Electronic Colloquium on Computational Complexity
ECCC Technical Reports Series 1994 Report Nr: TR94–013

Complexity Theory and Genetics

(extended abstract)

P.Pudlák
Mathematical Institute
Academy of Sciences
Prague †

Received *December 15, 1994*

Abstract. We introduce a population genetics model in which the operators are effectively computable – computable in polynomial time on Probabilistic Turing Machines. We shall show that in this model a population can encode easily large amount of information from enviroment into genetic code. Then it can process the information as a paralel computer. More precisely, we show that it can simulate polynomial space computations in polynomially many steps, even if the recombination rules are very simple.

† An essential part of this research has been done while visiting Fachbereich Informatik, Universität Dortmund as a a Humboldt Fellow.

1 Introduction

The evolution of life is surely one of the most interesting questions in science. Mathematical genetics is a growing field and has produced deep mathematical results. Nevertheless very little has been done from the point of view of complexity theory. This is paradoxical, since the main problem is essentially a question about the complexity: how is it possible that highly organized (complex) organism have evolved? In mathematical population genetics evolutionary operators are studied from the point of view of dynamics without considering their computational complexity. In this paper we will consider some consequences of the assumption that the evolutionary operators are efficiently computable (in polynomial time on Probabilistic Turing Machines).

We know that the present organisms have evolved by natural selection. We know also basic facts about the mechanism by which the information about the organisms is stored and passed to new generations. It is based on the genetic code represented by a small number of DNA's, where each DNA can be thought of as a string in a finite alphabet. There are essentially two mechanisms by which new codes can be generated: crossing-over and random mutations. The presently accepted theory is that new genes (parts of the genetic code) are formed by mutations and then, provided they improve the fitness, they are spread through the population.

Consider a (population of a) species as one system and the environment in which it lives as another. Let us think of the species as frequencies of particular codes and a mechanism to produce new ones. In this setting the species receives information about the changing environment by "seeing" that for some codes the frequency is growing and for others decreasing.[1] The species reacts by changing frequencies and, in particular, producing new codes. The reaction may be very simple: let the selective forces act freely, except that some mutation ratio is kept. The mutations are needed to keep some variability of the species, so that it can adapt, if the environment changes and also for evolution of better adapted organisms. It is however conceivable that the species has already recorded some changes of environment and has developed better reactions to its changes.

The purpose of this article is to ask the question, whether it is possible that species use some less trivial reactions which require some computation using the information obtained from the environment as an input. To this end we shall present a model which reflects the real heredity mechanism and show that very complex computations are possible in it. Once the possibility of computations on the level of genetic code is shown theoretically, we can look for such computations in Nature.

As we want to compare "genetic" computations with computations on classical devices, the inputs must be given in the same way. The environment is, however, an essentially different form of input, the interaction may be very complex and there is no obvious way to identify it with a sequence of 0's and 1's. Therefore we shall first show that for any string, say of 0's and 1's, natural selection can change the population so that this string appears on a particular site of the code in almost all organisms. Then we shall confine myself to the situation where only this part of the code is used as input and ignore the environment. There are probably many other ways in which the species can get information. We shall show another one, namely that it can compute with high precision the frequency of occurrences of particular pieces of the code in the same sense as above, i.e. the number will be encoded in almost all organisms.

We shall introduce a model, which we call *Genetic Turing Machines,* and compare it with the standard mathematical model of computation the Turing Machine. Roughly speaking a Genetic Turing Machine is a population of tapes with an evolutionary operator which is computable by a Probabilistic Turing Machine in polynomial time. Initially the tapes contain the input string at the beginning and the rest is filled uniformly with one symbol from the finite alphabet. The population develops in discrete generations (computation steps) according to the evolutionary operator. The mating is completely random and the population is considered to be infinite. Both assumptions are, of course, unrealistic, however they simplify the computations considerably. The concept of Genetic Turing Machines is interesting also from purely mathematical point of view, since it naturally generalizes Probabilistic Turing Machines by replacing a liner operator by a quadratic one.

Then we shall consider the sets computable in small number of steps on Genetic Turing Machines, where *small* is represented as *polynomial in the length of the input.* We shall show that the class of such sets is equal to $\mathcal{PSPACE}$, the class of sets computable on Turing Machines with polynomial restriction on the tape. Though unproven, it is generally assumed that $\mathcal{PSPACE}$ is a much larger class than $\mathcal{P}$, the class of

[1] We shall show in Section 3 that theoretically the information about the frequency of a particular trait can be computed and spread through the population very fast.

sets computable on Turing Machines with polynomial restriction on time, which is accepted as a good mathematical approximation of efficiently computable sets. This is also true for the probabilistic version of $\mathcal{P}$, denoted by $\mathcal{BPP}$, where the machine can use random bits in the computation. The larger power of Genetic Turing Machines is, of course, enabled by the fact that the population is large (note that exponentially large finite population would suffice). We cannot use the simulation of $\mathcal{PSPACE}$ to design better computers etc. The meaning of this result is different: it shows that the kind of parallelism that Genetic Turing Machines use is positively correlated with efficiency, namely, if we increase the population we get more power. In particular, in Nature one generation takes a lot of time if compared with one step on a computer; the parallelism can partially make up for this handicap. The conclusion for computer science is that Genetic Turing Machine is a model of parallel computation and that a "random architecture" can be very powerful.

A natural objection is that it is not possible to realize such computations with the restricted means which we can expect in the case of DNA's. In order to counter this objection we shall show that a general Genetic Turing Machine can be efficiently simulated by a special one with a very restricted evolution operator. The restriction is that only crossing-over is allowed, where the positions on which the strings are switched are determined purely by small neighborhoods of the positions. (The underlying idea is that Turing Machines can be simulated by rewriting systems and crossing-over is sufficiently general to simulate rewriting rules.)

This research is related also to the popular field of Genetic Algorithms, however our goals are different. Genetic Algorithms serve as heuristics for solving practical problems, usually optimization problems, which are difficult to analyze theoretically. The main tool is a suitable selection operator. What we suggest here is to study the power of genetic type computations from the point of view of complexity theory. We want to investigate how evolution can be controlled by heredity mechanisms. Nevertheless the results of this paper do have some relevance for Genetic Algorithms. In particular, our simulation of $\mathcal{PSPACE}$ in polynomial time shows that, at least in principle, algorithms can be parallelized using the genetic approach.

2 Basic concepts and notation

In this section we recall some basic concepts from complexity theory and genetics. We start with complexity theory.

The standard model of computation used in complexity theory is the *Turing Machine.* A Turing Machine consists of a *finite device* which controls the computation and infinite memory represented as an infinite *tape.* The tape has cells where symbols from a fixed *finite alphabet* can be written and rewritten using a *head.* The device can be represented by a finite automaton or simply by a program with instructions about the two possible movements of the head and possible symbols which are read and written. We shall confine ourselves to one-tape Turing Machines. Multitape Turing Machines are only needed when complexity is to be determined more precisely.

The machine starts with an input word written on the tape; the rest of the tape cells contain some fixed symbol. After the control device reaches one of particular instructions it stops. The output is what is written on the tape at this moment. If we want only to recognize some set of inputs, we label some stop instructions as accepting and some as rejecting. Thus the machine determines the set of words on which it stops on an accepting instruction.

