2009

Complexity cores in average-case complexity theory

Gopalakrishnan Krishnasamy Sivaprakasam
Iowa State University

Complexity cores in average-case complexity theory

by

K. S. Gopalakrishnan

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Pavan Aduri, Major Professor
Jack H. Lutz
Ananda Weerasinghe

Iowa State University

Ames, Iowa

2009

TABLE OF CONTENTS

ACKNOWLEDGMENTS

First and foremost, I would like to express my heartfelt gratitude and thanks to my advisor Dr. Pavan Aduri for his guidance, patience and support throughout this research and the writing of this thesis.

I would also like to thank the committee members Dr. Jack H. Lutz and Dr. Ananda Weerasinghe for their time, helpful discussions and comments.

ABSTRACT

In average-case complexity theory, one of the interesting questions is whether the existence of worst-case hard problems in NP implies the existence of problems in NP that are hard on average. In other words, 'If P $\neq$ NP then NP $\not\subseteq$ Average-P'. It is not known whether such worst-case to average-case connection exists for NP. However it is known that such connections exist for complexity classes such as EXP and PSPACE. This worst-case to average-case connections for classes such as EXP and PSPACE are obtained via random self-reductions. There is an evidence that techniques used to obtain worst-case to average-case connections for EXP and PSPACE do not work for NP.

In this thesis, we present an approach which may be helpful to establish worst-case and average-case connection for NP. Our approach is based on the notion of complexity cores. The main result is "If P $\neq$ NP and there is a language in NP whose complexity core belongs to NP, then NP $\not\subseteq$ Average-P." Thus to exhibit a worst-case to average-case connection for NP, it suffices to show the existence of a language whose core is in NP.

CHAPTER 1. INTRODUCTION

Computational complexity classifies computational problems into various classes based on the amount resources needed by algorithms. Typical resources are time and space. The amount of resources needed by an algorithm is measured based on the worst-case behavior of algorithms. For example, in the case of time, the longest running time taken by an algorithm on any input of size n is considered. Thus this classification measures the worst-case complexity of computational problems. This classification gives rise to various central complexity classes such as P, NP, PSPACE etc. The class P consists of all problems that can be solved by algorithms whose running time is bounded by a polynomial. Whereas the class NP consists of problems that can be solved by non-deterministic algorithms whose running time is bounded by a polynomial.

Consider a problem that does not belong to P. Every algorithm that decides this problem takes more than polynomial-time on some strings. However, it is possible that there is an algorithm that runs in polynomial-time on most of the inputs, and there are very few instances on which the algorithm takes more than polynomial-time. The average value of the running time could be a polynomial. We know several examples of problems whose average-case running time is smaller than the best-known worst-case running time. For example, consider Hamiltonian path problem. Since this problem is NP-complete, it is believed that the worst-case complexity of this problem is exponential. However, there is an algorithm that solves this problem in expected linear time under the commonly used distribution on random graphs. Another example is the quick sort algorithm: though the worst-case complexity of this algorithm is $O(n^2)$, it runs in $O(n \log n)$ time under a uniform distribution. Because of these examples, as well as ordinary computational practice, the average-case complexity of a computational

problem seems to be a more appropriate measure than the worst-case complexity.

Levin (Lev86) was the first to advocate a general study of average-case complexity. An average-case complexity class consists of pairs (L, μ) called distributional problems, where L is a language and μ is a distribution over strings. Given a distributional problem, one would like to know whether there is an algorithm that solves L in average polynomial-time when the instances arise from the distribution μ. Levin defined a robust notion of what means for the running time of an algorithm to be polynomial on μ-average. A distributional problem is said to be in Average-P if there is an algorithm whose running time is polynomial on μ-average. Thus Average-P is the average-case analogue of the class P. Clearly, this definition depends on the underlying distribution μ. Ideally, we would like μ to capture distributions that arise in practice. Levin again suggested that distributions that can be sampled in polynomial-time best capture distributions that arise in practice. Such distributions are called polynomial-time samplable distributions.

The class DistNP is the class of all distributional problems where the language L belongs to NP and the distribution is a polynomial-time samplable distribution. The question of whether DistNP is in Average-P is the average-case analogue of whether P equals NP.

As mentioned above, there exist computational problems whose average-case complexities and worst-case complexities differ. However, there exist computational problems whose worst-case and average-case complexities are the same. For example, it is known that the problem of computing discrete logarithm is easy on average if and only if it is easy on worst-case. Similarly, computing the permanent of matrix is easy on average if and only if it is easy on worst-case. Such connections have been explored further, and we now have a very good understanding of the worst-case and average-case complexities of various computational problems that lie in higher complexity classes such as EXP and PSPACE. It is known that EXP is easy on average (for probabilistic algorithms) if and only if it is easy in worst-case (for probabilistic algorithms). Similar results are known for PSPACE.

