Complexity theory and genetics, the computational power of crossing over

P.Pudlák
Mathematical Institute
Academy of Sciences
Prague *

April 26, 1999

Abstract

We study the computational power of systems where information is stored in independent strings and each computational step consists of exchanging information between randomly chosen pairs. The input for the system is environment which selects certain strings. To this end we introduce a population genetics model in which the operators of selection and inheritance are effectively computable (in polynomial time on probabilistic Turing machines). We show that such systems are as powerful as the usual models of parallel computations, namely they can simulate polynomial space computations in polynomially many steps. We also show that the model has the same power if the recombination rules for strings are very simple (context sensitive crossing over), which suggests that similar processes might be exploited by real organisms.

1 Introduction

There is a growing interest in genetics among researchers working in theoretical computer science. Quite a lot of research has been done on the analysis of known and designing new efficient algorithms in molecular and population genetics, see e.g. a survey by Karp [6]. More recently people got interested into computational aspects of population genetics. The main source of this interest is perhaps the widespread use of the heuristic method called *genetic algorithms*. Genetic algorithms are successfully applied in various branches, but a solid theoretical foundation is missing, though some special cases have been analyzed [12, 14]. The standard mathematical model of genetic

*Supported by grant A1019901 of the Academy of Sciences of the Czech Republic and and cooperative research grant INT-9600919/ME-103 of the U.S. National Science Foundation and the Academy of Sciences of the Czech Republic. An essential part of this research has been done while visiting Fachbereich Informatik, Universität Dortmund as a Humboldt Fellow.

like systems is based on quadratic dynamical systems. Such systems have applications not only in genetics, but also in other fields such as the theory of gases in physics and the study of random formulas in the theory of boolean functions [18, 15]. It has been shown that under certain technical conditions such systems converge to a stationary distribution [13]. Then, from the computational point of view, the basic question is the rate of convergence. Some results in this direction have been obtained also in [13]. For more specific operators, based on special forms of crossing over (uniform crossover, one-point crossover, Poisson model), concrete estimates on the convergence rate have been obtained in [11].

The aim of this paper is to study computational complexity of genetic systems. Independently a similar approach was considered by Arora, Rabani and Vazirani [2]. They show that even if the quadratic operator is efficiently computable, the evolution of the system (most likely) cannot be efficiently simulated. More precisely, they call a quadratic dynamical system *succinctly-defined* if the operator is determined by a polynomial time probabilistic Turing machine. Then they construct such an operator for which the sampling problem is $\mathcal{PSPACE}$-complete. This is equivalent to our Theorem 5.1. They prove moreover that sampling of a general quadratic dynamical system can be reduced to a *symmetric* one, (see Section 2 for the definition).

Another related paper is [3]. They show that it is algorithmically undecidable, whether some genome can ever evolve from a given one provided that certain sequences code "killing" genes and thus must be avoided in the evolution. They use some similar simulation techniques as the present paper.

In this paper our primary motivation are natural genetic systems. 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 a mechanism to produce new distribution of genotypes from a given one, depending on the environment. In this setting the species receives information about the changing environment by the selection pressure. The species reacts by changing the distribution of the genotypes and, possibly, producing new genotypes (by combining the existing ones and by mutations). Thus we think of a species as a system which processes information coming from the environment. The question is how complex this information processing can be. For instance the species can just keep a fixed mutation rate and let the selective forces to choose genomes which produce the fittest individuals. There are, however, recorded cases where the reaction to the changing environment is more complex. For instance, the mutation rate can increase under selective pressure (see [17] for a brief survey), or can change the dominance relation between alleles. It is also conceivable that the species has a stock of genes which are not expressed, but can be turned on under suitable circumstances. We want to propose that the key mechanism for such phenomena could be recombination. By recombination we mean homologous crossing over and therefore we shall preferably use the second of the two terms. We shall show that, at least theoretically, genetic systems with crossing over have very strong computational power.

