KOLMOGOROV COMPLEXITY FOR PROBABILISTIC COMPUTATIONS: TOWARDS RESOURCE-BOUNDED KOLMOGOROV COMPLEXITY FOR QUANTUM COMPUTING

RAJESH RACHHPAL SAINI

Computer Science Department

APPROVED:

Vladik Y. Kreinovich, Ph.D., Chair

Luc Longpré, Ph.D.

Scott Starks, Ph. D.

Charles H. Ambler, Ph.D.
Dean of the Graduate School

To my

FAMILY

who have always loved me

KOLMOGOROV COMPLEXITY FOR PROBABILISTIC COMPUTATIONS: TOWARDS RESOURCE-BOUNDED KOLMOGOROV COMPLEXITY FOR QUANTUM COMPUTING

by

RAJESH RACHHPAL SAINI, B.E.E.E.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Computer Science Department

THE UNIVERSITY OF TEXAS AT EL PASO

August 2002

(discard this page)

Abstract

The notion of Kolmogorov complexity is very useful in areas ranging from data compression to cryptography to foundations of physics. Some applications use resource bounded Kolmogorov complexity, a version that takes into consideration the running time of the corresponding program.

The notion of Kolmogorov complexity was originally proposed for traditional (deterministic) algorithms. Lately, it has been shown that in many important problems, probabilistic (in particular quantum) algorithms perform much better than the deterministic ones. It is therefore desirable to generalize the notion of Kolmogorov complexity to probabilistic algorithms. In this thesis, we proposed such a generalization. The resulting definition (partially) explains heuristic quantum versions of Kolmogorov complexity that have been proposed by P. Vitányi, S. Laplante, and others.

Table of Contents

Page

Abstract iv

Chapter

1 Introduction 1

1.1 Practical Problems that Led to the Notion of Kolmogorov Complexity 1

1.2 Kolmogorov Complexity as a Way of Solving Problems 3

1.3 Resource Bounded Kolmogorov Complexity 4

1.4 Quantum Computing 5

1.5 Earlier Work 8

1.5.1 Vitányi's definition 8

1.5.2 Definition by Sophie Laplante and Others 9

1.5.3 Gacs's Definition of Quantum Kolmogorov Complexity 10

1.6 Formulation of the Problem 14

2 Kolmogorov Complexity for Probabilistic Computation 15

2.1 Monte Carlo Algorithms 15

2.2 Motivation 16

2.3 Main Result 17

2.4 Comparison of Our Result With Known Definitions of Quantum Kolmogorov Complexity 23

3 Resource Bounded Kolmogorov Complexity for Probabilistic Computation 25
3.1 Motivation 25
3.2 Main Result 27
4 Conclusions 30
References 31

Chapter 1

Introduction

1.1 Practical Problems that Led to the Notion of Kolmogorov Complexity

In this section we are going to discuss some real life problems that led to the notion of Kolmogorov Complexity. The first problem is the problem of formalizing expert reasoning, in particular, experts' ability to distinguish between random and non-random sequences. Why is expert knowledge necessary? Computers were originally invented to process data and they are still used a lot to process data. In some cases, there is no need for expert knowledge. As an example, we can take a mobile robot. There are several ultrasonic sensors which measure the distance to the nearest obstacles. The computer then processes the sensor data, and computes the direction the robot should take to avoid the obstacles.

In many cases, however, someone needs to look into the result of processing and make a decision. For example, a medical doctor, no matter what the computer outputs, still looks at an X-ray, and makes a decision about a patient. A plane is usually controlled by an auto-pilot, but a human pilot monitors the situation and,

if necessary, makes a decision. For example, if a plane starts shaking in random directions, it is probably turbulence which usually can be handled by an auto-pilot. However, if the shaking appears non-random, tilted in a certain direction, it may indicate a more serious problem that requires a pilot's intervention. A pilot needs to personally check whether the motions of a plane are random or not, because there is no good algorithmic way of capturing this intuitive notion of randomness.

Another case when it is desirable to be able to check randomness is steganography, the technique of hiding information in images. A computer image is composed of information about pixels. In a black and white image, each pixel is represented by its intensity. A human eye can distinguish about 100 different intensity levels. To represent these levels it is sufficient to use one byte, i.e., 8 bits. There is no need to use all 8 bits, because with 8 bits we can represent 256 intensity levels from 0 to 255. However, since it is difficult to use pieces of bytes in modern computers, computers usually use the entire byte to store the intensity information. As a result, the computer-stored information about the intensity is too subtle for the human eye to see. For example, intensity levels 240 and 241 cannot be distinguished by a human eye. In general, if we replace the last bit of the intensity, the human eye cannot see the difference. So we can encode a message in last bits of bytes corresponding to different pixels. A good image contains $1000 \times 1000 = 10^6$ pixels. So, we have one million bits to store hidden information. This is not just a theoretical possibility, it was recently in the news that Al Qaeda and other terrorist groups are sending messages hidden in innocent-looking images. How can we detect the presence of a hidden image? In the original image, the last bits appear random. So, if the sequence of the last bit exhibit a certain non-random pattern, it is probable that there is a message hidden in this image. At present, it is very difficult to distinguish between random and non-random sequences, so experts do it manually.

Another case where we need to distinguish between random and non-random se-

quences is Search for Extra-Terrestrial Intelligence (SETI). The main idea behind SETI is that we look for signals from the sky; most of the signals are random; hypothetic SETI signals are not random. Hopefully, in the future, we would be able to detect a non-random sequence of extra terrestrial origin.

Another important problem is data compression. This problem is especially important for communications. The only reason why we can download movies over the web is that researchers have developed sophisticated compression algorithms that achieve two to three order of magnitude compression. Without this compression, downloading a movie will take days or even years. It is important to know whether we have reached the limit of lossless compression ability or not. If we have reached this limit, then the only way to further increase communication speed is to build a network with larger capacity. If he haven't yet reached the limit, then we can achieve the same increase by designing a new compression algorithm, without spending billions on computer network upgrades.

These two problems – characterizing randomness and checking limitations of data compression – led to the notion of Kolmogorov complexity.

1.2 Kolmogorov Complexity as a Way of Solving Problems

Let us start with the problem of data compression. In the computer every piece of information is represented as the sequence of 0's and 1's, i.e., as a binary string x. Compression means that instead of storing the original string x, we store some files (pieces of data) such that when necessary, we will be able to get x back. Let p denote the compressed information together with the code necessary to decompress this information. We can view p as a "program" that generates x, a program

written in machine code. The fewer bits in p, the better the compression. So the "optimal" compression can be defined as a compression with the smallest possible number of bits, i.e., the shortest program p that generates x. The length of the shortest program is called the Kolmogorov complexity of the binary string x. Formally $K(x) = \min\{l(p)|p \text{ generates } x\}$.

