Universidade Técnica de Lisboa

Instituto Superior Técnico

Departamento de Matemática

Average Complexity

Tiago Salvador

Mathematics Project
Licenciatura em Matemática Aplicada e Computação

Supervisors:
Prof. Carlos Caleiro

2010

Contents

1 Introduction

We begin our discussion by introducing some basic definitions of the Theory of Complexity. Let Σ be a finite alphabet (that is, a finite nonempty set) with at least two elements, and let Σ^* be the set of finite strings over Σ. Then a language over Σ is a subset L of Σ^*. A decision problem in Σ^* is simply a function $\chi : \Sigma^* \mapsto \{0,1\}$. To each language L we associate the obvious decision problem, also denoted by L, such that, for each $x \in \Sigma^*$, $L(x) = 1$ if $x \in L$ and $L(x) = 0$ otherwise. Given a binary relation $S \subseteq \Sigma^* \times \Sigma^*$, a search problem for S, also denoted by S, is a function such that for each $x \in \Sigma^*$, $S(x) = \perp$ if there is no $y \in \Sigma^*$ such that $(x, y) \in S$ and $S(x) = z$ if there is a y such that $(x, y) \in S$ and $(x, z) \in S$. Note that there may be multiple y such that $(x, y) \in S$. The unique solution search problem of S is to find, on input x, the unique y satisfying $(x, y) \in S$. If no such a y exists or it is not unique then nothing is required. In our discussion, typically, $\Sigma = \{0,1\}$ and naturally a language L will be a subset of $\{0,1\}^*$.

Let $\prec$ denote the following order in $\{0,1\}^*$: $x \prec y$ if and only if $|x| < |y|$ or $|x| = |y|$ and there is n such that $x_n < y_n$ and for every $i < n$, $x_i = y_i$. Naturally, $x \preceq y$ if and only if $x = y$ or $x \prec y$.

For technical reasons, the algorithms in this paper are given, in addition to the input x, a parameter n corresponding to the distribution D_n from which x was sampled. We write $A\,(x; n)$ to denote the output of algorithm A on input x and parameter n.

Definition 1.1 *We say that an algorithm A is a polynomial-time algorithm for a decision problem L if there is a polynomial p such that for every $x \in \{0,1\}^n$, $A\,(x; n)$ runs in time at most p(n) and outputs L(x).*

Definition 1.2 *We define* **P** *to be the class of decision problems that admit a polynomial-time algorithm.*

Definition 1.3 *We say that $S \subseteq \Sigma^* \times \Sigma^*$ is a NP-relation if $L_S \in \mathbf{P}$ where $L_S = \{x\#y : (x, y) \in S\}$ is a language over $\Sigma \cup \{\#\}$ and $\# \notin \Sigma$.*

Intuitively, S is a NP-relation, if given $x, y \in \Sigma^*$ we can check in polynomial time whether or not $(x, y) \in S$.

Definition 1.4 **NP** *is the class of decision problems L for which there exists a NP-relation* $S \subseteq \Sigma^* \times \Sigma^*$ *and* $k \in \mathbb{N}$ *such that*

- *For every* $x \in \Sigma^*$, $x \in L \Leftrightarrow \exists_{y \in \Sigma^*} \Big((x, y) \in S \wedge |y| \leq |x|^k \Big)$.

In our discussion, given a language $L \in \mathbf{NP}$ by an implicit NP-relation we understand a NP-relation S such that $L = \{x : \exists_{w \in \Sigma^*} (x, w) \in S\}$. Also, when we say "a search algorithm for L", we mean an algorithm for the search problem S, that is, an algorithm that on input $x \in L$ outputs a witness w that x is a member of L, i.e., a w such that $(x, w) \in S$. We call such a w a L-witness for x and S the implicit search problem for L.

It is clear that if one has an efficient search algorithm for a language $L \in \mathbf{NP}$ then immediately we have an efficient decision algorithm for L. However, the converse is not believed to be true in general, for instance, if one-way functions exists. In fact, the converse is true if L is **NP-complete**.

Definition 1.5 *L is* **NP*-complete*** *if and only if the following two conditions are satisfied:*

- $L \in \mathbf{NP}$
- *for any* $L' \in \mathbf{NP}$, *L' is polynomial-time reducible to L (written as* $L' \leq_p L$*), where* $L' \leq_p L$ *if and only if:*
  - *there is a function* $f : \Sigma^* \mapsto \Sigma^*$ *such that* $\forall_{w \in \Sigma^*} w \in L' \Leftrightarrow f(w) \in L$
  - *there is a polynomial-time algorithm A such that on input w outputs f(w).*

The efficient search algorithm for L, where L is **NP-complete** and has an efficient decision algorithm, is constructed based on the following idea: on input x, the solution y is found by asking queries of the form "is p a prefix of a solution to x?".

Definition 1.6 ***NP-search*** *is the class of search problems S for which S is a NP-relation.*

Observe that if $S \in$ **NP-search** then $L(S) \stackrel{\text{def}}{=} \{x : \exists_{y \in \Sigma^*} (x, y) \in S\} \in \mathbf{NP}$.

The study of the average-case complexity of intractable problems began in the 1970s motivated by developments on the foundations of cryptography and the search for methods to deal with the intractability of NP-hard problems.

There was an established idea that NP-complete problems were hard to solve and so there was no chance to have good algorithms to solve them. However, that's not entirely true since a problem may not have any efficient worst-case algorithm but still it may be possible to solve it for "most" instances. In order to deal with the limitations of a worst-case approach, one needs in fact an average-case approach. Along with the problem itself, there must be a probability distribution to characterize the instances in order to express what instances arise in practice.

The idea of having an algorithm that solves our problem for typical instances is very important in cryptography. The security of modern cryptosystems is based on the difficulty of solving an underlying problem. Intuitively, if any efficient algorithm that tries to solve the problem only succeeds with very small probability, our crytosystems are secure. Hence, in cryptography we are not only interested in the existence of hard problems but also that most of its instances are hard to solve.

We can now roughly state what the concept of "efficient on average" should mean: by an "efficient on average" algorithm, we intend an algorithm that solves our problem efficiently for common instances and that is allowed to be inefficient for instances that occur with very small probability.

In Chapter 2 we present the basic definitions of the average-case complexity as well as the subtle difficulties that surprisingly arise. In Chapter 3 we discuss the relation between search and decision problems, presenting a theorem of Ben-David et al. and respective proof. Finally in Chapter 4 we make some general remarks on the average-case complexity.