Now we describe the main results of the paper. We shall introduce the usual formalism of population genetics where we add conditions that the fitness and inheritance operators are effectively computable. We naturally identify effective computability with computations which can be performed in polynomial time by probabilistic Turing machines. In Sections 2-5 we work with this general model, in Section 5 we shall consider operators based on crossing over.

To motivate the rest of the paper we show in Section 3 that in this setting individual genomes can reflect some global features of the current population. For instance the frequency of some gene can be encoded in almost all genomes with high precision, provided that the frequency of the gene is constant for polynomially many generations. (Again, polynomially many generations is used for "short time".) In this way some information on the environment can enter into almost all genomes and then can be processed.

In the rest of the paper we concentrate on the inheritance operator. We study the complexity of the evolution of a system without influence of the environment. We are not interested in classical dynamical properties, such as convergence to an equilibrium, but rather in the computational complexity of this process. In fact it seems that the additional conditions of effective computability can hardly ensure some "nice" dynamical properties, (see [2] for a discussion). In Section 4 we study some general properties of this model. This computational model naturally generalizes the probabilistic Turing machines by replacing a linear operator by a quadratic one. Therefore we propose the name *genetic Turing machine* for it. 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. 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.

In Section 5 we shall show that the power of genetic Turing machines running in polynomial time (i.e. using polynomially many generations) is equal to the power of Turing machines using polynomial space (as mentioned above, this was independently proven in [2]). This means that genetic Turing machines are a model of parallel computation, thus they are extremely powerful.

In Section 5 we show that general genetic Turing machines can be simulated by genetic Turing machines where the inheritance operator is based on a very simple manipulation on the strings, namely on *context sensitive crossing over*. Context sensitive crossing over means that recombination occurs on a particular locus depending on the base pairs in the neighbourhood of this locus. We propose this as an approximation of the actual process of crossing over which is complex and not fully understood yet. One clear discrepancy is that in our model crossing over is a deterministic process, while in nature there must be randomness involved. The simulation is fairly complicated so we have not tried to extended it to a model involving randomness, but we conjecture that it is possible. It is not clear to us how important is context sensitivity in natural systems, but clearly this phenomenon does occur sometimes. A well-known example is the function of the recombination enzyme *RecBC* of *E. coli* [19]. This enzyme is sensitive to sites called *Chi* (the nucleotide sequence 5'-GCTGGTGG-3') on which it promotes recombination. Sites where recombination occurs more frequently than elsewhere are called *recombination hot spots* and they have been identified also in the genomes of other species, ranging from viruses to mammals.

The consequence of the simulation is that already systems using only context sensitive crossing over can have a very strong computational power. Thus it could be used for a complex regulatory function in expressing genes. But even if it is actually used, it does not seem very likely that this is the main mechanism. Mutual influence of genes on their expression and independent selection on them can probably alone explain very complex behaviour (see the last section and [10] for a simple example). By no means do we expect that any of the complex structures used in the proof of Theorem 6.1 occurs in nature; these are only add hoc tools for the proof.

Finally, let us say what this paper is *not* about (as some readers of the first draft have misinterpreted it). The paper does not deal with evolution of new species, dieing out, or evolution of new genes. The term "evolution" is used here as a synonym for "the change of the frequencies of genotypes" in a relatively stable population (perhaps, it would be less confusing to use the word "adaptation"). If any kind of computation is used by a species, it can help it to survive, maybe to to improve and to develop faster. However in order to develop new genes, not to say species, it does not suffice to use homologous crossing over, one needs gene duplications and mutations. In fact, we have to admit that we do not have a mathematical model or computer simulation showing that computations give some survival advantage. On the other hand it is clear that complexity theory should play an important role in explaining phylogeny, as well as ontogeny of species, see [7] for a related approach based on boolean circuits. Also we do not propose in this paper a new method of using DNA's for computations, though experiments such as [1] are extremely interesting.

