A Personal View of Average-Case Complexity

Russell Impagliazzo*
Computer Science and Engineering
UC, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114
russell@cs.ucsd.edu

April 17, 1995

Abstract

The structural theory of average-case complexity, introduced by Levin , gives a formal setting for discussing the types of inputs for which a problem is difficult. This is vital to understanding both when a seemingly difficult (e.g. NP-complete) problem is actually easy on almost all instances, and to determining which problems might be suitable for applications *requiring* hard problems, such as cryptography. This paper attempts to summarize the state of knowledge in this area, including some "folklore" results that have not explicitly appeared in print. We also try to standardize and unify definitions. Finally, we indicate what we feel are interesting research directions. We hope that this paper will motivate more research in this area and provide an introduction to the area for people new to it.

*Research Supported by NSF YI Award CCR-92-570979, Sloan Research Fellowship BR-3311, and USA-Israel BSF Grant 92-00043

1 Introduction

There is a large gap between a problem not being easy and the same problem being difficult. A problem could have no efficient worst-case algorithm but still be solvable for "most" instances, or on instances that arise in practice. Thus, a conventional completeness result can be relatively meaningless in terms of the "real life" difficulty of the problem, since two problems can both be NP- complete, but one can be solvable quickly on most instances that arise in practice and the other not. However, "average run-time" arguments of particular algorithms for particular distributions are also unenlightening as to the complexity of real instances of a problem. First, they only analyze the performance of specific algorithms rather than describing the inherent complexity of the problem. Secondly, the distributions of inputs that arise in practice are often difficult to characterize, so analysis of algorithms on "nice" distributions does not capture the "real-life" average difficulty.

Thus, a *structural theory of distributional complexity* is necessary. Such a theory should allow one to compare the inherent intractability of distributional problems (computational problems together with distributions on instances). It should also provide results that are meaningful with respect to instances from an arbitrary distribution that might arise.

Besides capturing more accurately the "real world" difficulty of problems, the "average-case complexity" of a problem is important in determining its suitability for applications such as cryptography and the de-randomization of algorithms. For such applications, one needs more than the mere existence of hard instances of the problem; one needs to be able to generate instances in a way that guarantees that almost all generated instances are hard.

For these reasons, Levin in [L1] introduced a structural theory of the average-case complexity of problems. The main contributions of his paper were a general notion of a distributional problem, a machine-independent definition of the average-case performance of an algorithm, an appropriate notion of reduction between distributional problems, and an example of a problem that was complete for the class of all NP problems on sufficiently "uniform" distributions. Since, he and many others have built on this foundation (see e.g., [BCGL],[G2],[VL], [G3]).

Despite the above work, I feel the structure of average-case complexity has not received the attention due to a central problem in complexity theory. The goal of this paper is to motivate more research in this area, and to make the research frontier more accessible to people starting work in this area.

Several caveats are necessary with respect to this goal. As this is basically a propaganda piece, I will present my own personal view of what makes the field exciting. I will not present a comprehensive summary or bibliography of work in the area, nor do I claim that the work mentioned here is the "best" in the area. I will also attempt to "clarify" and "simplify" concepts in the area by presenting both my own equivalent formulations and also by trying to make a uniform taxonomy for concepts. The current definitions are the product of much thought and work by top researchers, so many researchers in the area will consider my attempts to do this as a "confusion" and "complicating" of the issues rather than a "clarification and simplification" of them. However, I feel someone starting out in the area might benefit from seeing a variety of perspectives. Many of the results mentioned in this paper should be considered "folklore" in that they merely formally state ideas that are well-known to researchers in the area, but may not be obvious to beginners and to the best of my knowledge do not appear elsewhere in print.

2 Five possible worlds

To illustrate the central role in complexity theory of questions regarding the average case complexity of problems in NP, we will now take a guided tour of five possible (i.e., not currently known to be false) outcomes for these questions, and see how they would affect computer science. In each such "world", we will look at the influence of the outcomes of these questions on algorithm design for such areas as artificial intelligence and VLSI design, and for cryptography and computer security. We will also consider the more technical issue of derandomization of algorithms (the simulation of probabilistic algorithms by deterministic algorithms). This will have a much smaller impact on society than the other issues, but we include it as another situation (besides cryptography) where having difficult problems is actually useful.

Finally, to provide a human angle, we will consider the impact these questions would have had on the sad story of Professor Grouse, the teacher who assigned the young Gauss's class the problem of summing the numbers from 1 to 100. The beginning of this story is well-known, but few people realize that Professor Grouse then became obsessed with getting his revenge by humiliating Gauss in front of the class, by inventing problems Gauss could not solve. In real life, this led to Grouse's commitment to a lunatic asylum (not a pleasant end, especially in the 19'th century) and to Gauss's developing a life-long interest in number-theoretic algorithms. Here, we imagine how the story might have turned out had Grouse been an expert in computational complexity at a time when the main questions about average-case complexity had been resolved. (We believe that this story inspired Gurevich's "Challenger-Solver Game" [G1]).