This notion also helps to distinguish between random and non-random sequence. Indeed, what does it mean for a sequence to be non-random? It means that there is a pattern, for example a sequence consisting of 1 repeated 10,000 times is not random. Since we have a pattern, we can write a reasonably short program that reconstructs the sequence. In the above example you can use the following short program:

for i:=1 to 10000 do print(1);

So usually, if a sequence is non-random, we can compress it and produce a short program that generates the sequence x. On the other hand, if we have a truly random sequence that has no pattern, then the only way to produce this sequence is to enumerate all its bits. So if a sequence is not random, its Kolmogorov complexity is much smaller than its length. If the sequence is random, then the Kolmogorov complexity is close to its length. So we can formalize the randomness of a sequence x as $K(x) \approx l(x)$.

This definition does not solve all the problems related to data compression and randomness, because it has been proven that Kolmogorov complexity is not computable [12].

1.3 Resource Bounded Kolmogorov Complexity

In definition of Kolmogorov complexity, we took into consideration only the length of the program p. But some short programs require such a long running time that they

cannot finish in billions of years. To make the notion of Kolmogorov more realistic, we must limit not only the program size but also its running time.

For example, let us consider a problem from the class NP. A classical problem from this class is the satisfiability problem, in which, given a boolean formula

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2),$$

we check whether it is satisfiable, i.e., whether it is true for some inputs $X = (x_1, \ldots, x_n)$. In general, given Y, we want to find X such that some easy-to-check condition $C(X, Y)$ is satisfied. In propositional satisfiability problem, Y is the boolean formula, X is the sequence of the boolean values, and $C(X, Y)$ is the condition that when we substitute the values X into the formula Y, we get "true." The satisfiability problem is known to be NP–hard, which means, crudely speaking, that most probably there is no algorithm for solving this problem which is always faster than simply checking all 2^n possible combinations of boolean values. We can have a program that takes all possible combinations X of n boolean values, and checks them one after another until it finds the combination which makes the condition $C(X, Y)$ true. This is not a very complicated program to write, it is a short program, but the time taken for this program to finish may be, for large n, billions of years.

More formally, the resource bounded Kolmogorov complexity $K^t(x)$ of an object x bounded by time t is the length of the shortest program $l(p)$ that generates the object x using at most time t [12].

1.4 Quantum Computing

Kolmogorov complexity was invented for traditional (deterministic) computers. However, deterministic computers have their limit. No matter how fast modern computers are, there are still problems that take too much computational time and thus, cannot

yet be handled by modern computers. To solve these problems, we must design faster computers.

Examples of problem that needs faster computation are NP-hard problems. Many problems that we have in real life are NP-hard. Examples of NP-hard problems are optimal scheduling of three or more processors, robot navigation, interval computation, circuit satisfiability, and many more [3, 7]. Let's go over the circuit satisfiability problem. The circuit is a collection of AND, OR and NOT gates connected by wires. The input is a collection of n Boolean values and the output is a single Boolean value. The circuit satisfiability problem asks, given a circuit, whether there is an input that makes the output true, or, conversely, whether there is an input that makes the circuit output false. This problem is useful in digital design because if there is a subcircuit that always returns 0 or 1, then we can replace the elements of this subcircuit with just the output 0 or 1. There are no algorithms that would always solve this problem faster than trying all 2^n possible inputs to the circuit. For an input of a reasonable length $n \approx 300$, we would need 2^{300} computational steps. Even if we use a hypothetical computer for which each step takes the smallest physically possible time (the time during which light passes through the smallest known elementary particle), we would still need more computational steps that can be performed during the (approximately 20 billion years) lifetime of our Universe [10].

So far, the speed of computers has been doubling every 18 months; this is known as Moore's law. According to modern physics, all velocities are bounded by the speed of light, thus to make computer elements faster designers try to decrease the size of these elements. When we go down in size, to a single molecule or atom, then we have to apply the law of quantum mechanics, because quantum mechanics is what describes small particles.

In classical computers, the data is stored as 0's and 1's. For example, with one bit, we can represent two integers 0 and 1; with two bits, we can represent values 00,

01, 10, 11 which describes four integers 0, 1, 2, and 3 etc. In quantum mechanics, we still have states 0 and 1; they are usually written as $|0\rangle$ and $|1\rangle$. In addition to these two cases, a quantum bit (*qubit*) can also be in superposition of these two states. A general state of a single qubit is a superposition $c_0|0\rangle + c_1|1\rangle$, a general state of a 2-qubit system is $c_0|00\rangle + c_1|01\rangle + c_2|10\rangle + c_3|11\rangle$. Here c_0, c_1, c_2, c_3 are complex numbers known as *amplitudes*. The value $|c_0|^2$, $|c_1|^2$, $|c_2|^2$, and $|c_3|^2$, are the probabilities that the stored number is 0, 1, 2, or 3. The total probability is 1: $|c_0|^2 + |c_1|^2 + |c_2|^2 + |c_3|^2 = 1$.

Similarly, a general state of an n-qubit system is a superposition of 2^n states $|0\ldots0\rangle, |0\ldots01\rangle, \ldots, |1\ldots1\rangle$. This means that a quantum computer can, crudely speaking, in one step perform the same mathematical operation on 2^n different inputs. In order to accomplish the same task, a classical computer would have to repeat the same operation 2^n times or one would have to use 2^n processors in parallel [5].

In reality, the situation is not so straightforward. One problem with quantum computing is that while classical computer are deterministic – i.e.,(ideally) return the same result every time, a quantum computer is probabilistic – it returns the correct result only with a certain probability.

With this caveat, quantum algorithms perform better than classical algorithms on some problems. For example, Shor's algorithm can factorize an integer in polynomial time, and Grover's algorithm can search through a unsorted list in $\sqrt{N}$ time as against N time in classical algorithms [8]. Most current cryptographic systems are based on the difficulty of factorization, and with Shor's algorithm, we will be able to crack the encryption of all these cryptographic systems.

As we have mentioned, Kolmogorov complexity has various applications in computing [12], so it's reasonable to assume that it will be very useful to generalize the notion of Kolmogorov complexity to probabilistic and quantum computing.

1.5 Earlier Work

Several quantum versions of Kolmogorov complexity have been proposed [2, 6, 13, 14].

1.5.1 Vitányi's definition

Paul Vitányi in [13, 14] defines quantum Kolmogorov complexity of a state $|x\rangle$ as the smallest "penalized" length of the program p that generates a close state $|z\rangle$, where a penalized length is defined as a length plus the logarithmic term penalizing the program for the difference between $|x\rangle$ and $|z\rangle$:

$K(|x\rangle) = \min_p\{l(p) + \lceil -\log |\langle z|x\rangle|^2 \rceil \ | p \text{ generates } |z\rangle\}$.