Though our model seems similar to genetic algorithms, there is an essential difference and our results do not apply to them. Genetic algorithms are based on *selection* operators and *random* mutations and crossing overs. Introducing context sensitive crossing over is against the philosophy behind the genetic algorithms. Genetic algorithms, similarly as other heuristics, are applied to problems where we have no idea how to find an optimal solution efficiently. Therefore we cannot impose restrictions on the search process, such as context sensitiveness of the crossing over operation, or if so, then the restrictions must be very mild. On the other hand, it would be very interesting to analyze genetic algorithms in a similar way as we did for context sensitive crossing over, i.e. to characterize the complexity of such computations using the assumption that the selection operator is computable in random polynomial time and assuming some natural properties of mutations and random crossing over.

A short preliminary version of this paper has appeared in [9].

2 Basic concepts and notation

In this section we recall some basic concepts from complexity theory and introduce a population genetics model with efficiently computable operators.

The standard model of computation used in complexity theory is the *Turing machine.* A Turing machine is determined by a finite *program* 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 program is a set of 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. Multi-tape 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.* We can also interpret a probabilistic Turing machine as a sampling procedure: for a given input x the machine produces samples from a distribution determined by x. 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. In practice we can decide with high confidence, if the input word is accepted or not, by running the machine several times.

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 by a Turing machine in time bounded by a polynomial.

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

$\mathcal{PSPACE}$ 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 $\mathcal{P} \subseteq \mathcal{BPP} \subseteq \mathcal{PSPACE}$. $\mathcal{P}$ and $\mathcal{BPP}$ seem to be very close, they are likely equal, but there is strong evidence that $\mathcal{PSPACE}$ is much larger. Both conjectures seem very hard to prove. $\mathcal{PSPACE}$ is used to approximate feasible parallel computations. In contrast to the above classes, this is a very poor approximation, since one would need exponentially many processors to realize such computations, which is unrealistic. Therefore one should argue more carefully about devices which can compute $\mathcal{PSPACE}$ in polynomially many steps (which is the case of the genetic Turing machines that we shall introduce below). Namely, we can only deduce that the efficiency of such a device is positively correlated with the number of processors, thus by increasing the number of processors we can increase the speed. The brains of higher organisms can be used to document that parallelism can help very substantially. Neurons are extremely slow if compared to transistors in computers, still the brain is able to solve many problems very fast. The reason is apparently massive parallelism used by brains.

Now we present our population genetics formalism. Let G denote the possible genomes. 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 an initial vector z. We shall not consider more complex models such as those with two sexes, two levels (haploid and diploid) etc. These additional features may have some influence on the computational complexity results, but apparently only if we used more precise estimates.

In this paper we want to study computability aspects of the evolution of a population. 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

$$p(g,h;k) = \mathrm{Prob}[P(g,h) = k]. \tag{5}$$

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 $\{0,\dots,m-1\}$. For a subset $T \subseteq \{0,\dots,m-1\}$, 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

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

Inheritance coefficients $p(g,h;k)$ and the corresponding operator will be called *symmetric*, if

$$p(g,h;k) = p(h,g;k),$$

for every g,h,k. Let us note that in [13, 2, 12] the term *symmetric operator* has a different meaning. Firstly, they use a *mating operator*, instead of our inheritance operator, which is given by $\beta : G^4 \to [0,1]$, where $\sum_{k,l} \beta(i,j;k,l) = 1$, (two individuals interact to produce two new individuals). Secondly, they require $\beta(i,j;k,l) = \beta(j,i;k,l) = \beta(k,l;i,j)$, thus the system is, moreover, *locally reversible.*

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 blank 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. In complicated expressions we use $\exp(x)$ instead of e^x. To simplify some estimates we shall use the usual O and Ω notation: $f(n) = O(g(n))$ means $\limsup f(n)/g(n) < \infty$, and $f(n) = \Omega(g(n))$ means $\liminf f(n)/g(n) > 0$.

3 Environment as the input

We want to show that the information provided by the survival operator can be coded into g's. Having this information in the population, we can restrict ourselves to the inheritance operator and study its computational power. The fact that some information about the environment in which a species lives is encoded in its genome is obvious: the species is adapted to its environment and the phenotypes of the individuals are determined by their genotypes. We shall study this process from the point of view of computational efficiency on the abstract level presented above.