In this section, we will leave unresolved the questions of how to properly formalize the complexity assumptions behind the worlds. In particular, we will leave open which model of computation we are talking about, e.g., deterministic algorithms, probabilistic algorithms, Boolean circuits, or even quantum computers, and we shall ignore quantitative issues, such as whether an n^{100} time algorithm for satisfiability would be "feasible". We also assume that, if an algorithm exists, then it is known to the inhabitants of the world. We also ignore the issue of whether it might be possible that algorithms are fast for some input sizes but not others, which would have the effect of bouncing us from world to world as technology advanced.

We will take as our standard for whether these worlds are indeed "possible" the existence of an oracle relative to which the appropriate assumptions hold. Of course, this is far from a definitive answer, and the existence of an oracle should not stop the researcher from attempting to find non-relativizing techniques to narrow the range of possibilities. Indeed, it would be wonderful to eliminate one or more of these worlds from consideration, preferably the pestilent Pessiland. We will try to succinctly and informally describe what type of algorithm and/or lower bound would be needed to conclude that we are in a particular world. Barring the caveats mentioned in the previous paragraph, these conditions will basically cover all eventualities, thus showing that these are the only possible worlds. (This is an informal statement, and will be more true for some worlds than others.)

2.1 Algorithmica

Algorithmica is the world in which $P = NP$ or some moral equivalent, e.g., $NP \subseteq BPP$. In this world, Grouse would have even less success at stumping Gauss than he had in real life. Since Grouse needed to stump Gauss on a problem for which he (Grouse) could later present an answer to the class, he is restricted to problems which have succinct, easily verifiable solutions, i.e., NP. Gauss could use the method of verifying the solution to automatically solve the problem.

Such a method of automatically producing a solution for a problem from the method of recognizing a valid solution would revolutionize computer science. Seemingly intractable algorithmic problems would become trivial. Almost any type of optimization problem would be easy and automatic; for example, VLSI design would no longer use heuristics, but could instead produce exactly optimal layouts for problems once a criterion for optimality was given. Programming languages would not need to involve instructions on *how* the computation should be performed, Instead, one would just specify the properties that a desired output should have in relation to the input. If the specification language is such that it is easy to evaluate whether an output meets the specification, then the compiler could automatically feed it to the algorithm to solve the NP-complete problem to generate the output. (This is the motivation behind logic-programming languages such as PROLOG, but in Algorithmica it would actually work that way!)

Less obviously, $P = NP$ would make trivial many aspects of the artificial intelligence program that are in real life challenging to the point of despair. Inductive learning systems would replace our feeble attempts at expert systems. One could use an "Occam's Razor" based inductive learning algorithm to automatically train a computer to perform any task that humans can (see, e.g., []). Such an algorithm would take as input a training set of possible inputs and outputs produced by a human expert, and would produce the simplest algorithm that produced the same results as the expert. Thus, a computer could be taught to recognize and parse grammatically correct English just by having sufficiently many examples of correct and incorrect English statements, without needing any specialized knowledge of grammar or English. (This assumes merely that there *exists* a simple algorithm that humans use to parse natural languages. People have attempted to use neural nets to do similar learning tasks, but that implicitly makes the much stronger assumption that the task is performable by a constant depth threshold circuit, which is not always reasonable.)

Using the result that approximate counting is in the polynomial-time hierarchy [St], exponential sized spaces of possible sequences of events could be searched and a probability estimate for an event given observed facts could be output, thus producing Mr. Spock-like estimates for all sorts of complicated events. "Computer-assisted mathematics" would be a redundant phrase, since computers could find proofs for any theorem in time roughly the length of the proof. (We could use the above learning method to train the computer to search for "informal proofs acceptable to mathematicians" or "papers acceptable at FOCS"!) In short, as soon as a feasible algorithm for an NP-complete problem is found, the capacity of computers will become that currently depicted in science fiction.

On the other hand, in Algorithmica, there would be no way of telling different people or computers apart by informational means. The above-mentioned learning algorithms could simply learn to mimic the behavior of another machine or person. Any code that could be developed could be broken just as easily. It would do little good to keep the algorithm the code is based on secret, since an identical algorithm could be automatically generated from a small number of samples of encrypted and clear-text messages. There would be no way to allow some people access to information without making it available to everyone. Thus any means of identification would have to based on some physical measurement, and the security of the identification would have to be based on the unforgeability of the physical measurement and the extent to which all channels from the measuring device to the identifier are tamper-proof. In particular, any file or information remotely accessible via a possibly insecure channel would basically be publicly available. (The above assumes that no physical property is directly observable at a distance, which may not be true. In particular, it may be possible to identify people based on certain quantum effects [BBR]).