However, we do not know analogous results that connect the worst-case and average-case complexities of the class NP. Suppose every distributional problem in DistNP is easy on

average? Does this mean every language in NP can indeed be solved in polynomial-time? Can we establish a connection between the worst-case and average-case complexities of NP?

Known worst-case to average-case connections for classes such as EXP and PSPACE are primarily obtained via random-self-reductions. A language is random-self-reducible if an instance of the language is reduced to a few randomly chosen instances of the same problem. Worst-case to average-case connection for PSPACE is obtained in two steps: The first step shows that if a language L is random-self-reducible, then its worst-case and average-case complexities are the same. The second step shows that there exist complete languages for PSPACE that are random-self-reducible. This establishes the desired connection.

The above approach suggests that to obtain a worst-case to average-case connection for problems in NP, it suffices to exhibit an NP-complete problem that is random-self-reducible. However, Feigenbaum and Fortnow (FF93) showed that if there is an NP-complete problem that is random-self-reducible, then the polynomial-time hierarchy collapses. Since we believe that the polynomial-time hierarchy is infinite, we cannot hope to exhibit a random-self-reducible NP-complete problem. Thus we cannot use random-self-reductions as tool to establish worst-case to average-case connections for NP. Another technique that is used to obtain connections for classes such as EXP and PSPACE is that of error correcting codes. It is now known that these techniques also cannot be used to obtain connections for NP (Vio04).

Thus we need completely different approaches and techniques to establish worst-case and average-case connections for NP. In this thesis, we present an approach. We do not establish any connections between worst-case and average-case complexities for NP. Our main contribution is the suggestion of an approach that seems to avoid above mentioned obstacles.

Our approach is based on the notion of complexity cores. Consider a language L that is not in P. This means that every algorithm that solves L must run more than polynomial-time on infinitely many strings. Such string are called hard-instances for the given algorithm. Thus for every algorithm that solves L, there is a set of hard-instances. The set of hard instances depend on the algorithm. Can we identify a set of instances that turn out to be hard instances for every algorithm that solves L? Such a set is called the complexity core of L. The notion of

complexity core is first introduced by Lynch (Lyn75). It is known that every language L that is not in P has a complexity core S (Lyn75).

Consider a language L that is not in P, and let S be its complexity core. Let μ be a distribution that places most of its weight on strings from S. Since μ places most of its weight on S and every algorithm that solves L takes more than polynomial-time on strings from S, it must be the case that (L, μ) is not in average-polynomial-time. This suggests the following approach to establish a worst-case to average-case connection for NP. Assume NP does not equal P. Thus there is a language L in NP that is not in P. Consider a distribution that places most of its weight on the complexity core L. With respect to distribution, L is not in average polynomial-time. Thus NP is not Average-P with respect to distribution. Though this argument seems correct, there is a problem. We need to consider the complexity of the distribution μ. We would have established a connection only if μ were a polynomial-time samplable distribution. However, it is not clear that μ can be made polynomial-time samplable.

We show that if the complexity core S of L is in NP, then the distribution μ can indeed be made polynomial-time samplable. This is the main result of this thesis. Thus one way to establish a worst-case to average-case connection for NP is to exhibit a problem in NP whose complexity core is in NP.

Using complexity cores we establish another result. Consider a language L that is not in P. Is it possible that L is in average polynomial-time with respect to every distribution μ? Li and Vitanyi showed that there is a distribution μ such that L is in P if and only if L is in Average-P with respect to μ. However, this distribution is not computable. Schuler showed that there is a language L that is not in P, yet L is in average-polynomial-time with respect to every polynomial-time computable distribution.

We ask the following question: Is there a language L that is not in P, such that L is average-polynomial-time with respect to every P-samplable distribution? We showed that if such a language exists then EXP differs from BPP.

CHAPTER 2. PRELIMINARIES

Let Σ^* be the set of all strings defined on the binary alphabet $\Sigma = \{0, 1\}$. Denote by $\leq$ the standard lexicographical order on Σ^*. Let $x - 1$ denote the immediate predecessor of x. Given a Turing Machine M, $T_M(x)$ defines running time of M.

We denote probability distribution over the set Σ^n with μ_n. Given a string x of length n, $\mu_n(x)$ denotes the probability of x with respect to the distribution, and $\mu'_n(x) = \sum_{y \leq x} \mu'_n(y)$.