Our first task is to show that a large amount if information can be quickly transferred to the genome. This reduces essentially to showing 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 in a few generations 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 inheritance and survival operators 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 probability exponentially close to 1.*

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

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

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

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

determine the survival operator. Clearly, both operators are computable in probabilistic polynomial time. Consider some population z. Let a_i denote the frequency of g's which coincide with x on exactly i places. Let

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

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

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

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 initial segments 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.

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

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

Clearly j encodes the number of generations that have elapsed. 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 the well-known Chernoff 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 $1\vec{\#}$ occurs with frequency α and $0\vec{\#}$ with frequency $1-\alpha$. 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 α 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 $2\mathrm{e}^{-\lambda^2 2^n}$.*

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

Proof. As in the above proof we shall think of each g as a pair (b, y), where b is 0 or 1 and y is a number with binary representation of length n, or y is $\vec{\#}$. The machine P will give the following probabilities:

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

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

Consider the distribution of y's in the $i+1$-st generation. It can be easily computed that it has binomial distribution of order 2^{-i}. Thus, by Chernoff's bound, we get that for $i+1=n$ the frequency of (b,y)'s for which $|y-\alpha| \geq \lambda$ is

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

■

4 Genetic Turing machines

In the previous section we have shown that information about environment can be encoded into the population. Now we want to study, how this information can be processed further. Thus we concentrate on the inheritance operator. Because of a close relations to other extensions of the concept of the Turing machine, we shall call this model the genetic Turing machine. In this section we shall show some reductions and compare this computational model with some related concepts.

Definition 1 *A* genetic Turing machine *(or just* GTM*) 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). It defines an inheritance operator whose coefficients are given by the formula (5). The strings $g \in A^m$ will be called* tapes.

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.

Definition 2 *We say that a GTM P'* polynomially simulates *a GTM P for K generations, if there are polynomials $t(n)$ and $s(n)$, a number $0 < \varepsilon \leq 1$, a probabilistic Turing machine M_1 running in polynomial time and a deterministic Turing machine M_2 running in polynomial time such that the following holds for every n.*

Let n be fixed. Let $z^{(0)} : A^n \to [0,1]$ be an initial population for P; let us denote by $z^{(i)}$ the population in the i-th generation produced by P. The tapes of the simulating machine P' will have length $m = s(n)$; the alphabet of P' will be denoted by A'. We take as the initial population of P' the population obtained by applying M_1 to $z^{(0)}$, i.e.,

$$z'^{(0)}(g) = \sum_h z^{(0)}(h)\, \mathrm{Prob}(M_1(g) = h),$$

and denote by $z'^{(i)}$ the population in the i-th generation produced by P'.

The machine M_2 will determine if a tape $g \in A'^m$ simulates some $h \in A^m$, such tapes will be called simulating*, and if so it will construct such an h. The populations*

$z'^{(i)}$ will simulate the the original populations only for the multiples of t, $t = t(n)$. Namely, we require that the frequency of the simulating tapes be at least ε, i.e.,

$$\sum_{g \text{ simulating}} z'^{(it)}(g) \geq \varepsilon,$$

and the relative frequency of tapes g which simulate a tape h among all simulating tapes in generation it be $z^i(h)$, , i.e.,

$$\frac{\sum_{M_2(g)=h} z'^{(it)}(g)}{\sum_{g \text{ simulating}} z'^{(it)}(g)} = z^i(h),$$

for $i = 0, \ldots, K$.

We say that a GTM P' polynomially simulates *a GTM P, if this holds for all K.*

This definition is a bit more general than we need. In all simulations the machine M_1 will use only a constant number of random bits, thus each simulated tape will be represented by a fixed number of simulating tapes in the initial population. In this section, moreover, the relation of the simulating populations to the simulated populations will be much more direct, in particular, all tapes will be simulating ($\varepsilon = 1$).