There seems to be no reason why randomness could not be essential for the worst-case algorithm for the NP-complete problem. No general techniques for derandomization are known to be possible in a version of Algorithmica where, say, $NP = RP \neq P$.

To show that we are in Algorithmica, one needs to present an efficient algorithm for some NP-complete language. A relativized Algorithmica was given in [BGS].

2.2 Heuristica

Heuristica is the world where NP problems are intractable in the worst-case, but tractable on average for any samplable distribution.

Heuristica is in some sense a paradoxical world. Here, there *exist* hard instances of NP problems, but to *find* such hard instances is itself an intractable problem! In this world, Grouse might be able to find problems that Gauss cannot answer in class, but it might take Grouse a week to find a problem that Gauss could not solve in a day, and a year to find one that Gauss could not solve in a month. (Here, I am assuming that Gauss has some polynomial advantage over Grouse, since Gauss is after all a genius!) Presumably, "real-life " is not so adversarial that it would solve intractable problems just to give us a hard time , so for all practical purposes this world is indistinguishable from Algorithmica.

Or is it? In Heuristica, the time to solve a problem drawn from a distribution might be polynomial in not just the problem size but also *the time required to sample from the distribution* and the *fraction of problems from the distribution that are at least as "hard" as the given problem.* In other words, the average-case time to solve an NP problem is a function of the average-case time to think up the problem. This makes the situation not at all clear. Say that, on average, it takes us just twice as long to solve a problem as it does to think it up. As we all know, the solution to one mathematical problem invariably leads to another problem. So if we spend time T thinking up problem 1, and then $2T$ solving it, and the solution leads to a second problem 2, we have spent $3T$ time thinking up problem 2. Thus, it might take $6T$ time to solve problem 2 in Heuristica. (In Algorithmica, the time would be independent of how we thought up the problem.) Which leads to a problem 3 which took $10T$ steps to think up, and so $20T$ time to solve. Since this recursion is exponential, in a few iterations we have crossed the border between "feasible" and "infeasible".

A more specific example of a possible difference between Algorithmica and Heuristica would be $VLSI$ problems involving circuit minimization. In $VLSI$, algorithms should be given some representation of a function and then be able to design a circuit that is minimal with respect to certain costs that computes the function. In Algorithmica, you could make up such an algorithm in two stages. First, you could use your solution to an NP-complete problem to come up with an algorithm that will recognize when a circuit actually computes the specified function, this being a $Co-NP$ problem, since you could certify the circuit incorrect by providing one input on which it does not produce the specified value. Then, using the first algorithm as the defining criterion for what a possible solution is, the problem of minimization becomes an NP-type problem, and you can solve it using your algorithm for an NP-complete problem.

The same process in Heuristica is not guaranteed to produce good results. Your first algorithm will work well on *most* circuits and specifications, but you don't really care about most circuits. You really want an algorithm that will work well on circuits that are minimal instantiations of specifications! Such circuits might not be distributed in any nice way, and since it would seem to take exponential time to find such circuits, there is no reason why they might not be the hard to find, hard instances of the problem on which algorithms fail in Heuristica.

Thus, a central problem in the structure of average-case complexity is : if all problems in NP are easy on average, can the same be said of all problems in the polynomial hierarchy? (The circuit minimization problem is in Σ_2^P and problems involving repeated iterations of NP questions are in P^{NP}.) This question is explored in more detail in [SW]. The best known result along these lines is that of [BCGL] reducing average case search problems to average case decision problems.

As far as network security and cryptography go, there would not be much of a difference between Algorithmica and Heuristica. It would not be much help to have legitimate users spend huge amounts of time thinking up problems to uniquely identify them if eavesdroppers can solve the problems in comparable amounts of time. One should always assume that people willing to break a system are also willing to use significantly more resources doing so than legitimate users are willing to spend routinely!

As we shall see later, there are several ways of formalizing a problem's being "easy-on-average". In some of these definitions, some de-randomization follows; for example, one can show that if all NP problems have polynomial-on-average probabilistic algorithms in the sense of Levin, then $BPP = ZPP$. However, we feel this is more of an artifact of the definition than an essential fact about Heuristica. We will present alternate definitions in the next section.

