 1/k)$ is bounded by a polynomial of k if and only if $M(T,k)$ is bounded by a polynomial of k.

Proof. By Lemma 7.1, it suffices to prove only the first equivalence. Let $q(k) = Q(T,k)$. Without loss of generality, T is unbounded, $\mathbf{P}[T = \infty] = 0$ and ∞ is the only limit point for the range of T. (If the range of T has other limit points, replace T with $\lceil T \rceil$). Then $q(k) = \sup\{t : \mathbf{P}[T \geq 1/k]\} = \max\{t : \mathbf{P}[T \geq 1/k]\}$ and therefore $\mathbf{P}[T = q(k)] > 0$.

Further, without loss of generality, we may assume that the sequence $q(k)$ is strictly monotone. If $q(k) = q(k+1) = \ldots = q(k+j-1) < q(k+j)$ and either $k = 1$ or $q(k-1) < q(k)$, choose points $q'(k+1) < \ldots < q'(k+j-1)$ between $q(k)$ and $\min(q(k)+1, q(k+j))$ and modify T such that $Q(T,i) = q'(i)$ for $k < i < k+j$.

Every sufficiently large value of T belongs to some interval $(q(k-1), q(k)]$ and therefore T is bounded on average if there exists $\varepsilon > 0$ such that $\sum_k q(k)^\varepsilon \cdot \mathbf{P}[q(k-1) < T \geq q(k)] < \infty$. We have

$$\mathbf{P}[q(k-1) < T \geq q(k)] \leq \\ \mathbf{P}[T > q(k-1)] - \mathbf{P}[T \geq q(k+1)] \leq \\ \frac{1}{k-1} - \frac{1}{k+1} = O(k^{-2}).$$

It is clear that if $q(k)$ is bounded by a polynomial of k then T is bounded on average.

On the other hand, every value of T belongs to some interval $[q(k), q(k+1))$. We have

$$\mathbf{P}[q(k) \leq T < q(k+1)] \geq \\ \mathbf{P}[T \geq q(k)] - \mathbf{P}[Tleqq(k+2)] \geq \\ \frac{1}{k} - \frac{1}{k+2} = \Omega(k^{-2}).$$

If $\mathbf{E}(T^{1/j}) < \infty$ for some j then $\sum_k q(k)^{1/j} \cdot k^{-2} < c$ for some c, and therefore $q(k) \leq c^j k^{2j}$. QED

Now fix a domain. The notion of boundedness on average allows us to give the following useful characterization of AP functions.

Lemma 7.2. A function T from some domain D to $[0..\infty]$ is AP if and only if there exists a polynomial p and a bounded on average function B from D to $[0..\infty]$ such that $T(x) = p(x) \cdot B(x)$.

Proof. If $\mathbf{E}(T(x)^{1/k}/|x|) < 0$, the desired $B(x) = T(x)/|x|^k$. Suppose that $T(x)\|x|^i \cdot B(x)$, $\mathbf{E}(B(x)^{1/j}) < \infty$ and $k = \max(i,j)$. Then $\mathbf{E}((Tx)^{1/k}/|x|) \leq \mathbf{E}(B^{1/j} < \infty$. QED

It is easy to check that the class of bounded on average functions is closed

under, say, pointwise maxima ($h(x) = \max(f(x), g(x))$ and products. Hence the class of AP functions is closed under pointwise maxima and products.

8. One alternative to P=?NP

Is theoretical computer science a part of mathematics? A good case can be made in favor of the thesis that theoretical computer science is a part of applied mathematics. The word "applied" is essential here. Even a remote connection to possible applications gives important guidance. A case in point is the famous question P=?NP, so very popular in theoretical computer science. The importance of the P=?NP question is related to the thesis identifying feasible and polynomial time computations, thesis (1.0) of Section 1. (There is a beautiful article of Trakhtenbrot on the original motivation for the P=?NP question [Tr].) In this connection, a computer scientist, who isn't convinced that P captures feasibility, may question the centrality of the P=?NP question. A purer mathematician may be unmoved. The question was asked. The challenge was posed. Now is the time to solve the question rather than to try to get around it.

In [Gu2], the P=?NP question was criticized for a bias toward the positive solution and an alternative question RAP=?RNP, the counterpart of the P=?NP question in the average-case approach, was advertized. In that connection, the following game between Challenger and Solver was considered. Challenger chooses an NP problem Π and then repeatedly picks instances of Π, and Solver tries to solve them. If P=NP then Solver can win the game: By the more convincing part (1.1) of the thesis (1.0), polynomial time algorithms are feasible. On the other hand, it is not clear that Challenger can make Solver work much harder than he does if P$\neq$NP. It may be very hard to find hard instance of Π.

Now assume that Challenger employs a randomizing algorithm generating instances of Π in time polynomial in the

output. (Polynomiality in the output is a feasibility requirement in this context. There is a limit on Challenger's time.) This makes Π an RNP problem. We consider (randomizing) decision or search algorithms feasible if they run in time polynomial on average. Thus, Solver can win if RAP=RNP. What happens if RAP≠RNP? Can Challenger win the game? The answer seems to be yes. The following argument involves medians.