We say that $\mu = (\mu_1, \mu_2, \cdots)$ is an *ensemble of distributions* if each μ_n is a distribution over Σ^n. Levin (Lev86) considered a distribution over entire Σ^* rather than an ensemble of distributions. However, it is more convenient to use an ensemble of distributions rather than a single distribution. Gurevich (Gur91) and Impagliazzo (Imp95) noted that Levin's definition of average-polynomial time can be adapted to the case of an ensemble of distributions. In this thesis we follow this adaption.

Ideally, we would like to study the average-case complexity of a problem with respect to distributions that arise in practice. However, it is not easy to precisely capture distributions that arise in practice. Levin suggested that the distributions that arise practice are either polynomial-time computable or can be dominated by polynomial-time computable distributions.

An ensemble of distributions $\mu = (\mu_1, \mu_2, \cdots)$ is polynomial-time computable, if there is a polynomial-time bounded Turing machine M such that M of input x outputs $\mu'_n(x)$, where $|x| = n$.

Ben-David *et al.* (BCGL92) suggested that polynomial-time computability is can be too restrictive and suggested that polynomial-time samplable distributions more accurately capture distributions that arise on practice.

An ensemble $\mu = \{\mu_n\}_{n>0}$is polynomial-time samplable or P-samplable if there is a polynomial-time bounded randomized algorithm A such that

$$\Pr[A(1^n) = x] = \mu_n(x),$$

where the probability is taken over the coin tosses of A.

It is known that every polynomial-time computable distribution is also polynomial-time samplable. However, the converse does not hold if one-way functions exist.

Above definitions of polynomial-time computability and samplability can be generalized. Let $\mathcal{C}$ be a complexity class. We say that a distribution μ is $\mathcal{C}$-computable if there is Turing machine M that runs within the resource bounds of $\mathcal{C}$, and $M(x)$ outputs the value of $\mu'_{|x|}(x)$. Similarly if there is a probabilistic algorithm that runs within the resource bounds of $\mathcal{C}$ that outputs strings according the distributing μ, then we say that μ is $\mathcal{C}$-samplable.

Definition 1. The Uniform ensemble of distributions $U = (U_1, U_2, \cdots)$ is defined as

$$\forall n, x \in \Sigma^n, U_n(x) = \frac{1}{2^n}.$$

Definition 2. A Distributional problem is a pair (L, μ) where L is a language and μ is an ensemble of distributions.

2.1 Levin's definition of average polynomial time

The notion of average polynomial time was introduced by Levin (Lev86). Let M be the deterministic Turing machine which decides a language L and μ be an ensemble of distributions. Let $T_M(x)$ be the running time of M on input x. A naive way to define average-polynomial time is as follows: For all but finitely many n, $\sum_{x\in\Sigma^n} T_M(x)\mu(x) \leq p(n)$ for some polynomial $p(n)$. However, this definition does not turn out to be model independent. For example consider the following.

Let $A_n \subseteq \Sigma^n$ with $|A_n \cap \Sigma^n| = 2^n - n$. Also let

$$T_M(x) = \begin{cases} |x|, & \text{if } x \in A_{|x|}, \\ 2^{|x|}, & \text{otherwise.} \end{cases}$$

Consider the uniform ensemble of distributions $U = (U_1, U_2, \cdots)$. It is easy to check T_M satisfies the above definition of average polynomial time. However, T_M^2 is not average-polynomial. Thus if we consider a model on that is quadratically slower than the Turing machine model, the language may not be in average-polynomial on this model.

Now we state Levin's definition.

Definition 3. The running time T_M of a deterministic Turing machine is in average polynomial time with respect to a distribution μ if there exist a positive integer k such that

$$\forall n, \sum_{|x|=n} T_M^{\frac{1}{k}}(x)\mu_n(x) \leq O(n).$$

Definition 4. A distributional problem (L, μ) is solvable in average polynomial time if there exists a deterministic Turing machine M that decides L whose running time is average polynomial with respect to the distribution μ.

Definition 5. The complexity class Average-P is the class of distributional problems that are solvable in average polynomial time.