Let $F : A^m \times A^m \rightarrow A^m \times A^m$ be a function. Then we can interpret F as a random variable $\mathbf{F} : A^m \times A^m \rightarrow A^m$ as follows. Suppose $F(g,h) = (g', h')$, then we think of F as

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

An inheritance operator thus given by F represents the situation where parents have always two children uniquely determined by the parents. We shall call such operators and genetic Turing machines *conservative*. Let us note that it is consistent to think of such a system as a population of infinitely many strings where the evolution is done by randomly pairing them and replacing g, h by g', h', where $F(g,h) = (g', h')$. Clearly, a conservative operator given by an F is symmetric iff $F(g,h) = (g', h')$ and $F(h,g) = (g'', h'')$ implies $\{g', h'\} = \{g'', h''\}$ for every g, h.

Proposition 4.1 *Any genetic Turing machine P can be polynomially simulated by a symmetric conservative genetic Turing machine given by some $F : A^m \times A^m \rightarrow A^m \times A^m$ computable by a deterministic Turing machine in polynomial time.*

Proof. Let the genetic Turing machine be given by an alphabet A and a probabilistic Turing machine P, let the input size n be given. The idea of the proof is to simulate a pair of the original tapes as a new longer tape. We simulate one application of the original inheritance operator by T steps of the new one, where $T-1$ is the running time of the machine P. We start with tapes which have two occurrences of the simulated tape. In the first step, called *the crossing over step,* we shall cross over the tapes so that the two occurrences are uniformly mixed. Then the frequency of the pairs occurring on the simulated tapes will be the same as if we have drawn them randomly from the simulated population. In the next $T-1$ steps, let us call them *rewriting steps,* each step of the probabilistic Turing machine P working on a pair of tapes is simulated by one application of the new inheritance operator. So after T steps the frequency of the halves of the tapes will be the same as the frequency of the original tapes after one application of the original inheritance operator.

Let m be the space needed by P working on inputs of length $2n$, more precisely the length of strings needed to encode such configurations of P. W.l.o.g. we can assume that

- the string $kk\vec{\#}$ is different from the strings that code configurations of P working on such inputs,
- on each input of size $2n$ it always stops after exactly $T-1$ steps, T bounded by a polynomial in n.

The simulating tapes will have length m. The simulation of an initial population will be given by the transformation (the machine M_1 in the definition)

$$g \mapsto 0gg\vec{\#} \text{ with probability } 1/2,$$
$$g \mapsto 1gg\vec{\#} \text{ with probability } 1/2.$$

Define F as follows. For $g, h \in A^n$, $a, b \in \{0,1\}$,

$$F(agg\vec{\#}, bhh\vec{\#}) = (aghw, bhgw')$$

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

$$F(aw_1, bw_2) = (aw_1', bw_2')$$

where w_1 is a configuration (not final) and w_1' is the next configuration corresponding to the random bit b and where w_2 is a configuration (not final) and w_2' is the next configuration corresponding to the random bit a;

$$F(agw, bhw') = (agg\vec{\#}, bhh\vec{\#})$$

where gw and hw' encode final configurations of P.

For all other inputs F can be defined arbitrarily.

The simulation proceeds as follows. First the tapes $agg\vec{\#}$ and $bhh\vec{\#}$ are randomly mixed into $agh\vec{\#}$ and $bhg\vec{\#}$ by crossing over and the computation of P on them starts. Then for $T-1$ generations F works as the linear operator of the probabilistic machine, except that it is always performed on pairs. Note that each configuration aw_1 mates in half of the cases with a configuration bw_2 where $b=0$ and in the other half with a configuration where $b=1$. 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-th generation F transforms the final configuration gw of P into $gg\vec{\#}$. (The "final 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 iT-th generation of the GTM determined by F simulates the i-th generation of the GTM determined by P. ∎

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. This follows from the next lemma. ∎