2 Definitions of "Efficient on Average"

2.1 Distribution over Inputs

Definition 2.1 *Let L be a language and let $\mathcal{D}$ be the ensemble $\mathcal{D}=\{D_n\}_{n>0}$. For each n, D_n is a distribution over $\{0,1\}^{d(n)}$ and*

$$\forall_{x\in\{0,1\}^{d(n)}} \frac{D_{n+1}(f(x))}{\sum_{y\in\{0,1\}^{d(n)}} D_{n+1}(f(y))} = D_n(x)$$

where d is a polynomial and f is a function such that:

- *f is injective;*
- *f is invertible;*
- *for every n, if $x \in \{0,1\}^{d(n)}$ then $f(x) \in \{0,1\}^{d(n+1)}$.*

We call the pair $(L, \mathcal{D})$ a distributional decision problem.

When dealing with tuples of strings from differents ensembles, the function f for the tuples will be the aggregate function, that is $\langle f_1, \ldots, f_n\rangle$. Typically, f will be a padding function.

There is another common convention on how to specify $\mathcal{D}$. We can define $\mathcal{D}$ as a probability distribution over the set $\{0,1\}^*$ of all possible bit strings. This definition leads to a simple definition of reduction preserving average-case algorithms and under some conditions is equivalent to ours. On the other hand, our definition is common in cryptography and derandomization.

In our discussion, we will consider ensembles for which there is a polynomial-time algorithm that given an integer n produces samples from the distribution D_n. We can formalize this idea in the following way:

Definition 2.2 *As ensemble $\mathcal{D}=\{D_n\}_{n>0}$ is polynomial-time samplable if there is a randomized algorithm A that, on input number n, outputs a string in $\{0,1\}^{d(n)}$ and:*

- *There is a polynomial p such that, on input n, A runs in time at most $p(n)$, regardless of its internal coin tosses;*

- *For every n and for every $x \in \{0,1\}^{d(n)}$, $\boldsymbol{Pr}[A(n) = x] = D_n(x)$.*

We will also be interested in the ensembles whose cumulative distribution function is efficiently computable. If D is a distribution, let F_D be the cumulative distribution function, i.e., $F_D(x) = \sum_{y \preceq x} D(y)$.

Definition 2.3 *We say that an ensemble $\mathcal{D}$=$\{D_n\}_{n>0}$ is polynomial-time computable if there is an algorithm that, given an integer n and a string x, runs in time polynomial in n and computes $F_{D_n}(x)$.*

Observe that the function $D_n(x)$ is computable in time polynomial in n when $\{D_n\}_{n>0}$ is a computable ensemble since

$$D_n(x) = F_{D_n}(x) - F_{D_n}(x-1)$$

where $x - 1$ denotes the predecessor of x in the above defined $\prec$ order.

We let **PSamp** denote the class of polynomial-time samplable ensembles, and **PComp** denote the class of polynomial-time computable ensembles. Let U_n be the uniform distribution over $\{0,1\}^n$. We call the ensemble $\mathcal{U} = \{U_n\}_{n>0}$ the *uniform ensemble*, which is an example of a polynomial-time computable and samplable ensemble. The polynomial d, in definition 2.1, will naturally be the identity polynomial and the function f can be the padding with zero.

Definition 2.4 *A distributional complexity class is a collection of distributional decision problems. For a class of languages $\mathbf{C}$ and a class of ensembles $\mathbf{D}$, we use $(\mathbf{C}, \mathbf{D})$ to denote the distributional complexity class consisting of all problems (L, $\mathcal{D}$) where $L \in \mathbf{C}$ and $\mathcal{D} \in \mathbf{D}$.*

2.2 Notions of Average-Case Tractability

We will divide our discussion in three parts: errorless algorithms, heuristic algorithms and randomized heuristic algorithms.

2.2.1 Errorless Algorithms

An *errorless algorithm* is a deterministic algorithm that never makes mistakes. The simplest and most natural definition of efficient on average algorithm would be to say that it runs in expected polynomial-time. So considering a distributional decision problem (L, $\mathcal{D}$), an algorithm A would be efficient on average if there was a polynomial p such that

$$\mathbf{E}_{x\sim D_n}\left[t_A(x;n)\right] = \textstyle\sum_{x\in\{0,1\}^{d(n)}} D_n(x)t_A(x;n) \leq p(n)$$

for every n, where $t_A(x;n)$ is the running time of A on input x and parameter n. The notation $x \sim D_n$ means that the expected value refers to the sampling of the input x according to the distribution D_n.

However, such a definition is too restrictive, i.e., there are problems intuitively efficient on average that wouldn't be captured by this definition. For instance, assume that the inputs come from the uniform distribution over $\{0,1\}^n$, U_n. Let A be an algorithm that runs in time $\mathcal{O}(n^2)$ on all inputs of length n, except on a set of $2^{n/2}$ inputs on which it takes time $\mathcal{O}(2^{n/2})$. Let B be an algorithm that is quadratically slower than A, that is, for every $x \in \{0,1\}^n$, $t_B(x;n) = (t_A(x;n))^2$. So B is an algorithm that runs in time $\mathcal{O}(n^4)$ on all inputs of length n, except on a set of $2^{n/2}$ inputs on which it takes time $\mathcal{O}(2^n)$ and we have:

$$\begin{aligned}\mathbf{E}_{x\sim U_n}\left[t_A(x;n)\right] &= \textstyle\sum_{x\in\{0,1\}^n} U_n(x)t_A(x;n)\\ &= 2^{-n}\textstyle\sum_{x\in\{0,1\}^n} t_A(x;n)\\ &= 2^{-n}\left(\mathcal{O}(n^2)(2^n-2^{n/2}) + \mathcal{O}(2^{n/2})2^{n/2}\right)\\ &= \mathcal{O}(n^2),\end{aligned}$$

$$\begin{aligned}\mathbf{E}_{x\sim U_n}\left[t_B(x;n)\right] &= \textstyle\sum_{x\in\{0,1\}^n} U_n(x)t_B(x;n)\\ &= 2^{-n}\textstyle\sum_{x\in\{0,1\}^n} t_B(x;n)\\ &= 2^{-n}\left(\mathcal{O}(n^4)(2^n-2^{n/2}) + \mathcal{O}(2^{n})2^{n/2}\right)\\ &= \mathcal{O}(2^{n/2}).\end{aligned}$$

Hence, A would be efficient on average but B wouldn't, despite our intuition. In general, we want that if A is efficient on average and B is polynomially slower than A, then B is still efficient on average.

This example also points out a very important and intuitive notion: the algorithm can run in exponential time for some inputs and still be efficient on average if those inputs represent a very small fraction of all possible inputs. In fact, we want that the fraction of inputs requiring larger and larger running time to be smaller and smaller. Hence, there is a polynomial trade-off between the running time and fraction of inputs, which leads to the following definition.