Lemma 1. *Let L be a language decidable by a deterministic Turing machine M and μ be a P-samplable distribution. If* $(L, \mu) \in$ Average-P *then* $\forall l, \exists k, \forall n$, *where* $l, k, n \in N$ *such that*

$$\Pr_{x \in \mu_n} (T_M(x) > n^k) < \frac{1}{n^l}.$$

Proof. We prove this theorem by contradiction. Given (L, μ) is in Average-P. By definition, $\exists k$ and $\forall n$, where $k, n \in N$

$$\sum_{x \in \Sigma^n} \mu_n(x)(T_M^{\frac{1}{k}}(x)) \leq O(n) \tag{2.1}$$

Assume $\exists l, \forall k_1, \exists n$, where $l, k_1, n \in N$ such that

$$\Pr_{x \in \mu_n} (T_M(x) > n^{k_1}) \geq \frac{1}{n^l} \tag{2.2}$$

Fix k_1 as $(l+2)k$ and Let $A = \{x | T_M(x) > n^{k_1}\}$. Therefore 2.1 may be written as

$$\sum_{x \in \Sigma^n - A} \mu_n(x)(T_M^{\frac{1}{k}}(x)) + \sum_{x \in A} \mu_n(x)(T_M^{\frac{1}{k}}(x)) \leq O(n) \tag{2.3}$$

Consider the sum

$$\sum_{x \in A} \mu_n(x)(T_M^{\frac{1}{k}}(x)) \tag{2.4}$$

Using 2.2 in 2.4, we have

$$\sum_{x \in A} \mu_n(x)(T_M^{\frac{1}{k}}(x)) > \sum_{x \in A} \mu_n(x) n^{\frac{k_1}{k}}$$

i.e.

$$\sum_{x \in A} \mu_n(x)(T_M^{\frac{1}{k}}(x)) \geq n^2 \tag{2.5}$$

Here 2.5 can not be bounded by $O(n)$.

This is a contradiction to the fact $(L, \mu) \in$ Average-P.

Therefore our assumption is wrong. Hence the lemma. □

Definition 6. Let DistNP be the class of all distributional problems (L, μ) where $L \in$ NP and μ is a polynomial-time samplable.

2.2 Complexity Cores

Let L be a language that does not belong to P. This means that every algorithm that decides L will encounter hard instances—instances on which the algorithm takes more than polynomial-time. It is natural to ask whether there exist a set of instances that are hard for every algorithm. The notion of complexity cores proposed by Lynch (Lyn75) makes this precise.

Definition 7. Let $L \notin$ P. An infinite set S is a complexity core for L if for every Turing machine M that decides L and every polynomial p, M runs for more than $p(n)$ time on all but finitely many of the strings in S.

Lynch showed that every set that is not in P has a complexity core.

Theorem 1. *If $L \notin$ P, then L has an infinite complexity core S. Moreover S can be recognized in time $n^{\log n}$.*

2.3 2-Universal Hash Functions

Definition 8. Let $\mathcal{H}$ be a set of functions from U to T. $\mathcal{H}$ is a 2-universal family of functions if $\forall x \neq y \in U, \alpha, \beta \in T$

$$Pr_{h \in H}[h(x) = \alpha \wedge h(y) = \beta] = \frac{1}{|T|^2}.$$

Clearly the set of all functions from U to T is 2-universal. We are interested in a class of 2-universal hash functions whose cardinality is small. It is known that such a class of hash functions exist. Below we give an example of a class of 2-universal hash functions from Σ^n to Σ^m.

Let M be a $m \times n$ boolean matrix. Given such matrix M, define a function $h_M : \Sigma^n \rightarrow \Sigma^m$ as follows: $h_M(x) = Mx$. Here x is taken as a column vector and the multiplication is done over $GF(2)$. Let $\mathcal{H} = \{h_M | M \text{ is a } m \times n \text{ boolean matrix}\}$. It can be shown that $\mathcal{H}$ is 2-universal. Observe that the size of $\mathcal{H}$ is 2^{mn}.

Let $\mathcal{H}$ be a family of 2-universal hash functions from Σ^n to Σ^{k+1}. Let S be a subset of Σ^n whose cardinality is at most 2^k. The following theorem, known as the isolation theorem, is due to Valiant and Vazirani (VV86).

Theorem 2. *If we randomly pick $h \in \mathcal{H}$, that at least $\frac{|S|}{3}$ strings from Σ^{k+1} have exactly one inverse in S, with respect to h, with probability $\geq \frac{1}{4}$.*

CHAPTER 3. MAIN THEOREMS

Given a complexity class $\mathcal{C}$, we say that $\mathcal{C}$ is easy on average, if for every language $L \in \mathcal{C}$, for every p-samplable distribution μ, the distributional problem (L, μ) is in Average-P. A complexity class is easy on worst-case if $\mathcal{C} \subseteq$ P.