A *Probabilistic Turing Machine* has moreover the possibility to toss a fair coin during the computation and act according to this random bit. Thus the machine does not compute a function, but a *random variable.* For computing sets we use *bounded error* probabilistic Turing Machines. This is a machine which, for each input x, accepts x with probability at least $3/4$ or with probability at most $1/4$. This dichotomy is used to define the set of inputs accepted by the machine. (Clearly, by running the machine several times we can decide with high confidence, if the input word is accepted or not.)

There are two basic complexity measures *time* and *space.* Time is the number of steps used in the computation; space is the number of tape cells used in the computation. In the theory we are interested only in asymptotical behavior of these quantities. We measure the time and space requirement for computation of a given set as a function $f(n)$ depending on the length of the input word $n = |x|$.

We shall consider the following complexity classes.

$\mathcal{P}$ is the class of sets which can be accepted in time bounded by a polynomial on a Turing Machine.

$\mathcal{BPP}$ is the class of sets which can be accepted in time bounded by a polynomial on a bounded error Probabilistic Turing Machine.

$\mathcal{PSPACE}$ is the class of sets which can be accepted in space bounded by a polynomial on a Turing Machine.

$\mathcal{P}$ and $\mathcal{BPP}$ are used as theoretical approximations of what can really be computed using a deterministic, respectively probabilistic, real computing device. We have the inclusions $\mathcal{P} \subseteq \mathcal{BPP} \subseteq \mathcal{PSPACE}$. It has been conjectured that both inclusions are proper, however the proofs are beyond our present means. $\mathcal{P}$ and $\mathcal{BPP}$ do not seem to be very far apart, but there is strong evidence that $\mathcal{PSPACE}$ is much larger. Note that in particular a machine with a polynomially bounded space can actually run in exponential time, since the number of possible configurations on the tape of polynomial length is exponential.

We shall now recall some facts from genetics. The most important genetic information is encoded in strings of DNA. Each string has its own *chromosome* and the chromosomes occur in pairs. The two DNA's from a pair of chromosomes encode information about the same traits of the organism. (The resulting traits are (weighted) combinations of the information in the two DNA's.) The number of chromosomes is a fixed even number for each species and in most cases it is between 14 and 60. The length of the DNA is also fixed for each chromosome. DNA consists of two complementary strings which can be thought as words in the alphabet $\{A, C, G, T\}$, where A is the complement to G and C is the complement to T. The total length of all DNA's ranges from $7 \cdot 10^5$ to $2 \cdot 10^1 1$. The genetic information is passed to offsprings in two stages. First reproductive cells, *gametes,* are formed by *crossing-over* the two DNA's from the pairs of the chromosomes. Crossing-over means to put the two strings side by side, cut them at some places and replace the segments. Gametes have only one chromosome from each pair. When two organism mate, two gametes join into one cell, *zygote.* The zygote then develops into a full organism by replicating. Thus for each pair of the chromosomes of the offspring, one chromosome inherits genetic information from the mother and one from the father.

We shall consider a simplified situation where there is only one stage with an operation possibly more complex than crossing over. The two stage process may influence only the selective forces, but it is not essential for spreading information through the population. We shall also consider just one string of the genetic code. We have given more detailed, though still very rough, description above so that the reader can later see how much do we simplify the matters.

The evolution of a population of a species is further determined by the particular way of *mating* and by *fitness,* the ability to survive and reproduce, of organisms with particular genetic code. The usual simplifying assumption is *panmixia* i.e. the mating organisms meet completely randomly.

The basic concepts of mathematical population genetics are the following. Let G denote the possible forms of organisms of a given species; we can just take the possible genetic codes. A *population* is a mapping $z : G \to [0, 1]$. For a $g \in G$, $z(g)$ denotes the frequency of g in the population; thus we require

$$\sum_g z(g) = 1. \tag{1}$$

This means that we ignore the size of the population; in fact allowing real numbers as frequencies means that we assume that it is infinite. Evolution is determined by *inheritance coefficients* $p(g, h; k)$, the probability that g and h produce k, and *survival coefficients* $\lambda(g)$, the probability that g survives. In order to preserve (1), we assume

$$\begin{array}{l} p(g, h; k) \geq 0, \\ \sum_k p(g, h; k) = 1. \end{array} \tag{2}$$

For $\lambda(g)$ we require only $0 \leq \lambda(g) \leq 1$. The inheritance coefficients determine a quadratic operator on $[0, 1]^G$ given by the following equation:

$$z'(k) = \sum_{g,h} p(g, h; k) z(g) z(h). \tag{3}$$

We shall call it *inheritance operator.* The survival coefficients determine the following *survival* operator:

$$z'(g) = \frac{\lambda(g) z(g)}{\sum_h \lambda(h) z(h)}, \tag{4}$$

(additional conditions must be ensured so that $\sum_h \lambda(h) z(h)$ is never 0). The binary operator V obtained as the composition of these two is called the *evolutionary operator.* The population evolves in discrete steps by applying the evolutionary operator V to the initial vector z. We shall not talk about more complex models such as those with two sexes, two levels (zygote and gamete) etc.

In this paper we want to study computability aspects of evolution. Therefore we represent G by a set of strings A^m of length m in a finite alphabet A. It is natural to require that the offsprings of g and h are computed by a probabilistic Turing Machine P in polynomial (in m) time. Thus the inheritance coefficients are given by

$$p(g, h; k) = \text{Prob}[P(g, h) = k].$$

Formally, $P(g,h)$ denotes the random variable obtained by running P on the input gh, where we denote by gh the concatenation of the words g and h. Similarly, the survival coefficients are determined by a random variable $\Lambda : A^m \to \{0,1\}$ computed by a probabilistic Turing Machine:

$$\lambda(g) = \mathrm{Prob}[\Lambda(g) = 1].$$

We shall use the following notation. We shall think of m as the set $\{0,\ldots,m-1\}$. For a subset $T \subseteq \{0,\ldots,m-1\}$, we denote by $p|_T$ the restriction of p to A^T,

$$p|_T(h) = \sum_{g|_T = h} p(g).$$

We shall say that g and h *do not interact* if

$$p(g,h;g) = 1/2 \text{ and } p(g,h;h) = 1/2.$$

It is convenient to use nonsymmetric p, but note that almost all results remain true for the symmetric case, since we can symmetrize it very easily by taking $(p(g,h;k) + p(h,g;k))/2$. Note, however, that this requires an additional random bits (to choose the order of g and h.)

Sometimes we shall also need "empty" strings, then we shall assume that there is a special symbol # in the alphabet A and we denote by $\vec{\#}$ the string consisting of #'s.

3 Environment as the input

We want to show that the information provided by the survival operator can be coded into g's, thus it suffice to consider only the inheritance operator, if we are interested in the computational power of such a model. More precisely, we want to show that if the machine Λ has an additional input with a word x of length $n \leq m$, then it can force the population to develop so that x appears as an initial segment on almost all g's.

In the above setting the solution is trivial: take

$$\Lambda(g,x) = \begin{cases} 1 & \text{if } g|_n = x \\ 0 & \text{otherwise} \end{cases}$$

Then the condition is satisfied already in the next generation, provided that all g's have positive frequencies.