From the results of [ILe], being in Heuristica is basically equivalent to knowing a method of quickly solving almost all instances of one of the average-case complete problems on the uniform distribution (see e.g., [L1],[G2],[VL], [G3]). and having a lower bound for the worst-case complexity of some NP-complete problem. We do not know of any relativized Heuristica using Levin's definition of average-case complexity. However, there is an oracle in which every problem in NP has an algorithm that solves it on most instances, yet $NP \not\subseteq P/poly$ ([IR2]). The difference between the two definitions is that in the weaker one, the algorithm always runs in polynomial time but occasionally gives an incorrect answer, whereas Levin's stronger definition insists that the algorithm be always correct, but it may occasionally run for more than polynomial time. (This difference will be detailed in the next section.) We do not know whether these two criteria for NP being easy on average are equivalent, and we feel it is a question worth exploring.

2.3 Pessiland

Pessiland is, to my mind, the worst of all possible worlds, the world in which there are hard average-case problems, but no one-way functions. By the non-existence of one-way functions,we mean that any process $f(x)$ that is easy to compute is also easy to invert in the sense that, for almost all values of x, given $f(x)$, it is possible to find some x' with $f(x') = f(x)$ in roughly the same amount of time it took to compute $f(x)$. In Pessiland, it is easy to generate many hard instances of NP-problems. However, there is no way of generating hard *solved* instances of problems. For any such process of generating problems, consider the function which takes the random bits used by the generator as input and outputs the problem. If this function were invertible, then given the problem, one could find the random bits used to generate the problem, and hence the solution.

In Pessiland, Grouse could pose Gauss problems that even the budding genius could not solve. However, Grouse could not solve the problems either, and so Gauss's humiliation would be far from complete.

In Pessiland, problems for many domains will have no easy solutions. Progress will be like it is in our world: made slowly through a more complete understanding of the real-world situation and compromises by using unsatisfactory heuristics. Generic methods of problem solving will fail in most domains. However, a few relatively amazing generic algorithms are possible based only on the non-existence of one-way functions. For example, [ILe] gives a method of using a generic function inverter to learn in average polynomial time the behaviour of an unknown algorithm by observing its input-output behaviour on some samplable input distribution. It would also be possible to give a generic data compression method, where if one knows the process by which strings are being produced, i.e. an algorithm that produces samples according to the distribution, then, in the limit, strings can be compressed to an expected length of the entropy of the distribution ([IZ]).

Finding other algorithmic implications of the non-existence of one-way functions is an interesting research direction. More generally, the structural theory of cryptography under the axiom that one-way functions exist is rich; is there a similarly rich theory under the axiom that there are no one-way functions?

There does not seem to be a way of making use of the hard problems in Pessiland in cryptography. A problem that no one knows the answer to cannot be used to distinguish legitimate users from eavesdroppers. This intuition is made formal in [ILu], where it is shown that one-way functions are necessary for many cryptographic applications.

The existence of hard average-case problems in a non-uniform setting has been shown by Nisan and Wigderson ([NW])to be sufficient for generic de-randomization. Note that the definition of difficult problem they use is much stronger than the negation of Levin's definition of an easy-on-average problem. They give a smooth trade-off between the difficulty of a problem and its consequences for the de-randomization of algorithms; if a problem in E has exponential difficulty, then $P = BPP$; if such a problem has super-polynomial difficulty, then $BPP \subseteq DTIME(2^{n^{o(1)}})$.

Levin ([L2]) gives an example of a function that is complete for being one-way, so having an algorithm for inverting this function suffices to show that there are no one-way functions. To then show that you are in Pessiland, you need to give an average-case lower bound for some problem in NP.

2.4 Minicrypt

In Minicrypt, one-way functions exist, but public key cryptography is impossible. We here identify public key cryptography with the task of agreeing on a secret with a stranger via a publicly accessible channel, although strictly speaking, public key cryptography is just one method of accomplishing this task. The one-way function could be used to generate hard, solved problems: the generator would pick x, compute $y = f(x)$ and pose the search problem, "Find any x' with $f(x') = y$" knowing one solution, x. Thus, in Minicrypt, Grouse finally gains the upper hand, and can best Gauss in front of the class.

There are no known positive algorithmic aspects to Minicrypt, except that you can use the one-way function to get a pseudo-random generator that can be used to de-randomize algorithms [HILL].

On the other hand, it is possible for participants in a network to identify themselves to other participants and to authenticate messages as originating from them using electronic signatures [NY], [?]. It is possible to prove facts about a secret in in a way that discloses no other information about the secret ([?],[GMW]). It is possible, if a small amount of information is agreed upon in advance, to set up a private unbreakable code between two participants in the network that will allow them to talk privately over a publicly accessible channel. ([HILL],[GGM], [LR]). However, it is impossible to have secure elections over a public channel, or to establish a private code without sending some information through a secure channel. It is not known how to have anonymous digital money in such a world. Many other applications involving multiple participant protocols seem impossible if you cannot establish private codes on public channels.