Definition 2.5 ***(Trade-off Definition)*** *An algorithm A has average polynomial running time with respect to the ensemble $\mathcal{D}$ if there is an $\epsilon > 0$ and a polynomial p such that for every n and every t:*

$$\boldsymbol{Pr}_{x\sim D_n}\left[t_A(x;n) \geq t\right] \leq \tfrac{p(n)}{t^\epsilon}.$$

In his turn, Levin gave the following equivalent definition.

Definition 2.6 ***(Levin's Definition)*** *An algorithm A has average polynomial running time with respect to the ensemble $\mathcal{D}$ is there is an $\epsilon > 0$ such that*

$$\boldsymbol{E}_{x\sim D_n}\left[t_A(x;n)^\epsilon\right] = \mathcal{O}(n).$$

Naturally, $\mathcal{O}$(n) can be replaced by $\mathcal{O}\left(p\left(n\right)\right)$ where p is an arbitrary polynomial. We now prove that the definitions are equivalent.

Proposition 2.7 *An algorithm A has average polynomial running time with respect to the ensemble $\mathcal{D}$ according do Definition 2.5 if and only if it does according to Definition 2.6.*

Proof: We first prove that if A satisfies Definition 2.5 then it satisfies Definition 2.6. Suppose that the running time t_A of A satisfies

$$\mathbf{Pr}_{x\sim D_n}\left[t_A(x;n) \geq t\right] \leq \tfrac{n^c}{t^\epsilon}$$

for some positive constants c and ϵ and for every sufficiently large n . Define $\delta = \epsilon/(c{+}2)$. Using the fact that $t_A(x;n)$ is a non-negative random variable, we have

$$\begin{aligned}
\mathbf{E}_{x\sim D_n}\left[t_A(x;n)^\delta\right] &= \textstyle\sum_t \mathbf{Pr}_{x\sim D_n}\left[t_A(x;n)^\delta \geq t\right] \\
&\leq n + \textstyle\sum_{t\geq n} \mathbf{Pr}_{x\sim D_n}\left[t_A(x;n) \geq t^{1/\delta}\right] \\
&\leq n + \textstyle\sum_{t\geq n} n^c t^{-\epsilon/\delta} \\
&= n + \textstyle\sum_{t\geq n} n^c t^{-(c+2)} \\
&\leq n + \textstyle\sum_{t\geq n} t^{-2} \\
&= n + \mathcal{O}(1) \\
&= \mathcal{O}(n).
\end{aligned}$$

For the other implication, suppose that

$$\mathbf{E}_{x\sim D_n}\left[t_A(x;n)^\epsilon\right] = \mathcal{O}(n).$$

Using the Markov's inequality, we have

$$\mathbf{Pr}_{x\sim D_n}\left[t_A(x;n) \geq t\right] = \mathbf{Pr}_{x\sim D_n}\left[t_A(x;n)^\epsilon \geq t^\epsilon\right] \leq \tfrac{\mathbf{E}_{x\sim D_n}[t_A(x;n)^\epsilon]}{t^\epsilon} = \mathcal{O}(nt^{-\epsilon}).$$

□

We now see that these definitions are coherent with what we intended, that is, if A is efficient on average and B is polynomially slower than A, then B is still efficient on average, by proving that the algorithms A and B defined above have both average polynomial running time according to definition 2.6. Let $\epsilon_A = 1/2$ and $\epsilon_B = 1/4$. So:

$$\begin{aligned}
\mathbf{E}_{x\sim U_n}\left[(t_A(x;n))^{\epsilon_A}\right] &= \textstyle\sum_{x\in\{0,1\}^n} U_n(x)(t_A(x;n))^{\epsilon_A} \\
&= 2^{-n} \textstyle\sum_{x\in\{0,1\}^n} (t_A(x;n))^{\epsilon_A} \\
&= 2^{-n}\left(\mathcal{O}(n)(2^n - 2^{n/2}) + \mathcal{O}(2^{n/4})2^{n/2}\right) \\
&= \mathcal{O}(n),
\end{aligned}$$

$$\begin{aligned}
\mathbf{E}_{x\sim U_n}\left[(t_B(x;n))^{\epsilon_B}\right] &= \textstyle\sum_{x\in\{0,1\}^n} U_n(x)(t_B(x;n))^{\epsilon_B} \\
&= 2^{-n} \textstyle\sum_{x\in\{0,1\}^n} (t_B(x;n))^{\epsilon_B} \\
&= 2^{-n}\left(\mathcal{O}(n)(2^n - 2^{n/2}) + \mathcal{O}(2^{n/4})2^{n/2}\right) \\
&= \mathcal{O}(n).
\end{aligned}$$

Suppose now we have an algorithm A that has average polynomial running time according to the above definitions. In practice, we will only run A for a polynomial-time, but there may be some inputs on which A takes more than polynomial-time. For these inputs, we can say that A failed because we have to stop the computation without an answer. The following definition captures this notion.

Definition 2.8 *We say that an algorithm A is an errorless heuristic scheme for $(L, \mathcal{D})$ if there is a polynomial p such that:*

- *For every n, $\delta > 0$ and every $x \in \{0,1\}^{d(n)}$, $A\,(x; n, \delta)$ outputs either $L\,(x)$ or the special failure symbol $\perp$;*
- *For every n, $\delta > 0$ and every $x \in \{0,1\}^{d(n)}$, $A\,(x; n, \delta)$ runs in time at most $\frac{p(n)}{\delta}$;*
- *For every n and $\delta > 0$, $\boldsymbol{Pr}_{x \sim D_n}\left[A\,(x; n, \delta) = \perp\right] \leq \delta$.*

The parameter δ allows us to control the amount of time we are willing to sacrifice in order to obtain an answer. We now prove that errorless heuristic schemes are yet another way to capture the notion of average-case tractability of Definitions 2.5 and 2.6.