This solution has very little to do with reality, therefore, exceptionally, we shall consider a situation with a small population, where small means again polynomial in n. As we do not want to introduce another model, we will work with the one above and only ensure that those g's which are essential will have frequency $z(g)$ at least $1/n^k$ for some constant k. Then it can be shown that the evolution of an actually finite population of size say n^{2k} would be with high precision and high probability the same. We shall state the result for finite populations, but prove only the infinite approximation as stated above.

Proposition 3.1 *Let $m \geq 2n$. There exist polynomials $t_1(n)$ and $t_2(n)$ and random variables $P(g)$ and $\Lambda(g,x)$ computable in probabilistic polynomial time such that for every $x \in A^n$, the population of size $t_1(n)$ consisting only of $\vec{\#}$'s (i.e. $z(\vec{\#}) = 1$) evolves in $t_2(n)$ generations into a population where all members have x as the initial part (i.e. $z|_n(x) = 1$) with exponential probability.*

Proof. Let $A = \{0,1\}$. Let

$$P(g,h) = \begin{cases} g & \text{with probability } 1/2 - \varepsilon \\ h & \text{with probability } 1/2 - \varepsilon \\ g & \text{with one of the first } n \text{ bits changed} \\ & \text{with probability } \varepsilon \\ h & \text{with one of the first } n \text{ bits changed} \\ & \text{with probability } \varepsilon \end{cases}$$

Thus a particular bit is changed with probability ε/n in g and the same in h. We shall set $\varepsilon = 1/16n$. Let

$$\Lambda(g,x) = \frac{1}{n} \cdot \text{the number of positions on which } g|_n \text{ and } x \text{ coincide}$$

Consider some population z. Let a_i denote the frequency of g's which coincide with x on exactly i places. Let

$$a = \sum_{i=k+1}^{n} a_i, \quad b = \sum_{i=0}^{k} a_i.$$

Let a' be a after applying the inheritance operator. Then

$$a' \geq 2a^2(\frac{1}{2} - \varepsilon) + 2ab(\frac{1}{2} - \varepsilon) = a(1 - 2\varepsilon).$$

Now let a' be a after applying the survival operator. Then

$$a' = \frac{\sum_{i=k+1}^{n} \frac{i}{n} a_i}{\sum \frac{i}{n} a_i} = \frac{\sum_{i=k+1}^{n} \frac{i}{n} a_i}{\sum_{i=0}^{k} \frac{i}{n} a_i + \sum_{i=k+1}^{n} \frac{i}{n} a_i} \geq$$

$$\geq \frac{\frac{k+1}{n} a}{\frac{k}{n} b + \frac{k+1}{n} a} = \frac{(1+\frac{1}{k})a}{b + (1+\frac{1}{k})a} \geq \frac{(1+\frac{1}{n})a}{b + (1+\frac{1}{n})a} = \frac{(1+\frac{1}{n})a}{1 + \frac{1}{n}a}.$$

If $a \leq 3/4$ then it is

$$\geq a\frac{1+\frac{1}{n}}{1+\frac{3}{4n}} \geq a(1+\frac{1}{4n}).$$

Assuming $a \leq 3/4$ and substituting $\varepsilon = 1/16n$ we get after application of both operators

$$a' \geq a(1-2\varepsilon)(1+\frac{1}{4n}) \geq 1+\frac{1}{16n}.$$

Thus after $O(n)$ generations a increases to at least $\max\{3/4, 2a\}$. Furthermore, if $\sum_{i=k}^{n} a_i \geq 3/4$, then after applying the inheritance operator $\sum_{i=k+1}^{n} a_i$ becomes at least

$$\left(\frac{3}{4}\right)^2 \cdot 2\varepsilon \cdot \frac{1}{n} = \frac{9}{128n^2},$$

which, by the above estimate, increases to $3/4$ after $O(n\log n)$ generations. Thus after $O(n^2\log n)$ generations the frequency a_n becomes at least $3/4$.

We shall sketch how to increase this frequency to 1 minus an exponentially small term. Clearly the strategy must change after some time, otherwise the "mutations" prevent us to get such a good bound. To this end the machine for P can use the rest of the strings to keep track of the time. So after sufficiently long time it stops the mutations. Then it must determine which is the string x. This is possible due to Lemma 3.2 below. It enables P to compute the most frequent string at the beginning of g's. Then it just sets $P(g,h) = g$ with probability 1, if $g|_n = x$. This causes the other elements to disappear exponentially fast. ■

Another way to get information from environment is to compute the frequency of certain parts of the code. We shall present two such computations. The first one is the lemma that we needed in the proof above.

Lemma 3.2 *Let $m \geq n + 2\log n + 1$. Let $z : A^m \to [0,1]$ be a population such that the first n bits are separated from the rest by a special symbol and the next $\lceil 2\log n\rceil$ bits are 0, (this means that those g's which do not have this property have frequency 0). Then there exists an inheritance operator computed by a probabilistic polynomial time bounded Turing Machine P with the following properties:*

1. *the frequencies $z|_n$ are preserved;*
2. *after n generations, for almost all $g \in A^m$ the $\lceil\log n\rceil$ bits after the first n and the separation symbol encode the frequency $z|_n$ with precision 1/4; more precisely the frequency of g's for which the next $\lceil\log n\rceil$ bits do not encode the number $z|_n(g|_n)$ with precision 1/4 is at most $2\mathrm{e}^{-n/8}$.*

Proof. As we are not interested in the rest of the sequence g after $n+2\log n+1$ bits, we can think of each g as a triple (h,i,j), where we interpret the two parts of length $\lceil\log n\rceil$ as numbers. Initially $i = j = 0$. The machine P will produce outputs with the following probabilities.

$$P((h,i,j),(h',i',j')) =$$

$$\begin{cases} (h,i,j+1) & \text{with probability } 1/2 \quad \text{if } h \neq h' \\ (h',i',j'+1) & \text{with probability } 1/2 \quad \text{if } h \neq h' \\ (h,i+1,j+1) & \text{with probability } 1/2 \quad \text{if } h = h' \\ (h',i'+1,j'+1) & \text{with probability } 1/2 \quad \text{if } h = h' \end{cases}$$

Clearly j encodes the number of generations that have passed. Let h be fixed. It can be easily checked that the i associated with h in the j-th generation has binomial distribution

$$\binom{j}{i}\alpha^i(1-\alpha)^{j-i},$$

where $\alpha = z|_n(h)$, is the frequency of h. Thus for $j = n$, the frequency of those which have

$$\left|\frac{i}{n} - \alpha\right| \geq 1/4$$

is, by Chernoff's bound, at most

$$2\mathrm{e}^{-2\left(\frac{1}{4}\right)^2 n} = 2\mathrm{e}^{-n/8}.$$

Thus we only need to modify P so that it prints i/n instead of i in the n-th generation. ■

The next proposition shows that we can get very high precision, if we want to compute the frequency only for one bit.

Proposition 3.3 *Suppose we have a population z where 1 occurs on the first position with the frequency α (i.e. $\alpha = z|_{\{1\}}$). Then there exists an inheritance operator computed by a probabilistic polynomial time bounded Turing Machine P with the following properties:*

1. *the frequencies of the first bit are preserved;*
2. *after n generations, for almost all $g \in A^m$ the n bits after the first one encode the frequency $z|_{\{1\}}$ with exponential precision; more precisely the frequency of g's for which the next n bits do not encode the number α with precision λ is at most $2\mathrm{e}^{-\lambda^2 2^{n+1}}$.*

Thus for instance the precision $2^{-n/3}$ is achieved for the $1 - 2e^{-2^{n/3+1}}$ fraction of the population.