We believe that the complexity class NP does not equal to P, i.e, NP is not easy on average. Can it be the case that NP is not easy in worst-case yet it is easy on average? If that were the case, then in practice we have efficient algorithms for all problems in NP.

Though we know some NP-complete problems such as Halmiltonian cycle have efficient average-case algorithms (under the uniform distribution), we do not know efficient average-case algorithms for quite a few NP-complete problems. For example, we do not know if SAT has an efficient average-case algorithm under an uniform distribution. This indicates that NP may not be easy on average. Can we gain more evidence for this? One way to achieve this is to establish a connection between the worst-case complexity and average-case complexity for NP, i.e., prove a theorem of the following form: If P $\neq$ NP, then NP is not easy on average.

Such worst-case to average-case connection have been established for complexity classes such as PSPACE, #P, and EXP. In the case of EXP, one has to consider average-case complexity with respect to circuits or probabilistic algorithms. For example, we know that if PSPACE is easy on average, then PSPACE can be solved in probabilistic polynomial-time.

Can we establish a similar worst-case to average-case connection for NP? This is one of the most outstanding open questions in complexity theory. The techniques used to in the context of PSPACE, or EXP does not seem to be applicable to the case of NP. For example, it has been showed that if similar techniques can be used for the case of NP, then the polynomial-time hierarchy collapses—an unlikely consequence. Thus we need an entirely different approach to

solve this problem. In the first part of the thesis, we present an approach that is based on complexity cores.

3.1 Connections via Complexity Cores

Definition 9. Let $\mu = (\mu_1, \mu_2, ..., \mu_n, ...)$ be an ensemble of probability distributions over Σ^*. Let S be a subset of Σ^*. We say that μ places a non negligible weight on S if

$$\mu_n(S) \geq \frac{1}{n^2} \; when \; \Sigma^n \cap S \neq \emptyset.$$

By Theorem 1, every language that is not in P admits a complexity core. We will show that if μ is a distribution that places non-negligible weight on its complexity core, then L can not be solved in average polynomial-time with respect to μ.

Theorem 3. *Let $L \notin$ P and S be its complexity core. Suppose that a distribution μ places a non negligible weight on S then the distributional problem $(L, \mu) \notin$ Average-P.*

Proof. Assume $(L, \mu) \in$ Average-P, let M be a Turing machine M that decides L whose running time is polynomial on average with respect to μ. Given n, let By Lemma 1, there is a constant k such that for all but finitely m,any n,

$$\Pr_{x \in \mu_n}[H_n] \quad < \quad \frac{1}{n^3}, \tag{3.1}$$

where

$$H_n = \{x \mid T_M(x) > n^k, |x| = n\}.$$

Since S is the complexity core of L, there is a set S_1 such that for M takes more than n^k time on all strings from S_1 and S_1 contains all but finitely many strings from S. Thus there is a constant N such that for all $n \geq N$, $S^n = S_1^n$. This means that for every $n \geq N$, $S_1^n \subseteq H_n$. Moreover, for every $n \geq N$, $\mu(S^n) = \mu(S_1^n)$. Since S is infinite, there exist infinitely many $n \geq N$ for which $\mu(S^n) \geq 1/n^2$, thus for all such n, $\mu(S_1^n) \geq 1/n^2$. However, by Inequality 3.1, $\mu(H_n) < 1/n^3$ for every $n \geq N$. Since $S_1^n \subseteq H_n$, we have a contradiction. □

Thus a language can not be easy on average with respect to any distribution that places a non negligible weight on its complexity core. This suggests the following approach to establish a worst-case to average-case connection for NP. Let L be a language that is not in P and S be its complexity core. By previous theorem if μ were a distribution that places a non-negligible weight on S, then (L, μ) is not in Average-P. If we can make μ to be p-samplable, then we are done. How can we make μ to be p-samplable?

In general it is not possible. If S were a set of arbitrary complexity, then we can not hope to have a p-samplable distribution that places a non-negligible weight on S. We will first show that if S is "easy enough", then we can define define distributions that are $\mathrm{P}_{tt}^{\mathrm{NP}}$-samplable. We then show that if NP is easy on average, then we can convert those $\mathrm{P}_{tt}^{\mathrm{NP}}$-samplable distributions into P-samplable distributions.

We first consider the case when S in NP and has at most one string at each length.

Theorem 4. *Let $S \subseteq \Sigma^*$. Suppose $S \in$ NP and for all n, $|S \cap \Sigma^n| \leq 1$, then there exists a $\mathrm{P}_{tt}^{\mathrm{NP}}$-samplable distribution that places most of its weight on S.*

Proof. Consider the following language. Define

$$L = \{\langle 1^n, i\rangle \mid \exists x \text{ of length n in S whose } i^{th} \text{ bit is one}\}.$$

Clearly $L \in$ NP.