Suppose that Π is any RNP problem that is not RAP, and $T_S(x, r)$ is the running time of Solver's algorithm on instance x and a sequence r of coin flips. By Theorem 7.1, the median $M(T_S, k)$ of the time that Solver needs to solve k instances is not bounded by a polynomial of k. Furthermore, Challenger's generating process may be altered in such a way that the expectation $\mathbf{E}(T_C)$ of Challenger's time for generating one instance is bounded D [Gu2]. Then the expected time for generating independently k instances is $\Theta(k)$ and therefore (recall the remark that $M(T) \leq 2\mathbf{E}(T)$ in Section 7) the median $M(T_C, k)$ of Challenger's time needed to produce k instances is $\Theta(k)$. Thus, the median $M(T_S, k)$ of Solver's time is not bounded by any polynomial of the expectation $k \cdot \mathbf{E}(T_C)$ or the median $M(T_C, k)$ of Challenger's time.

It is easy to see that RAP≠RNP if there exists a one-way function. It is not known whether the converse is true. The converse fails under some oracle [IR], but the question is open and very exciting. The existence of one-way functions allows cryptography in a meaningful sense [ILL, Ha]. If the existence of one-way functions follows from RAP≠RNP then the question RAP=?RNP is beautifully balanced. Either all RNP problems are easy on average or else there are problems that are hard on average but then they can be used to do cryptography.

Acknowledgement. We are happy to thank Andreas Blass and Leonid Levin for most useful discussions, Ramarathnam Venkatesan and Moshe Vardi for comments on a draft of this paper, and our student Jordan Stojanovski who was asked to prove Theorem 7.1 and did a good job.

References

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

[BG] Andreas Blass and Yuri Gurevich, "On the Reduction Theory for Average-Case Complexity", in Proc. of CSL'90, 4th Workshop on Computer Science Logic (Eds. E. Börger, H. Kleine Büning and M. Richter), Springer LNCS, 1991.

[GJ] Michael R. Garey and David S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", Freeman, New York, 1979.

[Go] Oded Goldreich, "Towards a Theory of Average Case Complexity: A survey", TR-531, Computer Science Dept., Technion, Haifa, Israel, March 1988.

[Gr] Per Grape, "Complete Problems with L-Samplable Distributions", 2nd Scandinavian Workshop on Algorithm Theory, Springer Lecture Notes in Computer Science 447, 1990, 360–367.

[Gu1] Yuri Gurevich, "Average Case Complexity", J. Computer and System Sciences (a special issue on FOCS'87), to appear.

[Gu2] Y. Gurevich, "The Challenger-Solver game: Variations on the Theme of P=?NP", Bulletin of European Assoc. for Theor. Computer Science, October 1989, 112–121.

[Gu3] Yuri Gurevich, "Matrix Decomposition Problem is Complete for the Average Case", Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1990, 802–811. A full version of this paper, coauthored by Blass and Gurevich, is being prepared for publication.

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

[Ha] Johan Hastad, "Pseudo-Random Generators under Uniform Functions", Symposium on Theory of Computing, ACM, 1990, 395–404.

[IR] Russel Impagliazzo and Stephen Rudich, private communication.

[IL] Russel Impagliazzo and Leonid A. Levin, "No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random", Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1990, 812–821.

[ILL] Russel Impagliazzo, Leonid A. Levin and Michael Luby, "Pseudo-Random Generation from One-Way Functions", 21st Symposium on Theory of Computing, ACM, New York, 1989, 12–24.

[Jo] David S. Johnson, "The NP--Completeness Column", Journal of Algorithms 5 (1984), 284-299.

[Kn] P.M.W. Knijnenburg, "On Randomizing Decision Problems: A Survey of the Theory of Randomized NP", Tech. Report RUU-CS-88-15, Rijksuniversitait Utrecht, The Netherlands, March 1988.

[Le1] Leonid A. Levin, "Average Case Complete Problems", STOC 1984, the final version in SIAM Journal of Computing, 1986.

[Le2] Leonid A. Levin, "One-Way Functions and Pseudo-Random Generators", Symposium on Theory of Computing, ACM, 1985, 363–375.

[LV] Ming Li and Paul M. B. Vitani, "Average Case Complexity under the Universal Distribution Equals Worst Case Complexity", Manuscript, 1989.

[Sc] Robert E. Schapire, "The Emerging Theory of Average Case Complexity", Tech. Report MIT/LCS/TM-431, June 1990.

[Tr] Boris A. Trakhtenbrot, "A Survey of Russian Approaches to Perebor (Brute-Force Search) Algorithms", Annals of the History of Computing, 6:4 (1984), 384–400.

[VL] Ramarathnam Venkatesan and Leonid Levin, " Random Instances of a Graph Coloring Problem are Hard", Symposium on Theory of Computing, ACM, 1988, 217–222.

[Ve] Ramarathnam Venkatesan, private correspondence.

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