Proof. As in the above proof we can think of each g as a pair (b, y), a bit b and a number y with binary representation of length n. W.l.o.g. we can assume that initially $y = 0$. The machine P will give the following probabilities:

$$P((b,y)(b',y')) = \begin{cases} (b, y+y') & \text{with probability } 1/2 \\ (b', y+y') & \text{with probability } 1/2 \end{cases}$$

Consider the distribution of y's in the i-th generation. It can be easily computed that it has binomial distribution of order 2^{-i}. Thus, by Chernoff's bound, we get that for $i = n$ the frequency of (b, y)'s for which $|y - \alpha| \geq \lambda$ is

$$\leq 2e^{-2\lambda^2 2^n} = 2e^{-\lambda^2 2^{n+1}}.$$

■

4 Genetic Turing Machines

In this section we shall introduce our basic computational model and compare it with some related models.

A *Genetic Turing Machine* P is specified by a finite alphabet A and a Probabilistic Turing Machine P which has the property that for each m, it produces output strings of length m from input strings of length $2m$, (more precisely it produces a probability distribution on strings of length m). The strings $g \in A^m$ will be called *tapes.*

A Genetic Turing Machine determines evolution of a distribution (population) $z : A^M \to [0, 1]$ in the sense discussed in the previous section. We want, however, to compute on input strings, rather than distributions. Let us assume that $A = \{0, 1, \#\}$. For an input string $x \in \{0, 1\}^n$ we shall take the initial population z consisting solely of strings of the form $x\vec{\#} \in A^m$ (i.e. $z(x\vec{\#}) = 1$) and assume that m is sufficiently large. To simplify the matter, we shall assume that after computing for some time the machine will stop on all pairs with nonzero frequency. This is an inessential restriction in the most cases, since the machine can use a part of the additional space on tapes to keep track of time and stop after sufficiently long time has passed. The output is a probability distribution on strings $y \in \{0, 1\}^n$, which are initial segments of the tapes delimited by # .

Hence a Genetic Turing Machine computes the same thing (a probability distribution) as a Probabilistic Turing Machine. Thus we can use the same criterions for defining classes accepted by Genetic Turing Machines. In particular we define that P is a *bounded error* machine, if in the final population the frequency of 1's on the first position is either at least 3/4 or at most 1/4 (i.e. $z|_{\{1\}}(1) \geq 3/4$ or $z|_{\{1\}}(1) \leq 1/4$). We define that a bounded error Genetic Turing Machine accepts the set of the strings for which in the final population $z|_{\{1\}}(1) \geq 3/4$.

We shall show that it is possible to simulate general Genetic Turing Machines by machines of a very special form with only a polynomial increase of time. (A further reduction will be considered in Section 6.)

Assume that P uses an alphabet A and P' uses the alphabet $A' = A \cup \{\#\}$. We shall say that P' *simulates* P *in time* $t(m)$ *and space* $s(m)$, if the following holds for every m and every population $z : A^m \to [0, 1]$. Let $z' : A^{m'} \to [0, 1]$ with $m' = s(m)$, be such that

$$z'(g\vec{\#}) = z(g) \tag{5}$$

for every $g \in A^m$. Let $z^{(i)}$, resp. $z'^{(i)}$, denote z, resp. z', after i applications of the inheritance operator associated with P, resp. P'. Then the condition is that

$$z'^{(it)}|_m = z^{(i)}. \tag{6}$$

We shall say that P' *polynomially simulates* P, if P' simulates P in polynomial time and polynomial space.

We shall use the following notation. For $g, h \in A^m$, we denote by $\langle g, h\rangle$ the string of length m in alphabet $A \times A$ defined by $\langle g, h\rangle_i = (g_i, h_i)$. Let $F : (A \times A)^m \to (A \times A)^m$ be a function. Then we can interpret F as a random variable $F : (A \times A)^m \to A^m$ as follows. Suppose $F(\langle g, h\rangle) = \langle g', h'\rangle$, then we think of F as

$$F(g,h) = \begin{cases} g' & \text{with probability } 1/2 \\ h' & \text{with probability } 1/2 \end{cases}$$

An inheritance operator thus given by F represents the situation where parents have always two children.

In the following proposition we shall use an operator which is not symmetric; we can make it symmetric, but then we get only a weaker property: each parents have four children.

Proposition 4.1 *Any Genetic Turing Machine P can be polynomially simulated by a Genetic Turing Machine P' where*

1. *$P' : (A \times A)^m \to (A \times A)^m$ with the interpretation above and*

2. *P' is computable by a deterministic Turing Machine in linear time.*

Proof. Let the Genetic Turing Machine be given by an alphabet A and a Probabilistic Turing Machine P, let m be given. Let m' be the space needed by P working on inputs of length $2m$, more precisely the length of strings needed to encode such configurations of P. W.l.o.g. we can assume that P works as follows:

- on inputs $gh\vec{\#}$ it produces $kk\vec{\#}$ instead of just $k\vec{\#}$,
- the string $kk\vec{\#}$ is different from the strings that code configurations of P working on such inputs,
- on each input of size $2m$ it always stops after exactly t steps, t bounded by a polynomial.

Furthermore we shall assume that the initial population is of the form $gg\vec{\#}$ instead of $g\vec{\#}$. Recall that P uses one random bit in each computation step.

Define P' by

$$P'(\langle gg\vec{\#}, hh\vec{\#}\rangle) = \langle ghw, hgw'\rangle$$

where ghw resp. hgw' encode the initial configuration of P on gh resp. hg;

$$P'(\langle g_1h_1w_1, g_2h_2w_2\rangle) = \langle g_1'h_1'w_1', g_2'h_2'w_2'\rangle$$

where $g_1h_1w_1$ is a configuration (not final) and $g_1'h_1'w_1'$ is the next configuration corresponding to the random bit 0 and where $g_2h_2w_2$ is a configuration (not final) and $g_2'h_2'w_2'$ is the next configuration corresponding to the random bit 1;

$$P'(\langle ggw, hhw'\rangle) = \langle gg\vec{\#}, hh\vec{\#}\rangle$$

where ggw resp. hhw' encode end configurations of P.

For all other inputs P' can be defined arbitrarily. The simulation proceeds as follows. First the tapes $gg\vec{\#}$ and $hh\vec{\#}$ are randomly mixed into $gh\vec{\#}$ and $hg\vec{\#}$ by crossing-over and the computation of P on them starts. Then for t generations P' works as the linear operator of the probabilistic machine, except that it is always performed on pairs. Note that each configuration ghw occurs in half of the cases as the first component and in the other half as the second component. Thus the two next configuration corresponding to the two values of the random bit, will be produced with weight 1/2 each. In the $t+1$-st generation P' transforms the end configuration ggw of P into $gg\vec{\#}$. (The "end configuration" means that P has completed the computation of $P(g,h)$, not that the genetic computation stops.) Then the process is repeated with the new population of $gg\vec{\#}$'s etc. Thus the original Genetic Turing Machine is simulated in time $t+1$. ■

Essentially the same idea can be used to prove the next two propositions.