What does $|\langle z|x\rangle|^2$ mean? In classical physics, instead of qubits we have bits: 0 and 1. In classical physics, to check whether a result of the computation is correct or not, we can simply "measure" this result and compare it with the desired result.

In quantum mechanics, every measurement distorts the measured state, so comparison is not that simple. Measuring whether $|z\rangle = |x\rangle$ or $|z\rangle \neq |x\rangle$ means, in quantum mechanics, that we apply, to $|z\rangle$, a special measurement procedure the result of which will be either 1 (if $|z\rangle = |x\rangle$) or 0. A general quantum measurement procedure can be described by states that this procedure does not change. These states $|e_1\rangle, \ldots, |e_n\rangle, \ldots$, are mutually orthogonal and form a basis. For every state $|y\rangle$, if $|y\rangle = y_1|e_1\rangle + \ldots + y_n|e_n\rangle + \ldots$, as a result of these measurements, we get $|e_i\rangle$ with probability $|y_i|^2$. In particular, measuring whether $|z\rangle = |x\rangle$ means that we use a basis in which one of the vectors $|e_1\rangle$ is $|x\rangle$. Then, the probability that $|z\rangle = |x\rangle$ is equal to $|y_1|^2$, where y_1 is the coefficient of $|x\rangle$ in the expansion

$$|z\rangle = y_1|x\rangle + y_2|e_2\rangle + \ldots y_n|e_n\rangle + \ldots$$

Taking a dot product of both sides with $|x\rangle$ and using the fact that $|x\rangle, |e_2\rangle, \ldots, |e_n\rangle, \ldots$

are orthogonal unit vectors, we conclude that

$$\langle x|z\rangle = y_1\langle x|x\rangle + y_2\langle x|e_2\rangle + \ldots + y_n\langle x|e_n\rangle + \ldots$$

$$= y_1 \cdot 1 + y_2 \cdot 0 + \ldots + y_n \cdot 0 + \ldots = y_1.$$

So, $|y_1|^2 = |\langle x|z\rangle|^2 (= |\langle z|x\rangle|^2)$ is the probability of "yes" answer to the question "$|z\rangle = |x\rangle$?" when the computation resulted in $|z\rangle$.

If the states are absolutely identical, then the probability is 1 and the the penalty is 0. The smaller the probability, the larger the penalty. If the states $|x\rangle$ and $|z\rangle$ are orthogonal, then the probability is 0, so the penalty is infinite.

1.5.2 Definition by Sophie Laplante and Others

Another definition is given in [2] by S. Laplante, W. V. Dam, and A. Berthiaume. This paper defines quantum Kolmogorov complexity of a state as the smallest quantum length l_q of a program p such that this program p, given an auxiliary parameter k (like number of iterations) produces a state $|y_k\rangle$ that is getting closer and closer to $|x\rangle$ as $k \to \infty$:

$$QC(|x\rangle) = \min\left\{l_q(p)|p(k) \text{ generates } |y_k\rangle \text{ such that } |\langle y_k|x\rangle| \geq 1 - \frac{1}{k}\right\}.$$

The authors also propose a natural resource-bounded version of this definition:

$$QC(|x\rangle, t) = \min\Big\{l_q(p)|p(k) \text{ generates } |y_k\rangle \text{ such that } |\langle y_k|x\rangle| \geq 1 - \frac{1}{k},$$

$$\text{using at most time } t\Big\}.$$

One of the differences between Paul Vitányi's definition [13, 14] and Sophie Laplante et al. definition [2] is that in Vitányi's definition, the program length is calculated in bits, whereas in Sophie Laplante et al. definition the program length is calculated in qubits.

1.5.3 Gacs's Definition of Quantum Kolmogorov Complexity

A third definition was proposed by Peter Gacs [6]. This definition generalized a known equivalent definition of Kolmogorov complexity – as a logarithm of a universal prior semimeasure.

Before we go over the definition of quantum Kolmogorov complexity proposed by Peter Gacs [6], let us recall some notions that we would need to understand the definition.

Definition 1 [6, 12] *A non negative real function $g(x)$ defined on strings is called a* semimeasure *if $\sum_x g(x) \leq 1$ and a measure if the sum is 1.*

Definition 2 [6, 12] *A function is called* lower semicomputable *if there is a monotonically increasing sequence $g_n(x)$ of functions converging to it such that $(n, x) \to g_n(x)$ is a computable mapping into rational numbers.*

Proposition 1 [6, 12] *There is a semicomputable semimeasure μ with the property that for any other semicomputable semimeasure v there's a constant $c_v > 0$ such that for all x we have $c_v \cdot v(x) \leq \mu(x)$.*

A semicomputable semimeasure with the above property is called *universal*. It is known that any two universal semimeasures dominate each other within a multiplicative constant. We can therefore fix one such measure $m(x)$ and call it the *universal probability*.

Kolmogorov complexity can be defined in two ways:

(i) as before, as $K(x) = \min\{l(p)|p \text{ generates } x\}$,

(ii) $C(x) \stackrel{\text{def}}{=} -\log m(x)$ for the universal probability $m(x)$.

It turns out that $C(x)$ is equivalent to the so called prefix Kolmogorov complexity that is defined as

$$C(x) \stackrel{\text{def}}{=} \min\{l(p')|p' \text{ generates } x + \text{ something}\}.$$

One of the difference between Kolmogorov complexity $K(x)$ and prefix Kolmogorov complexity is that $K(x)$ generates the object x exactly, but $C(x)$ generates the object and also some extra bits which are not there in the original object. The Kolmogorov complexity $K(x)$ is known to be close to prefix Kolmogorov complexity $C(x)$ [12]: $C(x) \leq K(x) \leq C(x) + O(\log K(x))$. Peter Gacs generalizes the definition of $C(x)$ to the quantum case. In the classical case, the universal probability is given by m, the quantum analog of universal probability is given by μ. Here μ is the quantum universal (semi-) density matrix given by $\mu = \sum_{ki} p_{ki} |\Phi_{ki}\rangle\langle\Phi_{ki}|$, where $p_{ki} \geq 0$ and $\sum_{k,i} p_{ki} \leq 1$.

Suppose we have a state $|\Psi\rangle = a|\Psi_1\rangle + b|\Psi_2\rangle$, where a and b are complex numbers then the density matrix is given by

$$|\Psi\rangle\langle\Psi| = \begin{bmatrix} aa^* & ab^* \\ ba^* & bb^* \end{bmatrix},$$

where a^* and b^* are the complex conjugates of a and b. In a density matrix, the trace of the matrix is equal to 1, i.e., $aa^* + bb^* = 1$, and in a semi-density matrix, the trace of the matrix is ≤ 1.