To prove that the real world is Minicrypt, one would have to prove that no efficient algorithm exists for inverting some one-way functions, and also show how to break any secret-key agreement protocol. There seems to be no nice characterization of secret-key agreement protocols, and maybe this is inherent to the problem ([Ru]), so it is not clear how one could even start to do the latter. [IR] gives a relativized Minicrypt.

2.5 Cryptomania

In Cryptomania, public-key cryptography is possible, i,e., it is possible for two parties to agree on a secret message using only publicly accessible channels. In Cryptomania, Gauss is utterly humiliated; by means of conversations in class, Grouse and his pet student would be able to jointly choose a problem that they would both know the answer to, but which Gauss could not solve. In fact, in such a world, Grouse could arrange that all the students except Gauss would be able to solve the problems asked in class!

Such a secret key agreement protocol implies the existence of a one-way function [ILu], so we still have pseudo-randomness, signatures, identification, zero-knowledge, etc. Also, if one does the secret-key exchange using trap-door one-way functions (and all known protocols are either explicitly or implicitly using such functions), one can do almost any cryptographic task imaginable! (See [?],[?]). Any group of people can agree to jointly compute an arbitrary function of secret inputs without compromising their secrets. This directly includes, for example, secure electronic voting, or anonymous digital cash, although not necessarily in a practical form. Unlike in the other worlds where establishing privacy is a technological challenge, the technology of Cryptomania would limit the capability of authorities to restrict privacy. Most decisions about how much privacy is available to citizens of such a world would be guided by social and political processes rather than technical capability. For example, there are a whole gamut of possible electronic money systems , some of which protect user anonymity to a greater extent than others. Which becomes the standard is a matter of political choice – although perhaps not a democratic choice, since the standards are now set without much public discussion except within a small circle of interested parties.

This world is the one closest to the real world, in that as far as we know, the RSA cryptosystem is secure. Public key cryptography is currently in the transition process of being accepted as a standard, although both technical and political issues block full implementation of the abovementioned protocols.

However, blind acceptance of the existence of public key cryptosystems as a de facto complexity axiom is unwarranted. Currently, all known secure public key cryptosystems are based on variants of RSA, Rabin, and Diffie-Hellman cryptosystems. If an efficient way of factoring integers and solving discrete logarithms became known, then not only would the popular public key cryptosystems be broken, but there would be no candidate for a secure public-key cryptosystem, or any real methodology for coming up with such a candidate. There is no theoretical reason why factoring or discrete log should be intractable problems. Confidence that they are intractable is based on our ignorance of any good method for solving the problems after more than twenty years of intense research. However, the same twenty years have vastly improved number-theoretic algorithms, so there is no reason to suspect similar improvements do not lie ahead. This makes it impossible to pick parameters for public-key sizes that will be still secure in say 20 years. In fact, the earliest guess for such a parameter 20 years ago was recently broken. More speculatively, it has been recently shown how to solve both problems in the quantum computer model [Sh]. The existence of public-key cryptography is fragile at best.

To prove that we live in Cryptomania, one must prove that a particular secret-key exchange protocol is secure. Proving a strong lower bound on the average case time to factor or take discrete logs would be sufficient, and no other problems are currently candidates for founding public-key cryptography. Brassard[Bra] gives a relativized world where public-key cryptography is possible.

3 Definitional issues

The definitions Levin gave for the basic concepts of his theory seem counter-intuitive to many people on first reading. For example, he talks about the expectation of some positive power of the time taken by an algorithm, rather than that of the time. In this section, we will give some equivalent formulations of Levin's definitions that are intended to justify the definitions and make them seem more intuitive. We will also present some variations of these definitions that seem related but not equivalent.

3.1 Infinite input distributions versus ensembles of finite input distributions

One feature of Levin's definition that I personally find unappealing is that in his definition of a distributional problem, the input distribution is a single distribution on all inputs of all sizes. I prefer to think of the input distribution as being, at any fixed time, on a finite set of possible inputs of at most some fixed size. However, as technology improves, the size of inputs that we are interested in increases (since most computational problems arise from the technology itself). So the inputs for an average-case problem are to my mind best modeled by a *sequence* of finite probability distributions on strings of bounded size, where the sequence is parameterized by the input size. Fortunately, as we shall see, Levin's definition of average-case complexity remains pretty much unchanged under either model. So the choice of finite versus infinite input distributions is merely an aesthetic one.

The proof here is messy, but stupid. It

is included for completeness, but please feel free to accept the moral without getting bogged down in the computation. I include Levin's definition of a time function's being "polynomial-on-average" here without explanation or justification, so that we can eliminate the infinite distributions once and for all. If you don't want to try to make sense of this definition, skip to the next subsection, where an equivalent formulation is given.

(Intuitively, in the following, $T(i)$ represents the time taken by a machine on input i.)