There is a natural generalization of the Genetic Turing Machine to the case where the quadratic operator is replaced by an operator of degree $d > 2$. However essentially no additional power is gained.

Proposition 4.2 *The generalized Genetic Turing Machines with operators of a fixed degree $d > 2$ can be simulated by the usual ones in polynomial time.*

Proof-sketch. Let d be given. Use the same proof as in Proposition 4.1, except that you must take the tapes of the form $gg\ldots g\vec{\#}$ with d occurrences of g and the "mixing phase" must be $d-1$ steps. ■

A universal Turing Machine is a machine M such that for any Turing Machine M' there exist a string c (a "program" for M') such that for each input x, M computes the same output on $c\#x$ as M' on x. It is well known that universal Turing Machines exist and that they actually simulate in polynomial time. The same is true about universal Probabilistic Turing Machines (recall that they output a probability distribution). These in turn can be easily used to construct universal Genetic Turing Machines.

Proposition 4.3 *There exists a universal Genetic Turing Machine which simulates other Genetic Turing Machines in polynomial time and space.*

Proof-sketch. Apply the proof of Proposition 4.1 to a universal Probabilistic Turing Machine with the tapes of the form $c\#gg\vec{\#}$, where c is reserved for programs. ■

Let us now compare Genetic Turing Machines with Probabilistic Turing Machines and Quantum Turing Machines. Consider a Probabilistic Turing Machine M computing on inputs of size n. Suppose the machine always uses some restricted space and hence the computations can be coded by strings in A^m, for some m. Then we can think of the computation of M as evolution of $z : A^m \to [0,1]$ given by the random variable $P : A^m \to A^m$ defined by

$$P(g) = \begin{cases} h_0 & \text{with probability } 1/2 \\ h_1 & \text{with probability } 1/2 \end{cases}$$

where h_0, h_1 are the next possible configurations after g. Thus, taking

$$p(g;h) = \text{Prob}[P(g) = h],$$

the evolution of z is defined by

$$z'(h) = \sum_g p(g;h)z(g). \tag{7}$$

Hence the essential difference is that this operator is *linear,* while in Genetic Turing Machines it is *quadratic.* (For Probabilistic Turing Machines the random variable P is, moreover, given by simple rewriting rules; it is not clear if Genetic Turing Machines can use such rules, we shall use crossing-over in Section 6.) Let us observe that the conditions (2) correspond to the following ones for Probabilistic Turing Machines.

$$\begin{array}{l} p(g;k) \geq 0, \\ \sum_k p(g;k) = 1. \end{array} \tag{8}$$

There is another, more exotic, but physically well-founded concept of a Turing Machine which has a linear operator. It is the *Quantum Turing Machine* introduced by Deutch. It works in a similar way as Probabilistic Turing Machine, but the coefficients $p(g;h)$ can now be arbitrary *complex* numbers. The vector z represents only so called *amplitudes* and the probability is counted as its L_2 norm. The transformation (7) must preserve the L_2 norm, hence the matrix $\{p(g;h)\}_{g,h}$ must be unitary. This means that the requirement (2) is replaced by

$$\sum_g z(g)\overline{z(g)} = 1,$$

and (8) is replaced by

$$\begin{array}{l} \sum_h p(g;h)\overline{p(k;h)} = 0 \text{ for } g \neq k, \\ \sum_h p(g;h)\overline{p(g;h)} = 1. \end{array}$$

5 The power of genetic computations

In order to compare the computational power of Genetic Turing Machines with the classical ones we shall consider the complexity class of sets accepted by bounded error Genetic Turing Machines in polynomial time and show that it is equal to $\mathcal{PSPACE}$.

There is a natural concept of *space* and *time* in Genetic Turing Machine computations. The time is the number of generations. The space is the length of the tape. Note that we require that the computation of offsprings $P(g,h)$ is done by a Probabilistic Turing Machine whose time is bounded by a polynomial $t(n)$ depending on the input length n. Thus this measure is not very precise, as a lot of computation can be done between the generations. A more precise measure is obtained using Proposition 4.1 as the definition. Here we are interested only in polynomial bounds, thus we do not need the more precise measure.

We want to characterize the natural class of sets accepted by bounded error Genetic Turing Machines in polynomial time. Observe that the space can be bounded by a polynomial depending on time, similarly as for Turing Machines, hence such machines have also polynomially bounded tape. As Russel Impagliazzo pointed out, the usual simulations techniques can be used to prove that this class is contained in $\mathcal{PSPACE}$. Thus we get a full characterization from the following theorem.

Theorem 5.1 *Polynomial space on Turing Machines can be simulated by polynomial time on Genetic Turing Machines, i.e. $\mathcal{PSPACE}$ is contained in the class of sets accepted by bounded error Genetic Turing Machines in polynomial time.*

Proof. Let $L \in \mathcal{PSPACE}$. Let M be a Turing Machine accepting L running in polynomial space. Thus for a given input length n the configurations of M can be coded as strings of 0's and 1's of length N, where N is bounded by a polynomial depending on n. In particular the machine can run only for 2^N steps. We shall assume that it remains in the final configuration when it reaches such, thus we only need to determine, if its configuration after 2^N steps is an accepting configuration. We shall say that a configuration w_2 is k steps after configuration w_1, if it is the k-th next configuration after w_1.

Let a sufficiently large n, and hence N, be fixed. We shall describe the action of a bounded error Genetic Turing Machine P for the set L. The tape of P will encode (x, b, i, w_1, w_2) where

- x is the input,
- b will be the output bit,
- i is a number and
- w_1, w_2 are 0-1 strings of length N, which will encode configurations.

The initial population will be $(x, 0, 0, \vec{\#}, \vec{\#})$. The machine P will work as follows:

1. On an input pair $(x, 0, 0, \vec{\#}, \vec{\#}), (x, 0, 0, \vec{\#}, \vec{\#})$ it generates a random string w_1 of length N, each string with probability 2^{-N}. Then it checks, if w_1

is a configuration of M. If so, then it computes the configuration w_2 which is next after w_1 and produces $(x, 0, 1, w_1, w_2)$ as the output. Otherwise it produces $(x, 0, 0, \vec{0}, \vec{0})$.

2. On an input pair $(x, 0, i, w_1, w_2), (x, 0, i, w_2, w_3)$, where $i < N$, it produces $(x, b, i+1, w_1, w_3)$ where $b = 1$, if w_1 is the initial configuration and w_3 is an accepting configuration of M working on x, and $b = 0$ otherwise.

3. On an input pair $(x, 0, i, w_1, w_2), (x, 0, j, w_1', w_2')$ it will output $(x, 0, i, w_1, w_2)$, if $i > j$, and $(x, 0, j, w_1', w_2')$, if $i < j$. If $i = j$ and $w_2 \neq w_1'$, then they will not interact.

4. On an input pair $(x, 1, i, w_1, w_2), (x, b', i', w_1', w_2')$ it will produce $(x, 1, i, w_1, w_2)$.

In all other cases the pairs do not interact.