Consider the following algorithm that runs in polynomial-time and makes non-adaptive queries to L.

1. Input 1^n
2. For i = 1, 2, ..., n
3. If $\langle 1^n, i\rangle \in L$, then set $x_i = 1$
4. Else $x_i = 0$
5. Endfor
6. Output $x_1 x_2 ... x_n$

Consider an input length n. If $S^n = \emptyset$, then for every i, $1 \leq i \leq n$, $\langle 1^n, i\rangle \notin L$. Thus the above algorithm outputs 0^n. Suppose $S^n \neq \emptyset$. Let $y = y_1 y_2 \cdots y_n$ be the unique string that is

in S^n. Note that if $y_i = 1$, then $\langle 1^n, i\rangle \in L$, else $\langle 1^n, i\rangle \notin L$. Thus knowing the membership of $\langle 1^n, i\rangle$ determines the ith bit of y. The above algorithm finds the ith bit by querying the membership of $\langle 1^n, i\rangle$ in L. This is correctly computes the each bit of y, and so it outputs the string y.

Observe that the above algorithm does not use any randomness. So if $S^n \neq \emptyset$, then the probability that it outputs a string from S^n is one. If we let μ_n to be the distribution sampled by the above algorithm, then the ensemble $\mu = (\mu_1, \mu_2, \cdots)$ is a distribution that places a non-negligible weight on S.

Clearly the above algorithm runs in polynomial-time and makes queries to the language L. Note that all the queries are non-adaptive. This μ is a $\mathrm{P}^{\mathrm{NP}}_{tt}$-samplable distribution. □

In the above, we assumed that S has at most one string at each length. We now relax this condition. We will show that if S is any set in NP, then there is a $\mathrm{P}^{\mathrm{NP}}_{tt}$-samplable distribution that places a non-negligible weight on S. For this we will use the isolation theorem of Valiant and Vazirani.

Theorem 5. *Let $S \in$ NP. There is a $\mathrm{P}^{\mathrm{NP}}_{tt}$- samplable distribution that places non-negligible weight on S.*

Proof. We fix a small class of 2-universal hash functions. Let H^k_n be a class of 2-universal hash function from Σ^n to Σ^k such that the size of H^k_n is $2^{poly(n,k)}$. We can use the a family defined in the preliminaries.

We define a language L. The language consists tuples of the form $\langle 1^n, i, 1^k, h, \alpha\rangle$. Such a tuple belongs to L if all of the following conditions hold:

- $1 \leq i \leq n$, $1 \leq k \leq n$.
- $h \in H^{k+2}_n$, $\alpha \in \Sigma^{k+2}$.
- There is a string x of length n from S such that $h(x) = \alpha$ and the ith bit of x is one.

Since S is NP, and the hash functions from the family H^k_n can be easily computed, the language L is in NP. Now consider the following probabilistic algorithm that attempts to generate strings from S.

1. Input 1^n
2. Pick k uniformly at random from $\{1, 2, ..., n\}$
3. Pick h uniformly at random from H_n^{k+2}
4. Pick α at uniformly at random from Σ^{k+2}
5. For i = 1, 2, ..., n
6. If $\langle 1^n, i, 1^k, h, \alpha\rangle \in L$, then set $x_i = 1$
7. Else $x_i = 0$
8. Endfor
9. Output $x_1x_2...x_n$

Clearly the above algorithm runs in polynomial-time by making non-adaptive queries to L. Thus it is defining a $\mathrm{P}_{tt}^{\mathrm{NP}}$-samplable distribution μ.

Fix an input length n at which $S^n \neq \emptyset$. We will show that the above algorithm outputs a string from S^n with probability at least $1/n^2$. This shows that μ places a non-negligible weight on S.

Since $S^n \subseteq \Sigma^n$, there is a constant s, $1 \leq s \leq n$ such that $2^s \leq |S^n| \leq 2^{s+1}$. When k is picked uniformly at random from $\{1, 2, \cdots, n\}$, k will be s with probability $1/n$. From now assume that $s = k$. By Theorem 2, if we randomly pick a hash function h from H_n^{k+2}, then with probability at least $1/4$, at least $|S^n|/3$ elements from Σ^{k+2} will have exactly one inverse from S with respect to h. Consider the case when h satisfies this property.

Now the algorithm picks α uniformly at random from Σ^{k+2}. Since $|S^n| \geq 2^k$, with probability at least $1/12$, α has exactly one inverse from S^n with respect to h. Consider the case when α satisfies this property.