In classical computing, the universal probability of a sequence x is given by $m(x)$; in the quantum case, the quantum universal probability of a quantum state $|\Psi\rangle$ is given by $\langle\Psi|\mu|\Psi\rangle$.

In [6], the author has two different definitions for quantum Kolmogorov complexity depending on the order of taking the logarithm and the expectation:

$$\underline{H}(|\Psi\rangle) = -\log\langle\Psi|\mu|\Psi\rangle \tag{1}$$

$$\overline{H}(|\Psi\rangle) = -\langle\Psi|\log\mu|\Psi\rangle \tag{2}$$

To calculate $\langle\Psi|\mu|\Psi\rangle$, we take the eigenvectors of the operator μ, multiply them by $|\Psi\rangle$ and the eigenvalues and add up all these terms [6, 8]. For example, let us consider

the state $|\psi\rangle = 2^{-1/2}(|1\rangle + |2\rangle)$, where $|1\rangle$ and $|2\rangle$ are the eigenvectors of μ, and let $p_1 = 1$ and $p_2 = 0.5$ be the eigenvalues. So, we have,

$$\mu = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix},$$

$$|\psi\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix},$$

$$\langle\psi| = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix},$$

hence

$$\langle\Psi|\mu|\Psi\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix},$$

$$\langle\Psi|\mu|\Psi\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{0.5}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix},$$

$$\langle\Psi|\mu|\Psi\rangle = \begin{bmatrix} \frac{1}{2} + \frac{0.5}{2} \end{bmatrix} = 0.75,$$

and

$$\underline{H}(|\Psi\rangle) = -\log\langle\Psi|\mu|\Psi\rangle = -\log(0.75) = 0.415.$$

For $\overline{H}(|\Psi\rangle)$, we have

$$\log\mu = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix},$$

$$|\psi\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix},$$

$$\langle\psi| = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix},$$

$$\langle\Psi|\log\mu|\Psi\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix},$$

and

$$\langle\Psi|\log\mu|\Psi\rangle = \begin{bmatrix} 0 & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}.$$

Hence, in this example,

$$-\langle\Psi|\log\mu|\Psi\rangle = \frac{1}{2},$$

and

$$\overline{H}(|\Psi\rangle) = -\langle\Psi|\log\mu|\Psi\rangle = 0.5.$$

As we can see from the above example, the difference between the two complexities depending upon the order of taking the logarithm and the expectation can be larger; it is proven that $\underline{H} < \overline{H}$ [6]. It has also been proven that $\underline{H}$ is, within a constant factor, equal to Vitańyi's definition of $K(|X\rangle)$ quantum Kolmogorov complexity

$$\text{const} \cdot K(|X\rangle) \leq \underline{H} \leq \text{const} \cdot K(|X\rangle).$$

1.6 Formulation of the Problem

The proposed definitions of quantum Kolmogorov complexity are important steps towards deep understanding of quantum information. However, these definitions have been proposed in a very heuristic basis: there is no justification as to why the log penalty term was selected by Paul Vitańyi [13, 14] and from where the function $1 - \frac{1}{k}$ selected by Sophie Laplante et al. [2] comes. We need to take time and space into account to make the notion more realistic. It is therefore desirable to come up with some justifications for the proposed definitions. In this thesis, we describe such a justification for the general case of probabilistic computation.

Chapter 2

Kolmogorov Complexity for Probabilistic Computation

2.1 Monte Carlo Algorithms

A typical example of probabilistic computation is Monte-Carlo technique. A Monte-Carlo algorithm is a method of estimating a difficult-to-compute quantity which can be represented as an average of a certain easy-to-compute quantity over a natural (and easy to implement) probability distribution. Historically the first example of Monte-Carlo technique is computing an integral, let's explain this method on a toy example of computing $\int_0^1 x^2 dx$. From the geometric viewpoint, the integral is the area under the curve $y = x^2$. If we have a random point uniformly distributed in the square $[0, 1] \times [0, 1]$, what is the probability that this point is under the curve $y = x^2$? This probability is equal to the ratio between the area of the region under the curve (i.e., the integral) and the area of the entire box. In this case, the box has an area 1, so this probability is exactly equal to the integral. So, to determine the integral, we can take random points x and y and check whether $y \leq x^2$, i.e., whether

the point (x, y) is under the curve. After N such simulations, we count in how many of these simulations we had $y_i \le x_i^2$, and the frequency of $y_i \le x_i^2$ is an estimate for the integral. When $N \to \infty$, this frequency tends to the actual probability $\frac{1}{3}$.

Standard Kolmogorov complexity considers the case when the result of the program is generated exactly. In Monte Carlo programs, we only get an approximation to the desired number. Thus to describe the "complexity" of a real number we must take into consideration not only the length l of the corresponding program, but also the resulting approximation error ε. If the binary string X whose complexity we want to estimate represents the binary expansion of a real number - e.g., a numerical value of some complex integral - then, since Monte-Carlo methods provide only an approximation to the desired number, it is desirable, when describing the number's complexity, to take into consideration not only the length l of the corresponding problem, but also the corresponding approximation error.

In our toy example, we are computing the value $\frac{1}{3} = 0.333333\ldots = 0.010101\ldots_2$. If after running N simulations, we get the approximate value $0.3 = 0.0100110011\ldots_2$, the approximation error $\varepsilon = 0.033\ldots_{10}$. Thus we must look for a combination function $f(l, \varepsilon)$.

2.2 Motivation

So, we are looking for the combination function $f(l, \varepsilon)$. Here we can look at ε as a additional resource. It is known, from statistics, that the mean square approximation error ε of a Monte-Carlo method is asymptotically equal to $\frac{1}{\sqrt{N}}$, where N is the number of iterations of the corresponding Monte-Carlo simulation [4]. A Monte-Carlo program consists of N simulations, i.e., it has the form

for i:=1 to N do

(iterations)

If we fix the number of bits l in this program, then we thus fix the number of bits b in the binary expansion of N. The larger N, the more iterations we perform, and the more accurate the result. So, within the fixed number of bits, if we want the most accurate estimate, we must use the largest integer that can be described by these bits, i.e., $N = 1\ldots1$ (b times). If we allow one more bit in the program, i.e., if we consider programs of length $l' = l+1$, we can then use $N' = 1\ldots1$ ($b+1$ times), i.e., $N' \approx 2N$. With $N' = 2N$, we achieve an accuracy $\varepsilon' = \frac{\varepsilon}{\sqrt{2}}$.