Proposition 2.9 *A distributional decision problem $(L, \mathcal{D})$ admits an errorless heuristic scheme if and only if it admits an algorithm with average polynomial-time according to Definitions 2.5 and 2.6.*

Proof: Suppose that A is an algorithm that runs in average polynomial-time according to Definition 2.5. Hence, there is an $\epsilon > 0$ and a polynomial p such that for every n and every t:

$$\mathbf{Pr}_{x \sim D_n}\left[t_A(x; n) \geq t\right] \leq \frac{p(n)}{t^\epsilon}$$

Let B be the algorithm that on input x and parameters n, δ simulates $A(x; n)$ for $(p(n)/\delta)^{1/\epsilon}$ steps. If the simulation halts within the required number of steps, then $B\,(x; n, \delta)$ $= A(x; n)$. Otherwise, $B\,(x; n, \delta) = \perp$. By definition of B, the first and second conditions of errorless heuristic scheme are trivial satisfied. For the third, we have:

$$\mathbf{Pr}_{x \sim D_n}\left[B\,(x; n, \delta) = \perp\right] = \mathbf{Pr}_{x \sim D_n}\left[t_A(x; n) \geq (p(n)/\delta)^{1/\epsilon}\right] \leq \frac{p(n)}{((p(n)/\delta)^{1/\epsilon})^\epsilon} = \delta$$

which proves that B is an errorless heuristic scheme.

Suppose now that A is an errorless heuristic scheme for $(L, \mathcal{D})$. Define the algorithm B as follows: on input x and parameter n, simulate $A(x;n,1/2)$, if $A(x;n,1/2) \neq \perp$ then return the output of $A(x;n,1/2)$, otherwise simulate $A(x;n,1/4)$, and so on, simulating $A(x;n,1/8)$,..., $A(x;n,2^{-k})$,... until we reach a value of δ such that $A(x;n,\delta) \neq \perp$.

We first prove that B always halts for every input x. Suppose that

$$\exists_{y\in\{0,1\}^{d(n)}} \forall_{\delta>0}\ A(y;n,\delta) = \perp.$$

Let y have the above property and choose δ such that $\delta < D_n(y)$. So

$$\mathbf{Pr}_{x\sim D_n}\left[A\left(x;n,\delta\right) = \perp\right] \geq D_n(y),$$

but as A is an errorless heuristic scheme we have

$$\mathbf{Pr}_{x\sim D_n}\left[A\left(x;n,\delta\right) = \perp\right] \leq \delta.$$

which is a contradiction. Hence,

$$\forall_{y\in\{0,1\}^{d(n)}} \exists_{\delta>0}\ A(y;n,\delta) \neq \perp.$$

Finally we prove that B has average polynomial time according to Definition 2.5. Since A is an errorless heuristic scheme there is a polynomial p such that for every n, $\delta > 0$ and every $x \in \{0,1\}^{d(n)}$,

$$A\left(x;n,\delta\right) \text{ runs in time at most } \tfrac{p(n)}{\delta}.$$

Let $\epsilon = 1$ and r the polynomial $4p$. We will prove that for every n and every t

$$\mathbf{Pr}_{x\sim D_n}\left[t_A(x;n) \geq t\right] \leq \tfrac{r(n)}{t^\epsilon}.$$

Let $\delta_k = 2^{-k}$. Fix t and chose k such that

$$k = \min\left\{m : \sum_{i=1}^{m+1} \frac{p\left(n\right)}{\delta_k} \geq t\right\}$$

Observe that

$$\sum_{i=1}^{k+1} \frac{p\left(n\right)}{\delta_k} \geq t \Leftrightarrow \left(2^{k+2}-1\right) p\left(n\right) \geq t \Rightarrow 2^{k+2} p\left(n\right) \geq t \Leftrightarrow \frac{4p\left(n\right)}{t} \geq \frac{1}{2^k} = \delta_k.$$

Finally we have

$$\begin{aligned}
\mathbf{Pr}_{x\sim D_n}\left[t_B(x;n) \geq t\right] &\leq \mathbf{Pr}_{x\sim D_n}\left[t_B(x;n) \geq \sum_{i=1}^{k} \frac{p\left(n\right)}{\delta_k}\right] \\
&\leq \mathbf{Pr}_{x\sim D_n}\left[A(x;n,\delta_1) = \perp, \ldots, A(x;n,\delta_k) = \perp\right] \\
&= \mathbf{Pr}_{x\sim D_n}\left[A(x;n,\delta_k) = \perp\right] \\
&\leq \delta_k \\
&\leq \tfrac{r(n)}{t^\epsilon}.
\end{aligned}$$

□

With these three equivalent definitions of average-case efficiency , we can now define the complexity class of distributional decision problems that are easy on average, i.e., distributional decision problems that have an efficient on average algorithm.

Definition 2.10 *We define **AvgP** to be the class of distributional decision problems that admit an errorless heuristic scheme.*

2.2.2 Heuristic Algorithms

We now briefly focus our attention on algorithms that return incorrect answers on a small fraction of inputs.

Definition 2.11 *Let (L,D) be a distributional decision problem and $\delta > 0$. We say that an algorithm A is a heuristic algorithm for (L,D) with error probability at most δ if for all $n > 0$,*

$$\boldsymbol{Pr}_{x\sim D_n}\left[A\left(x;n,\delta\right) \neq L(x)\right] \leq \delta$$

As we did for the errorless algorithms, we can define an heuristic scheme for heuristic algorithms in the natural way.

Definition 2.12 *We say that an algorithm A is a heuristic scheme for (L,D) if there is a polynomial p such that*

- *For every n, $\delta > 0$ and every $x \in \{0,1\}^{d(n)}$, $A(x; n, \delta)$ runs in time at most $\frac{p(n)}{\delta}$;*
- *For $\delta > 0$, $A(\cdot;\cdot,\delta)$ is a heuristic algorithm for (L,D) with error probability at most δ.*

$A(\cdot;\cdot,\delta)$ is the algorithm that on input x and parameter n, simulates $A(x; n, \delta)$.

Definition 2.13 *We define **HeurP** to be the class of distributional decision problems that admit an heuristic scheme.*

Observe that if we replace the failure symbol $\perp$ by an arbitrary or random output, we turn an errorless algorithm into a heuristic algorithm, so **AvgP** $\subseteq$ **HeurP**.

2.2.3 Randomized Heuristic Algorithms

We now discuss randomized heuristic algorithms that will be important in the next section as we will deal with randomized reductions.

When dealing with randomized heuristic algorithms, there are two reasons why our algorithm can fail to produce the right answer: the heuristic can simply fail due to the input itself or the heuristic is actually good for that input but makes a bad internal coin toss. A good example of an algorithm that makes errors of the first type is the Fermat Primality Test. This algorithm on input n, answers that n is prime if and only if for k randomly chosen values of a such that $1 \leq a < n$,

$$a^{n-1} \equiv 1 \pmod{n}.$$

By Fermat's little theorem, for n prime, A produces the correct answer. However there are certain values of n, known as Carmichael numbers, for which all values of a coprime with n verify $a^{n-1} \equiv 1 \pmod{n}$. For those values of n, the algorithm simply fails.

In these section, "errorless" refers to the choice of input and not to the internal coin tosses of the algorithm.