It is clear, how the evolution from the initial population will look like. First tapes of the form $(x, 0, i, w_1, w_2)$ with w_2 one step after w_1 are created and the rest becomes $(x, 0, 0, \vec{0}, \vec{0})$. Tapes of the form $(x, 0, i, w_1, w_2)$ will gradually appear where w_2 is the configuration 2^i steps after the configuration w_1. Those with larger i will win over those with smaller i, so the average i will increase. The tapes $(x, 0, 0, \vec{0}, \vec{0})$, which do not code anything, do not produce new tapes and quickly disappear. Eventually a large part will have $i = N$, which, in particular, means that the finial configuration of M has been reached. If M accepts x, then $b = 1$ on these tapes. Then the tapes with $b = 1$ will increase their frequency, eventually over $3/4$. If M does not accept x, then tapes with $b = 1$ never appear. We have to prove that in the positive case the frequency $3/4$ is reached in polynomial time.

Claim 1. *Consider a particular generation in the evolution and let $1 \leq i \leq N$ be fixed. Then the frequencies of all $(x, 0, i, w_1, w_2)$, where w_2 is the configuration 2^i steps after a configuration w_1, have the same value.*

We shall prove it by induction on the generations. In the first generation all $(x, 0, i, w_1, w_2)$ with $i = 1$ have frequency 2^{-N} and for $i > 1$ their frequency is 0. The property is preserved to the next generation, because each such $(x, 0, i, w_1, w_2)$ can be produced in a unique way from tapes of the form $(x, 0, i-1, w_1', w_2')$. This is because M is deterministic and thus if w_2 is 2^i steps after w_1, $i > 1$, there exists exactly one w such that 2^{i-1} is steps after w_1, and w_2 is 2^{i-1} steps after w. Hence there exists exactly one pair of tapes which can produce $(x, 0, i, w_1, w_2)$. Consequently the new frequency of $(x, 0, i, w_1, w_2)$ is a function of the old frequency of $(x, 0, i, w_1, w_2)$ and the old frequencies of the corresponding pairs. These are the same for all such tapes by the induction assumption.

Let

$$K = 2(N + \lceil \log_2 N \rceil + 2) + 1.$$

We shall estimate the frequencies of tapes $(x, 0, i, w_1, w_2)$ in particular generations.

Claim 2. *Consider the aK-th generation, for some a, $1 \leq a \leq N$. Then either the sum of the frequencies of tapes $(x, 0, i, w_1, w_2)$ with $i \geq a$ is at least 1/4, or the sum of the frequencies of tapes $(x, 1, i, w_1, w_2)$ is at least 1/4.*

Again we proceed by induction.

Let $a = 1$. Since there exists a computation even of length 2^N, there is at least one pair w_1, w_2, where w_2 is one step after w_1. Hence in the first generation the frequency of tapes with $i = 0$ is at most $1 - 2^{-N}$. Due to rules 3 and 4, this decreases after N steps to $(1-2^{-N})^{2^{N+1}}$, which is less than $1/4$ for N sufficiently large.

Suppose the claim holds for some $a < N$. If the sum of the frequencies of tapes $(x, 1, i, w_1, w_2)$ is at least $1/4$ in the aK-th generation, then it is at least so in the $(a+1)K$-th generation. Thus suppose that the frequency of tapes with $i \geq a$ is at least $1/4$ in the aK-th generation. Hence for some $i_0 \geq a$ the frequency of tapes with $i = i_0$ is at least $1/4N$.

First suppose that $i_0 < N$. Again, there exists at least one pair w_1, w_2, where w_2 is 2^{i_0+1} steps after w_1. Let w be "between" w_2 and w_1, i.e. w is 2^{i_0} steps after w_1 and w_2 is 2^{i_0} steps after w. Then, by Claim 1, the frequencies of $(x, 0, i_0, w_1, w)$ and $(x, 0, i_0, w, w_2)$ are at least $1/4N2^N$, hence $(x, 0, i_0 + 1, w_1, w_2)$ or $(x, 1, i_0 + 1, w_1, w_2)$ has the frequency at least

$$\left(\frac{1}{4N2^N}\right)^2 = 2^{-(K-1)}$$

in the $aK + 1$-st generation. Thus the sum of the frequencies of tapes $(x, 0, i, w_1, w_2)$ with $i \leq a$ is at most $1 - 2^{-(K-1)}$. Due to rules 3 and 4 it decreases to

$$\left(1 - 2^{-(K-1)}\right)^{2^{K-1}} \leq \frac{1}{2}$$

after the next $K - 1$ generations.

If $i_0 = N$, then, since $a < N$, the frequency of tapes $(x, 0, i, w_1, w_2)$ with $i \leq a$ is at most $1 - 1/4N$ and this decreases to a value less than $1/2$ even sooner.

Thus the claim is proved.

Applying Claim 2 to $a = N$ we get that in the NK-th generation the sum of the frequencies of tapes

with $i = N$ or $b = 1$ is at least $1/4$. If the frequency of tapes with $b = 1$ is less than $1/4$, then, by Claim 1, the frequency of the tape (x, b, i, w_1, w_2), where w_1 encode the initial configuration of M on x and w_2 encode the end configuration of M on x, is at least $1/2^{N+2}$. If M accepts x, then this $b = 1$, hence this frequency is amplified to at least $3/4$ after $N+3$ generations. Thus if M accepts x, then in any case the frequency of this tape $(x, 1, i, w_1, w_2)$ will be at least $3/4$ after $N + 3$ generations. If M does not accept x, then $b = 1$ never appears.

Thus we can conclude that the initial population evolves so that after $O(n^2)$ generations the sum of the frequencies of tapes with $b = 1$ is at least $3/4$, if M accepts x, and it is 0 otherwise. ■

6 Reduction to crossing-over

Here we show that a general Genetic Turing Machine can be simulated by a Genetic Turing Machine which uses only crossing over where positions at which the crossing over is done is determined only by a small neighborhood of it. (Let us note that the word *recombination* is often used instead of *crossing over.*)

Let us be more precise. Let C be a set of quadruples of finite strings in alphabet A. We shall call C *local conditions* and assume that it is a finite set. Let $g, h \in A^m$. *Crossing over according to the set of conditions* C is the following procedure. Starting from the left side, consider the homologous positions in the strings g and h. If the part before the position ends with u_1 in g and with v_1 in h, and the part after the position starts with u_2 in g and with v_2 in h for some $(u_1, v_1, u_2, v_2) \in C$, then we switch the whole parts after the position and we move to the next position left. Otherwise we just move to the next position to the left. Thus if

$$\begin{array}{cccc|ccc} g_1g_2 & \dots & \dots & g_p \dots g_i & g_{i+1}g_{i+2} \dots g_r & \dots & \dots & g_m \\ h_1h_2 & \dots & \dots & h_q \dots h_i & h_{i+1}h_{i+2} \dots h_s & \dots & \dots & h_m \end{array}$$

and

$$(g_p \dots g_i, h_q \dots h_i, g_{i+1} \dots g_r, h_{i+1} \dots h_s) \in C,$$

then we get

$$\begin{array}{cccc|ccc} g_1g_2 & \dots & \dots & g_p \dots g_ih_{i+1} & h_{i+2} \dots h_s & \dots & \dots & h_m \\ h_1h_2 & \dots & \dots & h_q \dots h_ig_{i+1} & g_{i+2} \dots g_r & \dots & \dots & g_m \end{array}$$

Otherwise we get

$$\begin{array}{cccc|ccc} g_1g_2 & \dots & \dots & g_p \dots g_ig_{i+1} & g_{i+2} \dots g_r & \dots & \dots & g_m \\ h_1h_2 & \dots & \dots & h_q \dots h_ih_{i+1} & h_{i+2} \dots h_s & \dots & \dots & h_m \end{array}$$

We shall furthermore assume that we can use information about the beginning and the end of the string. E.g. when applying crossing-over we can assume that the words always start and end with a special symbol.