We do not want the "complexity" $C(l,\varepsilon)$ of the Monte-Carlo program to change if we simply use a different parameter N. So, it is natural to require that the combination function be invariant relative to this transformation $l \to l+1$, $\varepsilon \to \frac{\varepsilon}{\sqrt{2}}$:

$$f(l,\varepsilon) = f\left(l+1, \frac{\varepsilon}{\sqrt{2}}\right) \tag{1}$$

2.3 Main Result

Definition 3 *We say that a function $f(l,\varepsilon)$ is* invariant *if it satisfies* (1) *for all positive integers l and all real numbers $\varepsilon \in [0,1]$.*

Definition 4 *A function $F(z)$ of one variable is called* monotonic *if for every z', z'', $z' < z''$ implies $F(z') < F(z'')$.*

Definition 5 *We say that a function $f(l,\varepsilon)$ of two variables is* monotonic *if for every $l', \varepsilon', l'', \varepsilon''$, the following two conditions hold:*

- *$l' \le l''$ and $\varepsilon' < \varepsilon''$, implies $f(l',\varepsilon') < f(l'',\varepsilon'')$, and*
- *$l' < l''$ and $\varepsilon' \le \varepsilon''$, implies $f(l',\varepsilon') < f(l'',\varepsilon'')$.*

Theorem 1 *A increasing function $f(l,\varepsilon)$ is invariant(in the sense of (1)) if and only if $f(l,\varepsilon) = g(l + 2\log_2 \varepsilon)$ for some increasing function $g(z)$ of one variable.*

To prove the above theorem we will first have to prove the following Lemma.

Lemma 1 *A monotonic function $f(l,\varepsilon)$ is invariant if and only if $f(l,\varepsilon) = F(\varepsilon \cdot \sqrt{2}^{\,l})$ for some monotonic function $F(z)$ of one variable.*

Proof

To prove the lemma we need to prove the following two things:

(i) If $\varepsilon' \cdot \sqrt{2}^{\,l'} = \varepsilon'' \cdot \sqrt{2}^{\,l''}$ then $f(l',\varepsilon') = f(l'',\varepsilon'')$. This, as we will prove later, will guarantee that $f(l,\varepsilon) = F(\varepsilon \cdot \sqrt{2}^{\,l})$ for some function $F(z)$ of one variable.

(ii) We need to prove that if $\varepsilon' \cdot \sqrt{2}^{\,l'} < \varepsilon'' \cdot \sqrt{2}^{\,l''}$ then $f(l',\varepsilon') < f(l'',\varepsilon'')$. This, as we will prove later, will guarantee that the above function $F(z)$ is monotonic.

1°. Let us start by proving part (i). Let $\varepsilon', l', \varepsilon'', l''$ are such that $\varepsilon' \cdot \sqrt{2}^{\,l'} = \varepsilon'' \cdot \sqrt{2}^{\,l''}$. Let us prove that $f(l',\varepsilon') = f(l'',\varepsilon'')$.

The equality in Equation (1) is true for every l and ε. In particular it is true for $l = l'$ and $\varepsilon = \varepsilon'$, i.e.,

$$f(l',\varepsilon') = f\left(l'+1, \frac{\varepsilon'}{\sqrt{2}}\right)$$

It is also true for $l = l'+1$ and $\varepsilon = \dfrac{\varepsilon'}{\sqrt{2}}$.

Substituting $l = l'+1$ and $\varepsilon = \dfrac{\varepsilon'}{\sqrt{2}}$ into the formula (1) we conclude that

$$f\left(l'+1, \frac{\varepsilon'}{\sqrt{2}}\right) = f\left(l'+2, \frac{\varepsilon'}{\sqrt{2}^{\,2}}\right).$$

Thus $f(l',\varepsilon') = f\left(l'+1, \dfrac{\varepsilon'}{\sqrt{2}}\right) = f\left(l'+2, \dfrac{\varepsilon'}{\sqrt{2}^{\,2}}\right)$. Similarly we can prove that

$$f(l',\varepsilon') = f\left(l'+1, \frac{\varepsilon'}{\sqrt{2}}\right) = f\left(l'+2, \frac{\varepsilon'}{\sqrt{2}^{\,2}}\right) = f\left(l'+3, \frac{\varepsilon'}{\sqrt{2}^{\,3}}\right)$$

$$= \ldots = f\left(l'+k, \frac{\varepsilon'}{\sqrt{2}^{\,k}}\right) \qquad (2)$$

for an arbitrary k. In particular, for $k = l''$, we conclude that

$$f(l', \varepsilon') = f\left(l' + l'', \frac{\varepsilon'}{\sqrt{2}^{l''}}\right). \tag{3}$$

Similarly we can prove from (1) and (2)

$$f(l'', \varepsilon'') = f\left(l'' + 1, \frac{\varepsilon''}{\sqrt{2}}\right) = f\left(l'' + 2, \frac{\varepsilon''}{\sqrt{2}^{2}}\right) = f\left(l'' + 3, \frac{\varepsilon''}{\sqrt{2}^{3}}\right)$$

$$= ... = f\left(l'' + k, \frac{\varepsilon''}{\sqrt{2}^{k}}\right) \tag{4}$$

for an arbitrary k. In particular, for $k = l'$, we conclude that

$$f(l'', \varepsilon'') = f\left(l'' + l', \frac{\varepsilon''}{\sqrt{2}^{l'}}\right). \tag{5}$$

Let us denote $l' + l''$ by L. Then equation (3) takes the form

$$f(l', \varepsilon') = f\left(L, \frac{\varepsilon'}{\sqrt{2}^{l''}}\right). \tag{6}$$

and equation (5) takes the form

$$f(l'', \varepsilon'') = f\left(L, \frac{\varepsilon''}{\sqrt{2}^{l'}}\right). \tag{7}$$

Now substituting $l'' = L - l'$ into equation (6) we conclude

$$f(l', \varepsilon') = f\left(L, \frac{\varepsilon' \cdot \sqrt{2}^{l'}}{\sqrt{2}^{L}}\right). \tag{8}$$

Substituting $l' = L - l''$ into equation (7) we conclude that

$$f(l'', \varepsilon'') = f\left(L, \frac{\varepsilon'' \cdot \sqrt{2}^{l''}}{\sqrt{2}^{L}}\right). \tag{9}$$

Since we had assumed that $\varepsilon' \cdot \sqrt{2}^{l'} = \varepsilon'' \cdot \sqrt{2}^{l''}$ we can now conclude that

$$f(l', \varepsilon') = f(l'', \varepsilon''). \tag{10}$$

The statement is proven.