To recap, we have the following scenario: The algorithm randomly picked k, h, and α, and α has exactly one inverse in S^n with respect h. This event happens with probability at least $\frac{1}{n} \times \frac{1}{4} \times \frac{1}{12} = \frac{1}{48n}$.

Let $y = y_1y_2 \cdots y_n$ be the unique string from S^n such that $h(y) = \alpha$. Now, the ith bit of y is one if and only if the tuple $\langle 1^n, i, 1^k, h, \alpha\rangle$ belongs to L. Thus knowing the membership of $\langle 1^n, i, 1^k, h, \alpha\rangle$ in L helps determine y_i.

Since the above algorithm makes queries to L about tuples $\langle 1^n, i, h, 1^k, \alpha \rangle$, $1 \leq i \leq n$, it correctly constructs y. Finally the algorithm outputs y. It is guaranteed that $y \in S^n$.

Thus the above algorithm outputs a string from S^n with probability at least $\frac{1}{48n} \geq \frac{1}{n^2}$.

Thus the distribution defined by the above algorithm places a non-negligible weight on S. □

Now we are ready to present our main result. We will show that if P does not equal to NP, and there is language in NP whose complexity core is in NP, then NP is not easy on average. Thus to prove a worst-case to average-case connection for NP, it suffices to show that some language in NP has a complexity core that can be recognized in NP.

Theorem 6. *If* P $\neq$ NP *and there exist a language* $L \in$ NP *that has a complexity core in* NP*, then* NP *is not easy on average.*

Proof. Let L' be a language in NP, and let S be its complexity core that can be recognized in NP. Let L be a language that is defined as in the proof of the previous theorem. That is, L consists of tuples of the form $\langle 1^n, i, 1^k, h, \alpha \rangle$.

Now for the sake of contradiction, assume that NP is easy on average. So, for every p-samplable distribution μ, the language L is easy on average. Consider the following distribution μ: The distribution μ_n is defined by the following process:

- Pick i uniformly at random from $\{1, 2, ..., n\}$
- Pick k uniformly at random from $\{1, 2, ..., n\}$
- Pick h uniformly at random from H_n^{k+2}
- Pick α at uniformly at random from Σ^{k+2}
- Output $\langle 1^n, i, 1^k, h, \alpha \rangle$

Clearly μ is p-samplable. Thus (L, μ) is in average polynomial-time. Let M be machine that witnesses this. The following claim can be shown. The proof is similar to the proof of Lemma 1.

Claim 1. *There is a constant k such that for all but finitely many n*

$$\Pr[\exists i,\ 1 \leq i \leq n,\ T_M(\langle 1^n, i, 1^k, h, \alpha\rangle) > n^k] \leq 1/n^3,$$

where the probability is defined by making uniform random choices for k, h, and α.

Thus if we randomly pick k, h and α, then for all i, $1 \leq i \leq n$, the machine M runs in time n^k time with probability at last $1 - 1/n^2$.

Now consider the P^L_{tt}-samplable distribution ν that places non-negligible weight in S. Define a new distribution ρ as follows: On input 1^n, simulate the distribution ν_n. Whenever ν makes a query the language L, simulate the machine M for exactly n^k steps. If M does not halt within n^k steps, then output 0^n. Clearly this algorithm runs in polynomial-time. Thus ρ is a p-samplable distribution. We now argue that ρ places a non-negligible weight on the complexity core S.

By previous theorem, we know that if M halts with in n^k steps for all possible inputs, then this algorithm outputs a string from S^n with probability at least $1/48n$. However, the probability that M does not halt within n^ks steps is at most $1/n^3$, when this happens the distribution may not output a string from S^n. Thus the total probability that the machine may not output a string from S^n is at most $1 - \frac{1}{48n} + \frac{1}{n^3}$ this is at most $1 - 1/n^2$. Thus ρ_n places a weight of at least $1/n^2$ on S^n. Thus ρ places a non-negligible weight on S.

Since S is the complexity core of L, by Theorem 3, we know that (L, ρ) is not in Average-P. Since L is in NP and ρ is p-samplable, by our assumption that NP is easy on average, (L, μ) must be in Average-P. This is a contradiction. □

3.2 Languages that are easy with respect to all distributions

Clearly every language L in P is easy on average with respect to every distribution. Is there a language that is not easy in the worst-case, and yet L is easy on average with respect to all distributions? If we consider all possible distributions, then the answer is "No". It is known that there is a distribution μ such that the worst-case complexity of L is the same as the average-case complexity of L with respect to μ (LV97). This distribution is defined based

on Kolmogorov complexity, and is uncomputable. What happens when we restrict attention to computable distributions? What is the smallest class of distributions for which the average complexity of a problem coincides with the worst-case complexity. We provide answers to these questions.