Definition 2.14 *Let (L,D) be a distributional decision problem and $\delta > 0$. We say that a randomized polynomial-time algorithm A is a randomized errorless heuristic algorithm of failure probability at most δ if, for every n > 0 and every $x \in \{0,1\}^{d(n)}$,*

$$\boldsymbol{Pr}[A(x;n) \notin \{L(x), \perp\}] \leq 1/4$$

where the probability is taken over the internal coin tosses of A, and

$$\boldsymbol{Pr}_{x \sim D_n}[\boldsymbol{Pr}[A(x;n) = \perp] \geq 1/4] \leq \delta$$

where the inner probability is over the internal coin tosses of A.

One undesirable property of our algorithm is the existence of instances $x \in \{0,1\}^{d(n)}$ such that if we would run our algorithm on input $(x;n)$ k times, for some large k, we would have more than k/4 returns of the failure symbol $\perp$, since this is a sign that the algorithm doesn't know the correct answer. However, due to the second condition, we are able to guarantee with high probability over the randomness of the algorithm that this won't happen for more than a δ-fraction of instances of x. Furthermore, if less than a quarter of the runs return $\perp$, we obtain the correct answer for x with high probability over the randomness of the algorithm, because the first condition guarantees that most of the runs that do not output $\perp$ will output the correct answer.

We can now define a randomized errorless heuristic scheme and the respective complexity class of distributional decision problems.

Definition 2.15 *We say that an algorithm A is a randomized errorless heuristic scheme for (L,D) if there is a polynomial p such that*

- *For every n, $\delta > 0$ and every $x \in \{0,1\}^{d(n)}$, $A(x;n,\delta)$ runs in time at most $\frac{p(n)}{\delta}$;*
- *For $\delta > 0$, $A(\cdot;\cdot,\delta)$ is a randomized errorless heuristic algorithm for (L,D) of failure probability at most δ.*

Definition 2.16 *We define* ***AvgBPP*** *to be the class of distributional decision problems that admit a randomized errorless heuristic scheme.*

In these definitions we assumed that our algorithm only makes mistakes due to bad internal choices. If we consider heuristic algorithm, i.e, if we allow errors due to the input, we don't have to distinguish the errors and the definitions are simpler.

Definition 2.17 *Let (L,D) be a distributional decision problem and $\delta > 0$. We say that a randomized polynomial-time algorithm A is a randomized heuristic algorithm of failure probability at most δ if, for every $n > 0$,*

$$\boldsymbol{Pr}_{x \sim D_n} \left[\boldsymbol{Pr} [A(x; n) \neq L(x)] \geq 1/4 \right] \leq \delta$$

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

Once again, we define the associated scheme and the respective complexity class of distributional decision problems.

Definition 2.18 *We say that an algorithm A is a randomized heuristic scheme for (L,D) if there is a polynomial p such that*

- *For every n, $\delta > 0$ and every $x \in \{0,1\}^{d(n)}$, $A(x; n, \delta)$ runs in time at most $\frac{p(n)}{\delta}$;*
- *For $\delta > 0$, $A(\cdot; \cdot, \delta)$ is a randomized heuristic algorithm for (L,D) of failure probability at most δ.*

Definition 2.19 *We define **HeurBPP** to be the class of distributional decision problems that admit a randomized heuristic scheme.*

We now make the same observation that we did for **AvgP** and **HeurP**. Since we can turn a randomized errorless heuristic algorithm into a randomized heuristic algorithm, we have that **AvgBPP** $\subseteq$ **HeurBPP**.

3 Decision versus Search and One-Way Functions

In this section we introduce the definitions of efficient on average search algorithms and present a result of Ben-David et al. on the equivalence of search and decision algorithms for problems in $(\mathbf{NP}, \mathbf{PComp})$, exploring its consequences in cryptography.

Since we will deal with search problems, we define distributional search problem.

Definition 3.1 *Let S be a binary relation and let $\mathcal{D}$ be the ensemble $\mathcal{D}=\{D_n\}_{n>0}$. For each n, D_n is a distribution over $\{0,1\}^{d(n)}$ and*

$$\forall_{x\in\{0,1\}^{d(n)}} \frac{D_{n+1}(f(x))}{\sum_{y\in\{0,1\}^{d(n)}} D_{n+1}(f(y))} = D_n(x)$$

where d is a polynomial and f is a function such that:

- *f is injective;*
- *f is invertible;*
- *for every n, if $x \in \{0,1\}^{d(n)}$ then $f(x) \in \{0,1\}^{d(n+1)}$.*

We call the pair (S, $\mathcal{D}$) a distributional search problem.

When S is a unique solution search problem, we call the pair (S, $\mathcal{D}$) a distributional unique-search problem.

3.1 Search Algorithms

As we did in chapter 2 for decision algorithms, we will discuss in this section errorless heuristic, randomized errorless heuristic and randomized heuristic search schemes.

By analogy with our discussion in chapter 2, we define average polynomial-time search algorithms in the natural way.

Definition 3.2 *For a distributional search problem (S, $\mathcal{D}$) where $S \in$ **NP-search**, we say that A is a deterministic average polynomial-time search algorithm for (S, $\mathcal{D}$) if*

- *for every n and every $x \in L(S) \cap \{0,1\}^{d(n)}$, $A(x;n)$ outputs an L(S)-witness for x;*

- *there is an $\epsilon > 0$ such that*

$$\boldsymbol{E}_{x\sim D_n}\left[t_A(x;n)^{\epsilon}\right] = \mathcal{O}(n).$$

As in the case of decision algorithms (remember Proposition 2.9), this definition is equivalent to the following.

Definition 3.3 *We say that an algorithm A is an errorless heuristic search scheme for (S, $\mathcal{D}$), where $S \in$ **NP-search**, if there is a polynomial p such that:*

- *For every n, $\delta > 0$ and every $x \in L(S) \cap \{0,1\}^{d(n)}$, $A(x;n,\delta)$ outputs either an $L(S)$-witness for x or the special failure symbol $\perp$;*
- *For every n, $\delta > 0$ and every $x \in L(S) \cap \{0,1\}^{d(n)}$, $A(x;n,\delta)$ runs in time at most $\frac{p(n)}{\delta}$;*
- *For every n and $\delta > 0$, $\boldsymbol{Pr}_{x\sim D_n}\left[A(x;n,\delta) = \perp\right] \leq \delta$.*

Observe that nothing is said when $x \notin L(S)$, hence, the output of the algorithm can be arbitrary. If we chose to output anything other than the special symbol $\perp$, this provides a certificate that $x \notin L(S)$ as it can be efficiently checked that the output of the algorithm is not a witness for x.