DEFINITION 3.1: A *distribution* on the positive integers Z^+ is a function $\mu : Z^+ \rightarrow R$ where $\mu(i) \geq 0$ and $\sum_{i\in Z^+} \mu(i) = 1$, A distribution on a finite set S is the same replacing Z^+ with S in the sum. An *ensemble of distributions* is a sequence of distributions μ_n, $n \in Z^+$, where each μ_n is a distribution on the set of positive integers with binary length at most n. A function $T : Z^+ \rightarrow Z^+$ is *polynomial on average* with respect to μ, a distribution on Z^+, if there is some $\epsilon > 0$ so that $\sum_{i\in Z^+} T(i)^\epsilon |i|^{-1}\mu(i)$ converges. We say that T is *polynomial on average* with respect to an ensemble of distributions $\mu_n, n \in Z^+$ if there is an $\epsilon > 0$ so that the expectation of $T(i)^\epsilon$ when i is chosen according to μ_n is $O(n)$,

Proposition 1: Let μ be a distribution on Z^+ and let μ_n be the restriction of μ to numbers of length at most n. Then any function T is polynomial on average with respect to μ if and only if it is polynomial on average with respect to the ensemble μ_n, $n \in Z^+$.

Proof: Assume T is polynomial on average with respect to μ. So $\sum_i T(i)^\epsilon |i|^{-1}\mu(i)$ converges for some $\epsilon > 0$. Then $\sum_{i,|i|\leq n} T(i)^\epsilon \mu_n(i) \leq \sum_{i,|i|\leq n}(n/|i|)T(i)^\epsilon(\mu(i)/Prob_{i\in_\mu Z^+}[|i| \leq n]) = O(n)\sum_i T(i)^\epsilon |i|^{-1}\mu(i) = O(n)$, so T is polynomial on average with respect to μ_n.

Conversely, if T is polynomial on average with respect to μ_n, there is some $\epsilon > 0$ so that $T(i)^\epsilon$ has expectation $O(n)$ when i is chosen according to n. Then $\sum_{i,|i|=n} T(i)^\epsilon \mu(i) \leq \sum_{i,|i|\leq n} T(i)^\epsilon \mu(i) \leq \sum_{i,|i|\leq n} T(i)^\epsilon \mu_n(i) = O(n)$. Thus $\sum_i (T(i)^{\epsilon/3})|i|^{-1}\mu(i) = \sum_{i,T(i)^{\epsilon/3}\leq|i|} T(i)^{\epsilon/3}|i|^{-1}\mu(i) + \sum_{i,T(i)^{\epsilon/3}>|i|} T(i)^{\epsilon/3}|i|^{-1}\mu(i) \leq \sum_i \mu(i) + \sum_{i,T(i)^{\epsilon/3}\geq|i|}(T(i)^\epsilon/(|i|T(i)^{2/3\epsilon}))\mu(i) \leq 1 + \sum_n \sum_{i,|i|=n}(T(i)^\epsilon\mu(i))/n^3 = 1+\sum_n O(n)/n^3 = 1+\sum_n O(1/n^2)$, which converges. So T is polynomial on average with respect to μ. □.

From now on then, we will look at the input as coming from one element of an ensemble of distributions.

3.2 Expected Time versus the "Average Case"

Why did Levin look at the expectation of T^ϵ rather than T? The traditional answer is that the expectation of a function might be small, but some polynomial of that function, huge, For example, if $T(x) = n$ for all but a $1/2^n$ fraction of inputs, but was 2^n on those inputs, then the expectation of T is $O(n)$, but the expectation of T^2 is $O(2^n)$. Thus, if you first do a computation that's *expected* polynomial time, and then compute a worst-case polynomial-time function of the result, the whole process might not be expected polynomial time. Levin's definition closes the class of average-case polynomial problems under such transformations.

However, I think there's a better reason. Levin's definition is not intended to capture the expected cost to the solver; rather, it captures the trade-off between a measure of difficulty and the *fraction* of hard instances of the problem, i.e., between a time bound T and the fraction of instances that take the algorithm more than T time. This trade-off should be polynomial in T: only a sub-polynomial fraction of instances should require super-polynomial time, only a quasi-polynomial fraction more than quasi-polynomial time, etc. Thus, the time to find, through random sampling, an instance requiring more than T time is at least T^{ϵ}, so the poser does not have more than a polynomial advantage over the solver. Levin hints at this in the last sentence of his original paper, and Gurevich has explained it nicely in [G1]. However, I feel that the following formal statement based on this intuition might be helpful to have in the literature:

DEFINITION 3.2: A distributional problem is a function f and an input ensemle μ_n, $n \in Z^+$. The distributional problem f on input ensemble μ_n is said to be in $AvgP$ if there is an algorithm to compute f whose running time is polynomial on average with respect to μ_n. An algorithm computes f with *benign faults* if it either outputs an element of the range of f or " ?" and if it outputs anything other than ?, it is correct (f of the input.) A polynomial-time benign algorithm scheme for a function f on μ_n is an algorithm $A(x, \delta)$ so that:

- A runs in time polynomial in $|x|$ and $1/\delta$.
- A computes $f(x)$ with benign faults.
- $\forall \delta, 1 > \delta > 0$ and all $n \in Z^+$, $Prob_{x \in_{\mu_n} Z^+}[A(x, \delta) = ?] \leq \delta$.