2°. Let us now prove that there exists a function $F(z)$ of one variable such that for every l, ε

$$f(l, \varepsilon) = F(\varepsilon \cdot \sqrt{2}^l) \tag{11}$$

Let us define $F(z)$ and show that for this $F(z)$, the equality (11) indeed holds for every l and ε. To define $F(z)$ for a given z we must find an integer l and a real number ε for which $z = \varepsilon \cdot \sqrt{2}^l$. For example we can pick $l = 0$ and $\varepsilon = z$. We define $F(z)$ as

$$F(z) \stackrel{\text{def}}{=} f(0, z). \tag{12}$$

By definition of F, we have

$$F(\varepsilon \cdot \sqrt{2}^l) = f(0, \varepsilon \cdot \sqrt{2}^l). \tag{13}$$

So, the desired equality (11) takes the following form

$$f(l, \varepsilon) = f(0, \varepsilon \cdot \sqrt{2}^l). \tag{14}$$

We have already proven that if $\varepsilon' \cdot \sqrt{2}^{l'} = \varepsilon'' \cdot \sqrt{2}^{l''}$ then $f(l', \varepsilon') = f(l'', \varepsilon'')$. In our case for $l' = l, \varepsilon' = \varepsilon, l'' = 0, \varepsilon'' = \varepsilon \cdot \sqrt{2}^l$ we get

$$\varepsilon \cdot \sqrt{2}^l = \varepsilon \cdot \sqrt{2}^l. \tag{15}$$

Hence we have proved that $f(l, \varepsilon) = F(\varepsilon \cdot \sqrt{2}^l)$ for the function $F(z)$ (defined by the formula (12)).

3°. Let's prove part (ii) now. From (8) and (9), we conclude that

$$f(l', \varepsilon') = f\left(L, \frac{\varepsilon' \cdot \sqrt{2}^{l'}}{\sqrt{2}^L}\right)$$

and

$$f(l'', \varepsilon'') = f\left(L, \frac{\varepsilon'' \cdot \sqrt{2}^{l''}}{\sqrt{2}^L}\right).$$

We assumed that $\varepsilon' \cdot \sqrt{2}^{l'} < \varepsilon'' \cdot \sqrt{2}^{l''}$, therefore

$$\frac{\varepsilon' \cdot \sqrt{2}^{l'}}{\sqrt{2}^{L}} < \frac{\varepsilon'' \cdot \sqrt{2}^{l''}}{\sqrt{2}^{L}}.$$

We also assumed that the function f is monotonic. So, from the Definition 3 of monotonicity, we conclude that

$$f\left(L, \frac{\varepsilon' \cdot \sqrt{2}^{l'}}{\sqrt{2}^{L}}\right) < f\left(L, \frac{\varepsilon'' \cdot \sqrt{2}^{l''}}{\sqrt{2}^{L}}\right).$$

hence

$$f(l', \varepsilon') < f(l'', \varepsilon''). \tag{16}$$

The statement is proven.

4°. Let us now prove that the function $F(z)$ is monotonic.

According to Definition 2, we need to prove if $z' < z''$ then $F(z') < F(z'')$. By definition of $F(z)$ (formula (12)) we have $F(z) = f(0, z)$. So, what we need to prove is that if $z' < z''$ then

$$f(0, z') < f(0, z'') \tag{17}$$

This follows directly from the fact that the function $f(l, \varepsilon)$ is monotonic (Definition 3). In our case for $l' = 0, \varepsilon' = z', l'' = 0, \varepsilon'' = z''$, we get $f(0, z') < f(0, z'')$.

Hence we have proved that the function $F(z)$ is monotonic. The lemma is proven. Now, we can prove the theorem itself.

We have a function $F(z)$, so we define a new function

$$g(z) \stackrel{\text{def}}{=} F(\sqrt{2}^{z}) \tag{18}$$

Given X, how can we find z such that

$$g(z) = F(X)? \tag{19}$$

By definition, we know that $g(z) = F(\sqrt{2}^{\,z})$. So, to find z for which $g(z) = F(X)$, we must find z for which $\sqrt{2}^{\,z} = X$. Therefore $z = \log_{\sqrt{2}} X$. Substituting this value of z into (19), we get

$$F(X) = g(\log_{\sqrt{2}} X). \tag{20}$$

According to Lemma 1, we have

$$f(l, \varepsilon) = F(\varepsilon \cdot \sqrt{2}^{\,l}).$$

Substituting $X = \varepsilon \cdot \sqrt{2}^{\,l}$ into the formula (20), we conclude that

$$f(l, \varepsilon) = g(z),$$

where

$$z \stackrel{\text{def}}{=} \log_{\sqrt{2}}(\varepsilon \cdot \sqrt{2}^{\,l})$$

The logarithm of a product is equal to the sum of the logarithms, so

$$z = \log_{\sqrt{2}} \varepsilon + \log_{\sqrt{2}} \sqrt{2}^{\,l}.$$

By properties of a logarithm,

$$\log_{\sqrt{2}} \varepsilon = \frac{\log_2 \varepsilon}{\log_2 \sqrt{2}} = \frac{\log_2 \varepsilon}{\frac{1}{2}} = 2 \log_2 \varepsilon$$

By definition of a logarithm,

$$\log_{\sqrt{2}} \left(\sqrt{2}^{\,l} \right) = l,$$

so

$$z = \log_{\sqrt{2}}(\varepsilon \cdot \sqrt{2}^{\,l}) = 2 \log_2 \varepsilon + l$$

and therefore,

$$f(l, \varepsilon) = g(l + 2 \log_2 \varepsilon) \tag{21}$$

The theorem is proven.

2.4 Comparison of Our Result With Known Definitions of Quantum Kolmogorov Complexity

The existing definitions of quantum Kolmogorov complexity use the quantity $|\langle x|y\rangle|$ to describe how close the state $|x\rangle$ and $|y\rangle$ are. In order to compare these definitions with our formula, let us describe the distance between the states in terms of quantity $|\langle x|y\rangle|$.

In quantum mechanics, the same state can be represented by several different vectors; namely, for every vector y, the vector $y' = ye^{i\alpha}$, where α is a real-valued constant, represents the same state. This fact is easy to check, because for every measurement procedure, the vectors y and $ye^{i\alpha}$ lead to the same probabilities.