Once again, the term "errorless" in randomized algorithms refers to the choice of the input and not to the internal coin tosses of the algorithm.

Definition 3.4 *Let (S,D) be a distributional decision problem where $S \in$ **NP-search**. We say that A is a randomized errorless heuristic search scheme for (S,D) if there is a polynomial p such that*

- *For every n, $\delta > 0$ and every $x \in \{0,1\}^{d(n)}$, $A(x;n,\delta)$ runs in time at most $\frac{p(n)}{\delta}$ and outputs a string $w \in \{0,1\}^*$ or the special symbol $\perp$;*
- *For every n, $\delta > 0$ and every $x \in L(S) \cap \{0,1\}^{d(n)}$,*

$$\boldsymbol{Pr}\left[A(x;n,\delta)\, is\ not\ a\ witness\ for\ x\ and\ A(x;n,\delta) \neq \perp\right] \leq 1/4$$

where the probability is taken over the internal coin tosses of A;

- *For every n and $\delta > 0$,*

$$\boldsymbol{Pr}_{x \sim D_n}\left[\boldsymbol{Pr}[A(x; n, \delta) = \perp] \geq 1/4\right] \leq \delta$$

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

This definition can be simplified if we allow errors due to the input.

Definition 3.5 *Let (S,D) be a distributional decision problem where $S \in$ **NP-search**. We say that A is a randomized heuristic search scheme for (S,D) if there is a polynomial p such that*

- *For every n, $\delta > 0$ and every $x \in \{0,1\}^{d(n)}$, $A(x; n, \delta)$ runs in time at most $\frac{p(n)}{\delta}$ and outputs a string $w \in \{0,1\}^*$ or the special symbol $\perp$;*

- *For every n and $\delta > 0$,*

$$\boldsymbol{Pr}_{x \sim D_n}\left[\boldsymbol{Pr}[A(x; n, \delta) = \perp] \geq 1/4\right] \leq \delta$$

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

3.2 Reductions

We now present two definitions of reduction of one distributional problem to another. Intuitively, the reduction should be efficiently computable, yield a valid result and "preserve" the probability distribution. The last requirement is needed since if we have an efficient on average algorithm for the second problem, we want that the first problem also has one. Hence, we have to ensure that the reduction does not map very likely instances of the first problem to rare instances of the second problem.

Definition 3.6 *Let $(S_1, \mathcal{D}^1)$ and $(S_2, \mathcal{D}^2)$ be distributional search problems where $S_1, S_2 \in$ **NP-search**. We say that a randomized algorithm A randomly reduces S_1 to S_2 and write $S_1 \leq_{Avg} S_2$ if the following three conditions hold:*

- ***Efficiency***: *there is an $\epsilon > 0$ such that*

$$\boldsymbol{E}_{x\sim D_n^1}\left[t_A(x;n)^\epsilon\right] = \mathcal{O}(n).$$

- ***Validity***: *for every n and every $x \in \{0,1\}^{d^1(n)}$*

$$\boldsymbol{Pr}\left[A^{S_2}(x;n) \equiv\ S_1(x)\right] \geq \tfrac{2}{3}$$

  where $A^{S_2}(x;n)$ denotes the output of the oracle algorithm A on input x and access to an oracle for S_2 and the probability is taken over the internal coin tosses of A. The notation $\equiv$ means that both outputs are from the same type, that is, both are $\perp$ or witnesses for x.

- ***Domination***: *there is a polynomial p and m such that for every n and $y \in \{0,1\}^{m(n)}$*

$$D^2_{m(n)}(y) \geq \tfrac{1}{p(n)} \textstyle\sum_{x\in\{0,1\}^{d^1(n)}} Ask_A(x,y) D^1_n(x)$$

  where $Ask_A(x,y)$ is the probability (taken over the internal coin tosses of A) that A asks a query y on input x.

Although we do not prove it here we assume that if $S_1 \leq_{Avg} S_2$ and S_2 has a randomized (errorless) heuristic search scheme then S_1 also has one.

Definition 3.7 *Let $(S, \mathcal{D}^S)$ be a distributional search problem and $(L, \mathcal{D}^L)$ a distributional decision problems where $S \in$ **NP-search** and $L \in$ **NP**. We say that a randomized algorithm A randomly reduces S to L and write $S \leq_{Avg} L$ if the following three conditions hold:*

- ***Efficiency***: *there is an $\epsilon > 0$ such that*

$$\boldsymbol{E}_{x\sim D_n^S}\left[t_A(x;n)^\epsilon\right] = \mathcal{O}(n).$$

- ***Validity***: *for every n and every $x \in \{0,1\}^{d^S(n)}$*

$$\boldsymbol{Pr}\left[A^{L}(x;n) \equiv\ S(x)\right] \geq \tfrac{2}{3}$$

where $A^L(x;n)$ denotes the output of the oracle algorithm A on input x and access to an oracle for L and the probability is taken over the internal coin tosses of A. Now, the notation $\equiv$ means that $A^L(x;n)$ is a witness for x and $L(x) = 1$ or $A^L(x;n) = \perp$ and $L(x) = 0$.

- ***Domination**: there is a polynomial p and m such that for every n and $y \in \{0,1\}^{m(n)}$*

$$D^L_{m(n)}(y) \geq \tfrac{1}{p(n)} \textstyle\sum_{x \in \{0,1\}^{d^S(n)}} Ask_A(x,y) D^S_n(x)$$

where $Ask_A(x,y)$ is the probability (taken over the internal coin tosses of A) that A asks a query y on input x.

Once again, we assume that if $S \leq_{Avg} L$ and L admits a randomized (errorless) heuristic scheme then S admits a randomized (errorless) heuristic search scheme.

3.3 Ben-David et al. theorem

Before we present the proof of the theorem of Ben-David et al., we observe that the common decision to search reduction fails in the average case since it may not have the domination property. Remember that the answer of the search algorithm was constructed based on queries of the form "is p a prefix of a solution to x?", where x is the input. It is possible to define a distribution on the decision problem such that the reduction maps an instance of the search problem to a much more rare instance of the decision problem. Hence, the domination property does not hold.

Theorem 3.8 *Let $(S_1, \mathcal{D}^1) \in$ (**NP-search**, **PComp**). Then there distributional unique-search problem $(S_2, \mathcal{D}^2) \in$ (**NP-search**, **PComp**) such that $S_1 \leq_{Avg} S_2$.*

Proof: For the sake of simplicity assume $(x,y) \in S_1$ implies $|x| = |y|$. Let $n = |x|$ and $H_{n,k}$ be a set of universal hash functions. We define the unique solution search problem $S_2 \subseteq \{0,1\}^* \times \{0,1\}^*$ as follows: $(x', y) \in S_2$ for $x' = (x,k,h,\alpha)$ if and only if:

- $(x,y) \in S_1$
- $h \in H_{n,k}$

- $h(y) = \alpha$.