Proposition 2: A problem f on input ensemble μ_n is in AvgP if and only if it has a polynomial-time benign algorithm scheme.

Proof: Assume f on μ_n is in AvgP. Then there is an algorithm A so that for $T_A(x)$ the time T takes on input x, $Exp_{x \in_{\mu_n} Z^+}[T_A(x)^{\epsilon}] = O(n)$. Then $Prob[T_A(x) \geq O((kn)^{1/\epsilon})] \leq 1/k$. So the algorithm B where $B(x, \delta)$ simulates A for $O(n/\delta)^{1/\epsilon}$ steps, and outputs ? if A fails to halt is a benign algorithm scheme for f.

Conversely, assume $B(x, \delta)$ is a benign algorithm scheme for f with time at most $(|x|/\delta)^c$. Then let A be the algorithm that simulates B with parameters $\delta = 1/2,\ 1/4,\ 1/8, \ldots$ until an answer is given. The expectation of the power $1/2c$ of the time of A on inputs from μ_n is then at most: $(2n)^{1/2} + 1/2(4n)^{1/2} + 1/4(8n)^{1/2} + \ldots = n^{1/2}(\sum_i (2^{-i/2}) = O(n^{1/2})$., since at most $1/2$ of the inputs run for more than one iteration, at most $1/4$ more than two iterations, etc. So A is a polynomial on average algorithm for f $\Box$,

DEFINITION 3.3: A distribution ensemble μ_n is *samplable* if there is a probabilistic polynomial-time algorithm A that on input 2^n produces outputs distributed according to μ_n. The class $DistNP$ is the class of distributional problems in NP where the input distribution is samplable.

Proposition 3: If every problem in $DistNP$ has a polynomial-time benign error algorithm that produces an output with probability $1 - 1/n^2$, then $DistNP \subseteq AvgP$.

Sketch: We reduce finding a benign algorithm scheme for the problem to finding a $1/n^2$ benign error algorithm for the same problem but a slightly different input distribution. In the second problem, you pick an input by picking a random n' from 1 to n amd then sampling according to n' as the first problem does. Given an instance from the original problem, and an error parameter δ, we use the $1/n^2$ benign error algorithm on the input distribution for $n = 1/\delta$.

From this it follows that there is some fixed polynomial p so that there is an algorithm solving one of the average-case complete problems with probability $1 - 1/p(n)$ and only making benign faults, then $DistNP \subseteq AvgP$.

3.3 Extensions

Rephrasing Levin's definition in this light gives us some insight into extensions. The first obvious extension is to change our model from deterministic to probabilistic computation. There are several ways of doing this. The first would be to insist that all errors be benign on all random inputs of the algorithm . I call the resulting class $AvgZPP$, for average case, zero-error probabilistic algorithms. Then it is relatively easy to use results of [NW] to prove the following:

Proposition 4: If $DistNP \subseteq AvgZPP$ then $BPP = ZPP$.

However, this is saying less about the average case hardness of problems in NP then about error-free vs. error prone randomized computation. For example, it is an open problem whether $DistBPP \subseteq AvgZPP$, but a problem in BPP should not be considered hard on average instances! Thus we could define an average-case version of BPP:

DEFINITION 3.4: A probabilistic algorithm returning output possibly ? is *statistically benign* for decision problem f if on any input, the probability that the algorithm returns an answer other than $f(x)$ is at most 1/3. Similarly for a statistically benign algorithm scheme. The class of distributional problems which have poly-time statistically benign algorithm schemes is called $AvgBPP$.

It is also easy to present a non-uniform version of $AvgP$ in the obvious way, which we will call $AvgP/poly$.

However, even these more robust definitions fail to bridge the gap between what is not easy and what is hard. This gap is largely caused by the insistence on the algorithm making only benign errors.

DEFINITION 3.5: An *algorithm scheme* for a distributional problem is an algorithm $A(x, \delta)$ so that for x chosen according to the distribution ensemble and any fixed $\delta > 0$, the probability that A fails to return a correct answer is at most δ. HP for heuristic polynomial-time is the class of distributional problems with a deterministic poly-time algorithm scheme, and similarly HPP is the class of distributional problems with a probabilistic poly-time algorithm scheme, and $HP/poly$ with a non-uniform algorithm scheme.