If a state was represented by a unique vector, then we could describe the distance between the two states x and y as the Euclidean distance, i.e., as $\rho(x,y) = |x-y| = \sqrt{|x-y|^2}$. Since, in reality, each state corresponds to several different vectors, we cannot define the state as simply $|x-y|$, because $y = -x$ and x represent the same state, but $|x-y|$ is different from 0. Therefore, we define the distance between the states described by vectors x and y not as $|x-y|$, but as

$$\rho(x,y) \stackrel{\text{def}}{=} \min_{\alpha,\beta} |e^{i\alpha}x - e^{i\beta}y|. \tag{22}$$

Let us simplify this formula. Since the square is a monotonic function, $\rho^2(x,y)$ is equal to the minimum of the expression

$$z \stackrel{\text{def}}{=} |e^{i\alpha}x - e^{i\beta}y|^2 = (e^{i\alpha}x - e^{i\beta}y)(e^{i\alpha}x - e^{i\beta}y)^*.$$

Multiplying the terms $(e^{i\alpha}x - e^{i\beta}y)(e^{i\alpha}x - e^{i\beta}y)^*$, we conclude that

$$z = |e^{i\alpha}x|^2 + |e^{i\beta}y|^2 - 2 \cdot Re(e^{i\alpha}x \cdot e^{i\beta}y). \tag{23}$$

Since x and y are unit vectors, we get

$$z = 1 + 1 - 2 \cdot Re(e^{i\alpha}x \cdot e^{i\beta}y), \tag{24}$$

i.e.,

$$z = 2 - 2 \cdot Re(e^{i(\alpha-\beta)}\langle x|y\rangle). \tag{25}$$

A complex number $\langle x|y\rangle$ can be represented as $\langle x|y\rangle = r \cdot e^{i\theta}$, where $r = |\langle x|y\rangle|$. Then,

$$z = 2 - 2 \cdot Re(e^{i(\alpha-\beta)} r \cdot e^{i\Theta}), \tag{26}$$

i.e.,

$$z = 2 - 2 \cdot Re(r \cdot e^{i(\alpha-\beta+\Theta)}), \tag{27}$$

or

$$z = 2 - 2(r \cdot \cos(\alpha - \beta + \Theta)). \tag{28}$$

We know that $\rho^2(x, y)$ is equal to the minimum of this expression z. This is obtained if the cos term $r \cdot \cos(\alpha - \beta + \Theta)$ is the largest. It is the largest when the cosine is equal to 1. So, we can conclude that the square of the distance is equal to

$$\rho^2 = 2 - 2|\langle x|y\rangle|.$$

In Sophie Laplante's definition, we want $|\langle x|y_k\rangle| \geq 1 - \frac{1}{k}$. In terms of $\rho^2 = 2-2|\langle x, y\rangle|$, this condition is equivalent to $\rho^2(x, y_k) \leq \frac{2}{k}$, or, equivalently, to $\rho(y_k, x) \leq \sqrt{\frac{2}{k}}$.

This fits very well with our Monte Carlo based approach: in this approach, the approximation error ε decreases exactly as $\frac{1}{\sqrt{k}}$. The fact that we have a logarithmic terms $\log \varepsilon$ is in line with Vitányi's definition.

Chapter 3

Resource Bounded Kolmogorov Complexity for Probabilistic Computation

3.1 Motivation

In Chapter 2, we have come up with the formula for Kolmogorov complexity for probabilistic computation $f(l, \varepsilon)$ where we take into consideration not only the length of the program l but also the accuracy ε with which we get the answer. In this chapter, what we are trying to do is come up with the definition for resource bounded Kolmogorov complexity for probabilistic computation $f(l, t, \varepsilon)$, where we take into consideration time t as an additional resource.

Similarly to Chapter 2, if we add one more bit to the length of the program ($l' = l + 1$), then we can use twice as many Monte-Carlo iterations $N' = 2N$, as a result of which, in twice larger time $t' = 2t$, we achieve accuracy $\varepsilon' = \frac{\varepsilon}{\sqrt{2}}$. The "complexity measure" $f(l, t, \varepsilon)$ should not change if we simply change N to $2N$, so

we arrive at the following requirement:

$$f(l,t,\varepsilon) = f\left(l+1, 2t, \frac{\varepsilon}{\sqrt{2}}\right). \tag{1}$$

In the case of resource bounded Kolmogorov complexity, there is another requirement similar to the one described in [11]. This requirement can be illustrated on an example of a string x that is a solution to an NP-hard problem like propositional satisfiability. In other words, x is a binary string for which an easy-to-check condition $C(x, y_0)$ is satisfies for input y_0. A natural way to find x is to generate, one by one, all possible strings of given length n and check the condition $C(x, y_0)$ until this condition is satisfied. This exhaustive search requires, in the worst case, checking all possible binary strings of length $n-$ all 2^n of them. In other words, for this algorithm, the worst-case running time is $t = 2^n$. We can decrease this running time if we know the first bit x_1 of the sequence $x = x_1 \ldots x_n$. Then, instead of generating all 2^n possible sequences $x_1, \ldots, x_n$, we only have to check 2^{n-1} sequences $x_1^0 x_2 \ldots x_n$. The downside is that we need to store the bit x_1^0 in the program. So, the length of the program increases by one bit $l \to l+1$, while its time decreases to a half: $t \to \dfrac{t}{2}$. Similarly, we can hardcode the second bit into the program, the third bit, etc.

Vice versa, if a bit x_i is hardcoded into a program, we can delete it from the program and instead, try both $x_i = 0$ and $x_i = 1$. The length of the program is decreased by 1: $l' = l - 1$, while the time is increased twice: $t' = 2t$. The complexity measure $f(l,t,\varepsilon)$ should not change if we simply hard-code or delete a bit, so we arrive at the following requirement:

$$f(l,t,\varepsilon) = f\left(l-1, 2t, \varepsilon\right). \tag{2}$$

We want to describe all the functions satisfying conditions (1) and (2). Ideally, we should be able to describe such functions for all possible values of l, t, and ε, at present, we are only able to describe it for the case when $2^l \cdot t$ has the form $2^l \cdot t = 4^k$ for some integer k.

3.2 Main Result

Theorem 2 *If an increasing function $f(l,t,\varepsilon)$ is invariant (in the sense of* (1) *and* (2)*) then for all values for which $2^l \cdot t = 4^k$, we have $f(l,t,\varepsilon) = h(l + \log_2 t + 4\log_2 \varepsilon)$ for some increasing function $h(z)$ of one variable.*

Proof

Let $f(l,t,\varepsilon)$ be an increasing function that satisfies condition (1) and (2). In [11], it was proven that, if we have an increasing function of two variables $f(l,t)$ which satisfies the condition

$$f(l,t) = f(l-1,2t) \tag{3}$$

for all l and t, then

$$f(l,t) = F(t \cdot 2^l) \tag{4}$$

for some increasing function $F(z)$. In our case, we have an increasing function of three variables, so, to apply the theorem from [11], we fix ε. Namely, for every $\varepsilon > 0$, we define a new function

$$f_\varepsilon(l,t) \stackrel{\text{def}}{=} f(l,t,\varepsilon). \tag{5}$$

Substituting $f_\varepsilon(l,t)$ instead of $f(l,t,\varepsilon)$ into the equation (2), we conclude that

$$f_\varepsilon(l,t) = f(l-1,2t,\varepsilon). \tag{6}$$

By definition of the function f_ε, the right-hand side of this equation can be rewritten as $f_\varepsilon(l-1,2t)$. So, the increasing function $f_\varepsilon(l,t)$ of two variables satisfy the condition

$$f_\varepsilon(l,t) = f_\varepsilon(l-1,2t). \tag{7}$$

Thus, due to the above mentioned theorem [11], we conclude that

$$f_\varepsilon(l,t) = F_\varepsilon(t \cdot 2^l) \tag{8}$$

for some increasing function $F_\varepsilon(z)$.