Given a complexity class$\mathcal{C}$, define a new class $\exists_{poly}\mathcal{C}$ as follows: A language L belongs to $\exists_{poly}\mathcal{C}$ if there is a language L' in $\mathcal{C}$ and a polynomial p such that for every n, for all strings x in Σ^n

$$x \in L \Leftrightarrow \exists y, |y| = p(|x|), \langle x, y\rangle \in L'.$$

Let QP denote the class of languages that can be decide in time $O(n^{\log n})$.

Based on previous techniques, we show that for every language L that is not in P, there is a $P_{tt}^{\exists_{poly}\mathrm{QP}}$-samplable distribution μ such that L is in P if and only if L is in Average-P with respect to μ.

Theorem 7. *If $L \notin P$, then there exist $P_{tt}^{\exists_{poly}\mathrm{QP}}$-samplable distribution μ such that (L, μ) is not in* Average-P.

Proof. In Theorem 5, we showed that if a set S belongs to NP, then there is a P^{NP}-samplable distribution that places a non-negligible weight on S. This theorem can be extended to show that if S belongs to a complexity class $\mathcal{C}$, then there is a $\mathrm{P}_{tt}^{\exists_{poly}\mathcal{C}}$-samplable distribution that places a non-negligible weight in S. Thus if S is in QP, then there is a $\mathrm{P}_{tt}^{\exists_{poly}\mathrm{QP}}$-samplable distribution that places a non-negligible weight on S.

By Theorem 1 every language that is not in P has a complexity core that is S decidable in time $n^{\log n}$. With respect to any distribution that places a non-negligible weight on S, the language L can not be easy on average. So there is a $\mathrm{P}_{tt}^{\exists_{poly}\mathrm{QP}}$-samplable distribution μ such that (L, μ) is not easy on average. □

Thus for every language L that is not in P, there is a computable distribution μ such that (L, μ) is not in Average-P. Since $\exists_{poly}\mathrm{QP} \subseteq \mathrm{EXP}$, this distribution is EXP-computable. Can we reduce the complexity of this distribution? More specifically, can we make it p-computable or P-samplable?

Interestingly, for the case of p-computable distributions the answer is "No". Schuler (Sch95) showed that there is a language L that is not in P, and (L, μ) is in Average-P with respect to every p-computable distribution.

Can we extend Schuler's theorem to the case of p-samplable distributions. Our next theorem shows that that would imply a separation of EXP from BPP.

Theorem 8. *Suppose* EXP = BPP*, then for every language L that is not in* P*, there is a p-samplable distribution μ such that (L, μ) is not in* Average-P*.*

Proof. Let L be a language that is not in P. By previous theorem, there is $\mathrm{P}^{\exists_{poly}\mathrm{QP}}$-samplable distribution μ such that (L, μ) is not in Average-P. Since $\exists_{poly}\mathrm{QP} \subseteq$ EXP, if EXP = BPP, then we can replace all queries to $\exists_{poly}\mathrm{QP}$ to a language in BPP. Since $\mathrm{P}^{\mathrm{BPP}}$ is in BPP, we can make μ to be p-samplable. □

BIBLIOGRAPHY

[BCGL92] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193-219, 1992.

[FF93] J. Feigenbaum and L. Fortnow. Random-self reducibility of complete sets. *SIAM Journal on Computing*, 22:994-1005, 1993.

[Gur91] Y. Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42(3):346-398, 1991.

[Imp95] R. Impagliazzo. A personal view of average-case complexity theory. *In proceedings of the 10th Annual Conference on Structure in Complexity Theory*, pages 134-147, IEEE Computer Society Press, 1995.

[Lev86] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285-286, 1986.

[LV97] M. Li and P. Vitanyi. *An introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, 1997.

[Lyn75] N. Lynch. On reducibility to complex or sparse sets. *J.ACM 22*, 3:341-345, 1975.

[Sch95] R. Schuler. Some properties of sets tractable under every polynomial-time computable distribution. *Information Processing Letters*, 55:179-184, 1995.

[Vio04] E. Viola. The complexity of constructing pseudo random generators from hard functions. *Journal Computational Complexity*, 13(3-4):147-188, 2004.

[VV86] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theor.Comput.Sci.*, 47(1):85-93, 1986.