The crossing-over will be interpreted in the same sense as in Proposition 4.1, i.e. the inheritance coefficients will be $p(g, h; k_1) = 1/2$ and $p(g, h; k_2) = 1/2$ where k_1 and k_2 are the two strings resulting from crossing over g and h. Note that again this is not a symmetric operator.

It is clear that crossing over in Nature must allow some randomness, which is not present in the definition above. If a deterministic procedure as above was used, then a couple of parents would have always all children identical. The definition above can be generalized to something which is closer to reality. Namely, instead of taking *a set* of quadruples we can take *a probability distribution* on all quadruples of words of a certain length. Then the algorithm of crossing over described above would be the same, except that we would switch the strings at a particular site with some probability $0 \leq p \leq 1$ depending on the context. We shall prove the simulation for the more restricted case of deterministic crossing over. The reason is only to get a stronger mathematical result. In fact, a part of the simulation could be simplified, if we allowed randomness in crossing over.

Because of the very restricted means, we have to consider a weaker concept of simulation than we used in Section 4. Namely:

1. The simulated tapes g cannot be simply initial parts of the simulating tapes g'; instead we shall assume that g is represented by

$$a_1g_1a_2g_2 \dots a_mg_ma_{m+1}0a_{m+2}0 \dots a_{m'}0,$$

where $a_1, \dots, a_{m'}$ are strings of some fixed constant length. We can assume that they are all the same in the initial population, except for a_{m+1}, which determines the end of g.

2. Furthermore we need some *auxiliary tapes* which will not encode the original tapes at all and which will be used only for rewriting. Those which do code the original ones will be called *proper.* The population will consist of proper and auxiliary tapes. We shall ensure that the frequency of proper tapes is a fixed positive number.

3. We shall simulate the original frequencies not precisely, but up to some error ε. Let us denote the original population by z, the set of indices on which it is simulated by S and the restriction of the simulating population z' to S and to proper tapes by $z'|_{S,proper}$.

Then the equation (6), Section 4, is replaced by

$$| z'^{(it)}|_{S,proper} - z^{(i)} | \leq \varepsilon.$$

4. Finally we will not be able to keep the error small for ever, thus we shall consider only simulation of polynomially many generations.

In order to state the next theorem concisely, let us say that P can be *weakly polynomially simulated* by a class of Genetic Turing Machines $\mathcal{X}$, if for every polynomial q, there exists a Genetic Turing Machine $P' \in \mathcal{X}$ which simulates P on A^m in the above sense for $q(m)$ generations, in polynomial time, polynomial space and with error $\varepsilon \leq q(m)^{-1}$.

Theorem 6.1 *Every Genetic Turing Machine can be weakly polynomially simulated by a Genetic Turing Machine with crossing over determined by a finite set of local conditions.*

By the following lemma, we need to simulate just one step with sufficiently large precision.

Lemma 6.2 *Let V_1, V_2 be two inheritance operators determined by inheritance coefficients $p_1(g,h;k)$ resp. $p_2(g,h;k)$. Let l be a positive integer. Suppose that for every $g,h,k \in G$,*

$$|p_1(g,h;k) - p_2(g,h;k)| < 2^{-n^{l+1}},$$

and $|G| \leq 2^{n^l}$. Apply V_1, V_2 to the same population z n^l-times. Let z_1 resp. z_2 be the corresponding populations. Then

$$\sum_g |z_1(g) - z_2(g)|,$$

hence also the maximal distance, will still be exponentially small.

Proof. Suppose that

$$\max_{g,h,k} |p_1(g,h;k) - p_2(g,h;k)| \leq \delta.$$

Consider two populations such that

$$\sum_g |z_1(g) - z_2(g)| \leq \varepsilon.$$

Let z_1', z_2' be the populations in the next generation. Then

$$\sum_k |z_1'(k) - z_2'(k)| = \sum_k \left| \sum_{gh} p_1 z_1 z_1 - p_2 z_2 z_2 \right| =$$

$$= \sum_k \Big| \sum_{gh} p_1 z_1 z_1 - p_2 z_1 z_1 + p_2 z_1 z_1 - p_2 z_2 z_1 + p_2 z_2 z_1 - p_2 z_2 z_2 \Big| \leq$$

$$\leq \sum_{ghk} |(p_1 - p_2) z_1 z_1| + \sum_{ghk} |(z_1 - z_2) p_2 z_1| + \sum_{ghk} |(z_1 - z_2) p_2 z_2|$$

$$\leq \sum_k \left(\delta \sum_{gh} z_1(g) z_1(h) \right) + \sum_g \left(|z_1(g) - z_2(g)| \sum_{hk} p_2(g,h;k) z_1(h) \right) + \sum_h \left(|z_1(h) - z_2(h)| \sum_{gk} p_2(g,h;k) z_2(h) \right) =$$

$$= \sum_k \delta + \sum_g \left(|z_1(g) - z_2(g)| \sum_h z_1(h) \right) + \sum_h \left(|z_1(h) - z_2(h)| \sum_g z_2(h) \right) \leq |G|\delta + 2\varepsilon.$$

(We have omitted some g,h,k for sake of readability.) Thus if we start with $z_1 = z_2$, we get the bound for the i-th generation

$$\sum_k \left| z_1^{(i)}(k) - z_2^{(i)}(k) \right| \leq 3^i \delta |G|.$$

In our case $\delta = 2^{-n^{l+1}}$, $|G| \leq 2^{n^l}$, $i = n^l$, which gives an exponentially small upper bound $3^{n^l} 2^{n^l} 2^{-n^{l+1}}$. ■

The *proof of Theorem 6.1* will be an essential extension of the proof of Proposition 4.1. In this preliminary version we only present the main ideas; the details must yet be worked out.

As in the proof of Proposition 4.1 there will be two types of simulation steps:

- Steps where ggw_1 and hhw_2 produces ghw_1 and hgw_2. We shall call it *crossing-over steps.* The simulation is straightforward here. Let us recall that we have to encode the original tapes by tapes with some auxiliary strings placed between the symbols from the original ones. Thus e.g. ggw_1 should be in fact

  $$a_1 g_1 \ldots a_m g_m a_{m+1} g_1 \ldots a_{2m} g_m a_{2m+1} w_1 \ldots a_{m'} w_{m'-m}.$$

- Steps where an action of the Probabilistic Turing Machine is simulated. We shall call it *rewriting steps.* This will be done using auxiliary tapes.

Furthermore there will be also

- random bit generating steps, and
- waiting for synchronization steps.

1. The auxiliary tapes will have form $b_1 g_1 \ldots b_{m'} g_{m'}$ where b_i-s will be always different from a_i-s, thus proper and auxiliary tapes will be distinguishable locally. Furthermore we must ensure that the information bits g_i are also locally distinguishable from the labels a_i and b_i. This is quite easy.