Let us define a new function of two variable

$$F(z,\varepsilon) \stackrel{\text{def}}{=} F_\varepsilon(z). \tag{9}$$

Hence, from the definition of $F(z,\varepsilon)$ and the definition of $f_\varepsilon(l,t)$, we can conclude that (8) turns into:

$$f(l,t,\varepsilon) = F\left(t\cdot 2^l,\varepsilon\right). \tag{10}$$

Substituting the expression (10) for $f(l,t,\varepsilon)$ into the condition (1), we conclude that

$$F(t\cdot 2^l,\varepsilon) = F\left(2t\cdot 2^{l+1},\frac{\varepsilon}{\sqrt{2}}\right). \tag{11}$$

Since $2^l\cdot t = 4^k$, we get $2^{l+1}\cdot 2t = 2^l\cdot 2\cdot 2t = 4\cdot(2^l\cdot t) = 4\cdot 4^k$, so (11) takes the form

$$F(4^k,\varepsilon) = F\left(4\cdot 4^k,\frac{\varepsilon}{\sqrt{2}}\right). \tag{12}$$

Let us now introduce a new function

$$\tilde{F}(k,\varepsilon) \stackrel{\text{def}}{=} F(4^k,\varepsilon). \tag{13}$$

In terms of this new function, the equation (12) takes the following form:

$$\tilde{F}(k,\varepsilon) = \tilde{F}\left(k+1,\frac{\varepsilon}{\sqrt{2}}\right). \tag{14}$$

This function $\tilde{F}$ is monotonic since the function $F(z,\varepsilon)$ was monotonic. We have already proven, in Chapter 2, that monotonic function satisfying this condition (14) can be represented as

$$\tilde{F}(k,\varepsilon) = g(k+2\log_2\varepsilon) \tag{15}$$

for some increasing function $g(z)$ of one variable. Thus, when $2^l\cdot t = 4^k$, we have

$$F(4^k,\varepsilon) = F(2^l\cdot t,\varepsilon) = g(k+2\log_2\varepsilon). \tag{16}$$

By (10),

$$f(l,t,\varepsilon) = F\left(2^l \cdot t, \varepsilon\right) = g(k + 2\log_2 \varepsilon). \tag{17}$$

Since $2^l \cdot t = 4^k$, we have

$$k = \frac{l + \log_2 t}{2} \tag{18}$$

and therefore,

$$f(l,t,\varepsilon) = g\left(\frac{l + \log_2 t}{2} + 2\log_2 \varepsilon\right) \tag{19}$$

To complete the proof, let us define a new function $h(z) \stackrel{\text{def}}{=} g\left(\frac{z}{2}\right)$. Then, $g(t) = h(2t)$, and hence, the formula (19) takes the desired form:

$$f(l,t,\varepsilon) = h(l + \log_2 t + 4\log_2 \varepsilon). \tag{20}$$

The theorem is proven.

Chapter 4

Conclusions

In many areas of computing ranging from formalizing expert reasoning to estimating algorithm complexity to steganography, notions of Kolmogorov complexity and resource bounded Kolmogorov complexity turned out to be very useful. In this thesis, we have proposed and justified a generalization of these notions to probabilistic algorithms. Our results provide a motivation for heuristic quantum generalizations of Kolmogorov complexity originally proposed by Vitányi, Laplante and others.

References

[1] C. H. Bennett, and P. W. Shor, *Quantum information theory*, IEEE Transactions on Information Theory, 1998, Vol. 44, No. 6, pp. 2724–2742.

[2] A. Berthiaume, W. V. Dam, and S. Laplante, *Quantum Kolmogorov Complexity*, Proceedings of the 15th Annual IEEE Conference on Computational Complexity, 2000, pp. 240–249.

[3] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, 1990.

[4] A. Dubi, *Monte Carlo Applications in Systems Engineering*, John Wiley and Sons, New York, 2000.

[5] A. Ekert, P. Hayden, and H. Inamori, *Basic concepts in quantum computation*, Lectures on quantum computation, Les Houches summer school, 1999, http://www.qubit.org/oldsite/intros/comp/houches.ps

[6] P. Gacs, *Quantum algorithmic entropy*, Proceedings of the 16th Annual IEEE Conference on Computational Complexity, 2001, pp. 274–283.

[7] M. Garey, and D. Johnson, *Computers and Intractibility: A guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, 1979.

[8] J. Gruska, *Quantum Computing*, McGraw-Hill, London, 1999.

[9] V. Kreinovich, M. Koshelev and Y. Yam, *Towards the use of aesthetics in decision making: Kolmogorov Complexity formalizes Birkhoff's idea*, Bulletin of the European Association for Theoretical Computer Science (EATCS), 1998, Vol. 66, pp. 166–170.

[10] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer Academic Publishers, Dordrecht, 1998.

[11] V. Kreinovich, L. Longpré, and M. Koshelev, *Kolmogorov Complexity, statistical regularization of inverse problems, and Birkhoff's formulization of beauty*, Proceedings of the SPIE Conference on Bayesian Inference for Inverse Problems, San Diego, California, 1998, SPIE Vol. 3459, pp. 159–170.

[12] M. Li and P. M. B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, New York, 1997.

[13] P. M. B. Vitányi, *Three approaches to the quantitative definition of information in an individual pure Quantum state*, Proceedings of the 15th Annual IEEE Conference on Computational Complexity, 2000, pp. 263–270.

[14] P. M. B. Vitányi, *Quantum Kolmogorov Complexity based on classical descriptions*, IEEE Transactions on Information Theory, 2001, Vol. 47, No. 6, pp. 2464–2479.

[15] S. Vorobyov, *Lecture notes on Kolmogorov Complexity and Its Applications,* Computer Science department, Upssala University, Sweden, 2000, www.csd.uu.se/~vorobyov/Courses/KC/2000/all.ps

[16] Website of Centre for Quantum Computation, University of Oxford, http://www.qubit.org