We represent the tuples (x, k, h, α) as strings such that for every n all the elements in

$$\left\{(x, k, h, \alpha) : x \in \{0,1\}^{d^1(n)} \wedge k \in \{1, \ldots, n\} \wedge h \in H_{n,k} \wedge \alpha \in \{0,1\}^k\right\}$$

have the same size, which can be done with the padding of 0's.
We let $n' = |x'|$ and define $\mathcal{D}^2 = \left\{D_n^2\right\}_{n>0}$ such that

$$D_{n'}^2(x') = \begin{cases} D_n^1(x) n^{-1} \left|H_{n,k}\right|^{-1} 2^{-k} & \text{if } k \in \{1, \ldots, n\}, h \in H_{n,k} \text{ and } \alpha \in \{0,1\}^k, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $(S_2, \mathcal{D}^2) \in (\textbf{NP-search}, \textbf{PComp})$ (recall that the hashing functions are computable in polynomial-time). To prove that $S_1 \leq_{Avg} S_2$, we define the following randomized algorithm A: on input $x \in \{0,1\}^n$, the algorithm, for every value $k \in \{1, \ldots, n\}$, selects at random with uniform probability $h \in H_{n,k}$ and $\alpha \in \{0,1\}^k$ and makes the oracle query (x, k, h, α). This is repeated at most r times. If one of the oracle queries outputs a string $y \in \{0,1\}^n$ such that $(x', y) \in S_2$ for $x' = (x, k, h, \alpha)$, $A^{S_2}(x; n) = y$. Otherwise, $A^{S_2}(x; n) = \perp$.

Observe that we can only guaranty that the oracle outputs the correct answer on input $x' = (x, k, h, \alpha)$ if there is a unique y such that $(x', y) \in S_2$ since S_2 is a unique solution search problem. That's why we need to verify the output of the oracle.

A is efficient since $S_2 \in \textbf{NP-search}$ and the number of oracle queries is polynomial in n. To prove the validity property we fix n and x and define $m = |\{y : (x, y) \in S_1\}|$. Clearly, if $m = 0$ then

$$\mathbf{Pr}\left[A^{S_2}(x; n) \equiv S_1(x)\right] = 1 \geq \tfrac{2}{3}.$$

Now, consider the case $m = 1$. For a randomly selected $h \in H_{n,1}$ and $\alpha \in \{0,1\}$, the probability that there is a unique y such that $(x, y) \in S_1$ and $h(y) = \alpha$ is $\frac{1}{2}$. Hence for r randomly chosen pairs (h, α) such that $h \in H_{n,1}$ and $\alpha \in \{0,1\}$, the probability that there is not a unique y is $(1 - 1/2)^r$. Our algorithm will succeed, that is $A^{S_2}(x; n) \equiv S_1(x)$, if in particularly for $k = 1$ the oracle outputs y satisfying the above conditions, which we can only guaranty if y is unique. Hence, choosing $r \geq 2$ we have

$$\begin{aligned}\mathbf{Pr}\left[A^{S_2}(x;n) \equiv\ S_1(x)\right] &\geq 1-(1-1/2)^r\\ &\geq \tfrac{2}{3}.\end{aligned}$$

For $m \geq 2$, the proof follows the same idea but for $k = \lceil \log_2(m) \rceil$. Let $y_1, \ldots, y_m$ be the m different witnesses for x. By the properties of universal hashing (i.e., pairwise independence of the image of pairs of points) it follows that the probability that there is a unique y such that $h \in H_{n,k}$ and $\alpha \in \{0,1\}^k$ is

$$\begin{aligned}\textstyle\sum_{i=1}^{m} \mathbf{Pr}\left[y_i \in h^{-1}(\alpha) \text{ and } y_j \notin h^{-1}(\alpha) \text{ for j } \in \{1,\ldots,m\}\setminus\{i\}\right] &= \textstyle\sum_{i=1}^{m} \frac{1}{2^k}\left(\frac{2^k-1}{2^k}\right)^{(m-1)}\\ &= \tfrac{m}{2^k}\left(\tfrac{2^k-1}{2^k}\right)^{(m-1)}\end{aligned}$$

where the probability is taken over the choice of $h \in H_{n,k}$ and $\alpha \in \{0,1\}^k$ with uniform probability.

Due to the choice of k, we have that $m \leq 2^k < 2m$ and that the above probability is lower-bounded by

$$\tfrac{m}{2m}\left(\tfrac{2m-1}{2m}\right)^{(m-1)}$$

which is a decreasing function in m with

$$\lim_{m\to\infty} \tfrac{m}{2m}\left(\tfrac{2m-1}{2m}\right)^{(m-1)} = \tfrac{1}{2\sqrt{e}} \geq \tfrac{3}{10}.$$

Hence the probability that there is a unique y such that $h \in H_{n,k}$ and $\alpha \in \{0,1\}^k$ is $\geq \frac{3}{10}$. Now, just as above, but choosing $r \geq 4$ we have

$$\begin{aligned}\mathbf{Pr}\left[A^{S_2}(x;n) \equiv\ S_1(x)\right] &\geq 1-(1-\tfrac{3}{10})^r\\ &\geq \tfrac{2}{3}.\end{aligned}$$

Hence, if we choose $r = 4$ the validity property is satisfied.

Finally, to see that the domination property also holds observe that for every $x \in \{0,1\}^n$, $Ask_A(x,y)$

$$= \begin{cases} 4\left|H_{n,k}\right|^{-1} 2^{-k} & \text{if } y = (x,k,h,\alpha) \text{ with } k \in \{1,\ldots,n\}, h \in H_{n,k} \text{ and } \alpha \in \{0,1\}^k, \\ 0 & \text{otherwise.}\end{cases}$$

So, if we choose $p(n) = 4n$ and m such that $m(n) = n'$ and let $y = (x, k, h, \alpha)$,

$$\begin{aligned} D^2_{m(n)}(y) &= D^1_n(x) n^{-1} \left|H_{n,k}\right|^{-1} 2^{-k} \\ &= \tfrac{1}{4n} D^1_n(x) 4 \left|H_{n,k}\right|^{-1} 2^{-k} \\ &= \tfrac{1}{p(n)} \textstyle\sum_{x \in \{0,1\}^n} Ask_A(x,y) D^1_n(x). \end{aligned}$$

□

Theorem 3.9 *Let $(S, \mathcal{D}^S) \in$ (**NP-search**, **PComp**) be a distributional unique-search problem. Then there is a distributional decision problem $(L, \mathcal{D}^L) \in (\mathbf{NP}, \boldsymbol{PComp})$ such that $S \leq_{Avg} L$.*

Proof: Define for all n and i such that $1 \leq i \leq n$, the strings e^n_i where $e^n_i \in \{0,1\}^n$ and the i^{th} bit of e^n_i is 1 and the others 0. Once again, we assume that $(x, y) \in S_1$ implies $|x| = |y|$ and let $n = |x|$.