2. We shall ensure that on the auxiliary tapes all short sequences occur with frequencies bounded below by a positive constant. Thus one rewriting will be delayed only by a constant factor. The simplest way to do it is to allow a "random" rule for crossing-over where for some context crossing over would be done with probability 1/2. It can be done also for the "deterministic" crossing-over defined above using a more complicated schema for crossing-over auxiliary tapes.

Here is a possible way. Suppose we need to rewrite words of length up to k. We take periodic auxiliary tapes with period k and such that each has the same frequency p and the frequency of proper tapes q is much smaller than p. Thus p, q are positive constants depending only on k. If an auxiliary tape is used for rewriting, it is distorted. We will not allow rewriting which would make another change near a distorted place, thus the identity of an auxiliary tape, and hence also its frequency, will be preserved. Then we introduce a rule that two mating distorted auxiliary tapes will make all corrections on themselves which reduce the number of changed bits on both. This rule restores the original periodic tapes very efficiently. Thus the frequency of distortions will first grow, but eventually it will be balanced by the restoration force.

3. Rewriting cannot be simulated directly by crossing over, since we need to keep the labels a_i and b_i which distinguish proper and auxiliary tapes. Suppose we need to replace a part of the tape of the form $a_i g_i \ldots a_j g_j$ by $a_i g'_i \ldots a_j g'_j$. Then, naturally, one would like to use an auxiliary tape with the corresponding part $b_i g'_i \ldots b_j g'_j$. However the crossing-over algorithm first replaces only g_i by g'_i and then the context is distorted for the next replacement. A solution to this problem is to do this rewriting in two stages. We shall code one bit g_i by several ones, thus we may have codes also for pairs (g_i, g'_i). In the first stage we rewrite $a_i g_i \ldots a_j g_j$ to $a_i(g_i, g'_i) \ldots a_j(g_j, g'_j)$ (no context is lost). In the second we rewrite $a_i(g_i, g'_i) \ldots a_j(g_j, g'_j)$ to $a_i g'_i \ldots a_j g'_j$ (no context is needed).

4. We need to use a random bit for each rewriting, because the rewritings correspond to elementary actions of a Probabilistic Turing Machine. (This is different from Proposition 4.1, where the bit was provided by random order of the two tapes.) This can be done by having a special sort of auxiliary tapes which is used only for providing random bits, and thus it is not biased by ordinary rewriting. The next step of the Probabilistic Turing Machine will be simulated only when a new random bit is copied on the proper tape near the position which is to be rewritten.

5. The most difficult problem is to ensure synchronization. Rewriting using auxiliary tapes leads to a random process in which some tapes are rewritten faster than others. Also the mean time needed for rewriting may depend on the type of the tape. If we allowed the crossing-over step to be performed immediately after the tape is rewritten, those which were rewritten faster would get advantage. Therefore each proper tape will be additionally equipped with a clock which will control when crossing-over may occur. It will wait much longer than is needed for rewriting in the average, consequently only an exponentially small fraction of tapes will not be rewritten. Then it will fire a signal which will allow crossing-over. After the signal has been fired, such a proper tape will not enter new rewriting phase before it crosses-over with another one. They will have enough time to cross-over, thus again only an exponentially small fraction of tapes will not manage to do it in time.

The speed of the clock must be independent of the rewriting process. This can easily be achieved, since the clock will be placed on different cells of the tape and thus it will use different parts of the auxiliary tapes. The frequencies of combinations of symbols on these positions will be not influenced. The clock will be hidden in the labels a_i, thus the signal for starting the crossing-over steps will appear on the position between the first occurrence of a code of g and the second one, where we assume that after rewriting the tape encodes a string ggw.

6. We describe a suitable clock. Let us forget about the strings that will be between the bits coding the clock. The clock will have two versions

$$000\ldots0001000\ldots000$$

and the inverse

$$111\ldots1110111\ldots111$$

both with the same frequency. The position of 1 resp. 0 from the left determines the time. Advancing will

be done in two steps

$$000\ldots0001000\ldots000$$

$$000\ldots0001100\ldots000$$

$$000\ldots0000100\ldots000$$

and similarly for the inverse version. This way of rewriting has the advantage that the probability of rewriting depends only on the frequency of 0's and 1's on *single* positions. Since we use two complementary versions of the clock the original frequency 1/2 for each bit on the auxiliary tapes will not change. Thus the distribution of times shown on the clocks will be precisely binomial with $p = 1/4$. Now we can use Chernoff's bound. After t generations the frequency of tapes on which the clock shows a time between $t/4 - t^{3/4}$ and $t/4 + t^{3/4}$ will be at least

$$1 - 2e^{-2t^{1/2}}. \tag{9}$$

7. Let us compute what we need in order to be able to apply Lemma 6.2. In order to obtain precision $2^{-n^{k+1}}$ we set the time t for the crossing-over step not only so large that the frequency of the nonrewritten tapes is exponentially small, but also we take $t \geq n^{2k+2}$ which will ensure that the bound in (9) will be of the needed order.

We shall not present further computations in this preliminary version. Let us only note that a similar argument based on Chernoff's bound can be used to ensure sufficiently small frequency of nonrewritten tapes. ■

7 Conclusions

We have presented a new model of computation which can be considered as a generalization of randomized computations and also as a kind of parallel computations with random architecture. We have shown that it is very powerful, namely, that polynomial time computations on this model can simulate polynomial space computations on Turing Machines.

The conclusions for genetics are, of course, only speculations, as we do not know if and how the crossing-over is controlled. We have, however, shown a theoretical possibility that nontrivial information processing is possible on the molecular level, since we have proved that crossing-over alone is sufficiently strong to simulate complex computations. We have used a very restricted model in which the positions where the strings can cross-over are determined by a small neighborhood (local conditions). In reality the mechanism uses randomness and maybe more, which makes complex computations even more possible.

Let us also stress that combining Proposition 4.3 with Theorem 6.1 we get that there are local conditions for crossing-over which are *universal* in the sense that any genetic computation can be simulated by them, provided we encode a program for it on the tape. Looking at the proofs one could expect that such universal conditions must always be very complex. But I think the converse is more likely. I would even conjecture that randomly chosen local conditions are universal (if we take sufficiently long conditions with suitable probability).

We conclude with a bold speculation: The existence of universal local conditions means that it is possible that the same local conditions for crossing-over are used by all species, similarly as they all use the same genetic code for the synthesis of proteins, and that they are universal.

This suggests two direction in experimental research:

1. to look for reactions of populations to changing environment which cannot be explained by purely random mutations;

2. to estimate the probabilities of crossing-over on particular sites of the DNA.

References

[1] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer.* Proc. Roy. Soc. (London), A400, 97-117

[2] W.J. Ewens: *Mathematical Population Genetics.* Springer-Verlag, 1979.

[3] Y.I. Lyubich, *Mathematical Structures in Population Genetics.* Springer-Verlag, 1992.

[4] Y. Rabinovich, A. Sinclair, A. Wigderson, *Quadratic dynamical systems.* Proc. 33-rd IEEE Symp. FOCS 1992, 304-313.

[5] J.D. Watson, N.H. Hopkins, J.W. Roberts, J.A. Steitz, A.M. Weiner, *Molecular Biology of the Gene I, II.* Benjamin/Cummings, 1987.