To get some idea for the difference, [NW] shows how to use any problem in $DistNP$ but not in $HP/poly$ for derandomization. [IR2] was able to construct an oracle where $DistNP \subseteq HP$ but $NP \not\subseteq P/poly$, but the same for $AvgP/poly$ is not known. However, many of the reductions between average-case problems work equally well for the heuristic classes as for the average-case classes. Investigating the differences between the average-case and heuristic distributional classes is another important research direction.

References

[BGS] T.Baker, J. Gill and R. Solovay Relativizations of the P=NP question, SIAM J. Comput., 1975, pp. 431-442.

[BBR] Bennett, C., Brassard, G., Robert, J., "Privacy Amplification by Public Discussion", *Siam*

J. on Computing, Vol. 17, No. 2, 1988, pp. 210-229.

[BCGL] S. Ben-David, B. Chor,O. Goldreich, and M. Luby, On the Theory of Average Case Complexity, STOC 22 (1990), 379-386.

[Bra] G. Brassard, Relativized Cryptography, IEEE Trans. Inform. Theory, IT-29 (1983), 877-894.

[DH] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Trans. Inform. Theory*, Vol. 22, 1976, pp. 644-654.

[GGM] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions", *J. of ACM*, Vol. 33, No. 4, 1986, pp. 792-807.

[GMW] Goldreich, O., Micali, S., and Wigderson, A., "Proofs that Yield Nothing But their Validity or All Languages in NP have Zero-Knowledge Proofs", *J. of the ACM*, Vol. 38, No. 3, July 1991, pp. 691–729.

[G1] Y. Gurevich The Challenger-Solver Game Bulletin of the EATCS, October, 1991.

[G2] Y. Gurevich Average case completeness JCSS

[G3] Y. Gurevich Matrix block decomposition is complete for the average case 31'st FOCS, 1990, pp. 802-811.

[HILL] J. Hastad, R. Impagliazzo,L. Levin, and M. Luby, Pseudo-Random Generators Based on One-Way Functions. To appear, *SIAM Journal of Computing.*

[ILe] R. Impagliazzo and L. Levin, No Better Ways of Finding Hard NP-Problems Than Picking Uniformly at Random. *Proceedings of the* 31*'st IEEE Symposium on Foundations of Computer Science, 1990.*

[ILu] R. Impagliazzo and M. Luby, One-Way Functions are Essential for Complexity Based Cryptography. *Proceedings of the* 30*'th IEEE Symposium on Foundations of Computer Science, 1989.*

[IR] R. Impagliazzo and S. Rudich, Limits on the Provable Consequences of One-Way Functions. *Proceedings,* 20*'th ACM Symposium on Theory of Computing, 1989* .

[IR2] R. Impagliazzo and S. Rudich, in preparation.

[IZ] R. Impagliazzo and D. Zuckerman, How to Recycle Random Bits. *Proceedings of the* 30*'th IEEE Symposium on Foundations of Computer Science, 1989.*

[L1] L. Levin, Average Case Complete Problems SIAM J. Comput. 15 (1986), 285-286.

[L2] L. Levin. ?

[LR] Luby M., and Rackoff, C., "How to Construct Pseudorandom Permutations From Pseudorandom Functions", *SIAM J. on Computing*, Vol. 17, No. 2, 1988, pp. 373-386.

[NW] N. Nisan and A. Wigderson, Hardness vs. Randomness, JCSS ?

[NY] Naor, M. and Yung, M., "Universal One-way Hash Functions and Their Applications", 21^{rst} *STOC*, 1989, pp 33-43.

[OW] Ostrovsky, R and Wigderson, A., "One-way Functions are Essential for Non-Trivial Zero-Knowledge", 2^{nd} *Israel Symposium on the Theory of Computing and Systems*, 1993, pp. 3-17.

[RSA] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Comm. of the ACM*, Vol. 21, 1978, pp. 120-126.

[Rom] Rompel, J., "One-way Functions are Necessary and Sufficient for Secure Signatures", 22^{nd} *STOC*, 1990, pp 387-394.

[Ru] S. Rudich The Role of Interaction in Public Key Cryptography, Crypto, 91.

[Sh] P. Shor, Algorithms for Quantum Computation: Discrete Logarithms and Factoring, FOCS, 1994.

[St] L. Stockmeyer On approximation algorithms for #P TCS 3, 1977,1-22.

[SW] R. Schuler and O. Watanabe, Towards Average-Case Complexity Analysis of NP Optimization Problems, this proceedings.

[VL] R. Venkatesan and L. Levin Random instances of a graph coloring problem are hard, STOC 20 (1988), 217-222.