We define $L \subseteq \{0,1\}^*$ as follows: $x' \in L$ for $x' = (x, e^n_i)$ if and only if there is y such that $(x, y) \in S$, $1 \leq i \leq n$ and the i^{th} bit of y is 1.

Assuming that $n' = |x'|$, we define $\mathcal{D}^S = \left\{D^S_n\right\}_{n>0}$ where

$$D^S_{n'}(x') = \begin{cases} D^L_n(x) \frac{1}{n} & \text{if } x' = (x, e^n_i) \text{ and } 1 \leq i \leq n, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $(L, \mathcal{D}^L) \in (\mathbf{NP}, \mathbf{PComp})$.

To prove the reduction, consider the algorithm A that on input x, makes the oracle queries $(x, e^n_1), \ldots, (x, e^n_n)$ and constructs $y \in \{0,1\}^n$ such that i^{th} bit of y is $L(x, e^n_i)$. Finally, if $(x, y) \in S$, $A^L(x; n) = y$ and $A^L(x; n) = \perp$ otherwise.

Clearly the reduction is efficient and valid since S is a unique solution search problem and

$$\mathbf{Pr}\left[A^L(x; n) \equiv S(x)\right] = 1 \geq \tfrac{2}{3}.$$

To see that the domination property also holds define $p(n) = n$ and $m(n) = 2n$. Observe that for every $x \in \{0,1\}^n$

$$Ask_A(x,y) = \begin{cases} 1 & \text{if } y = (x, e_i^n) \text{ with } 1 \le i \le n, \\ 0 & \text{otherwise.} \end{cases}$$

Hence if we let $y = (x, e_i^n)$

$$D^S_{m(n)}(y) = D^L_n(x)\tfrac{1}{n} = D^L_n(x)\tfrac{1}{p(n)} = \tfrac{1}{p(n)} \textstyle\sum_{x \in \{0,1\}^n} Ask_A(x,y) D^L_n(x)$$

□

As a result of theorems 3.8 and 3.9, we proved the following theorem of Ben-David et al. concerning the question of decision versus search algorithms.

Theorem 3.10 ***Ben-David et al.*** *If* $(\mathbf{NP}, \boldsymbol{PComp}) \subseteq \boldsymbol{AvgBPP}$ *then every problem in* $(\mathbf{NP}, \boldsymbol{PComp})$ *has a randomized errorless heuristic search scheme*

This theorem allows us to reach the following conclusion: if all languages in **NP** have good on average decision algorithms, they also have good on average search algorithms. This theorem also has some meaningful consequences in cryptography, particularly in the existence of one-way functions, which are a fundamental tool in cryptography. First, we recall what a one-way function is.

Definition 3.11 *A function* $f : \{0,1\}^* \mapsto \{0,1\}^*$ *is called one-way if the following conditions hold:*

- *there a polynomial-time algorithm* A *such that every* $x \in \{0,1\}^*$, A *output* $f(x)$.
- *for every randomized polynomial-time algorithm B, every polynomial p and all sufficiently large n's*

$$\boldsymbol{Pr}\left[B(x;n) \in f^{-1}(f(x))\right] < \tfrac{1}{p(n)}.$$

where the probability is taken over the internal coin tosses of B *and the sampling of* x *according to the uniform distribution.*

So, intuitively one-way functions are functions that are easy to compute, but hard to invert.

The security of most modern cryptosystems relies in the difficulty in inverting a candidate one-way function. By the theorem of Ben-David et al. if all problems in NP are easy on average, then every candidate one-way function can be inverted on a random output. Thus a cryptographic one-way function can exist only if there are problems in $(\mathbf{NP}, \mathbf{PComp})$ that are hard on average for decision algorithms.

4 Conclusion

The objective with this paper was to fully understand the basic definitions of average-case complexity theory and present them in a rigorous and simple way, as well explore the connections with cryptography. Chapter 2 clearly achieves the first objective. However, the second one was only partially achieved with chapter 3 since the initial goal was to go further. We finish this paper with some remarks on what could have been explored.

There are few results about the average-case tractability of problems. One of the main reasons relies in the difficulty to characterize the distributions of inputs that arise in practice and so the analysis of the algorithm's performance on "nice" distributions won't capture the "real life" average difficulty.

Even in the worst case there are no results that say that a specific problem is hard, i. e., it doesn't have a polynomial time solution. This situation is "solved" by assuming that specific problems are hard (e.g. finding the discrete logarithm or finding a Hamilton-cycle in a given graph which is a NP-complete problem). This assumption is accepted since it is based on the fact that these problems are well-know and have been attacked with no success by mathematicians for a long time. Unfortunately, in cryptography, one needs hard average-case problems and this assumption refers to worst-case problems. One practical solution is to use the factoring problem: find the prime factors of a large integer. If the integer is picked as the product of two large primes and some additional constrains, we assume that the problem becomes difficult on average and so applicable in cryptography.

Ajtai in [3] presents us another possibility: Ajtai describes two different average-case problems and their cryptography applications, which are at least as difficult as some well-known worst-case problems concerning lattices.

Finally, observe that the question "is $\mathbf{P} = \mathbf{NP}$?" is not the right one to . Even if we prove that $\mathbf{P} \neq \mathbf{NP}$, in a cryptography point of view we will still need to know if $(\mathbf{NP}, \mathbf{PComp}) \subseteq \mathbf{AvgP}$. Furthermore, if one could answer the question "is $(\mathbf{NP}, \mathbf{PComp}) \subseteq \mathbf{AvgP}$?" then the question "is $\mathbf{P} = \mathbf{NP}$?" becomes irrelevant.

References

[1] Andrej Bogdanov, Luca Trevisan. *Average-Case Complexity*, 2008.

[2] Oded Goldreich. *The Foundations of Cryptography - Volume 1*, 2001.

[3] Miklós Ajtai. *Worst-Case Complexity, Average-Case Complexity and Lattice Problems*, 1998.

[4] Russel Impagliazzo. *A Personal View of Average-Case Complexity*, 1995.