Let us call a population of tapes z *uniformly mixed,* if the frequency of g is the product of the frequencies of its letter i.e.,

$$z(g) = \prod_i z|_{\{i\}}(g)$$

Lemma 4.3 *Let z be a population of tapes $g \in A^d$, $d > 1$. Apply to it consecutively the conservative operators $C_1, \ldots, C_{d-1}$, where C_i simply switches the i-th letter. Then the resulting population is uniformly mixed.*

Proof-sketch. For $d = 2$ it is well-known, for larger d use induction. ∎

Let us observe that this property is inherited by subpopulations determined by subsets of the alphabet, i.e., if $z : A^m \to [0,1]$ is uniformly mixed and $B \subseteq A$, then $z|_{B^m}$ is also uniformly mixed.

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 4.4 *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 the program of the simulated GTM. ∎

Let us now compare genetic Turing machines with probabilistic 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

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

the evolution of z is defined by

$$z'(h) = \sum_g p(g;h)z(g). \tag{6}$$

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; however, in Section 6 we shall show that one can base genetic Turing machines on crossing over.) Let us observe that the conditions (2) correspond to the following ones for probabilistic Turing machines.

$$\begin{array}{l} p(g;k) \geq 0, \\ \sum_k p(g;k) = 1. \end{array} \tag{7}$$

In particular, computations of probabilistic Turing machines are Markov's processes with operators computable by probabilistic Turing machines in polynomial time and, vice versa, such processes can be simulated by computations of probabilistic Turing machines.

Final, we mention briefly another example of a generalization of Turing machines which has a linear operator. It is the *Quantum Turing machine* introduced by Deutch [4]. This rather exotic concept seams to be physically well-founded and stronger than probabilistic Turing machines. 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 the so called *amplitudes* and the probability is counted as their L_2 norm. The transformation (6) must preserve the L_2 norm, hence the matrix $\{p(g;h)\}_{g,h}$ must be unitary. Formally, this means that the requirement (1) is replaced by

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

and (7) 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 this section we show that the power of genetic Turing machines can be characterized using Turing machines with bounded space.

A genetic Turing machine determines evolution of a distribution (population) $z : A^m \rightarrow [0,1]$ 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 $\#$.

The result of a computation of a genetic Turing machine is the same as in the case of a probabilistic Turing machine, namely, a probability distribution. 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. $z|_{\{0\}}(1) \geq 3/4$ or $z|_{\{0\}}(1) \leq 1/4$). We define that a bounded error genetic Turing machine accepts the set of the strings for which in the final population $z|_{\{0\}}(1) \geq 3/4$.

Theorem 5.1 *A language L is accepted by some bounded error genetic Turing machine with polynomially long tape in polynomially many steps iff L is in $\mathcal{PSPACE}$.*

Proof. 1. First we show that every such GTM can be simulated, with a polynomial precision, by a Turing machine with polynomially bounded tape. I am indebted to Russell Impagliazzo for the idea of this proof, ([2] uses the same idea).

The idea is to count approximately the frequencies of the tapes $z(g)$ gradually in all generations. To compute the frequency $z(g)$ in generation t we need to compute the frequencies of all possible ancestors. The number of the ancestors is exponential in t, but we can do it so that we always keep the frequencies of at most t of them.

Let m be the length of the tapes of a GTM, m polynomial in the input size n, let a be the size of the alphabet used on the tape. Suppose we want to simulate the GTM for K generations, K polynomial in n.

First we estimate the precision needed to compute the frequencies. Let $\varepsilon_t \geq 0$ be the precision in generation t. We need that $\varepsilon_K a^m < 1/4$. Then we can accept, if the approximation of the frequency $z_{\{0\}}(1)$ (the frequency of the tapes with 1 on the first position) is $> 1/2$. Suppose we count all frequencies using binary numbers between 0 and 1 with $b > -\log_2 \frac{1}{4} 3^{-K} a^{-m(K+1)}$ bits. Then the rounding error will be some $\varepsilon < \frac{1}{4} 3^{-K} a^{-m(K+1)}$.

Let us denote by $z^{(t)}(g)$ the frequency of g in the t-th generation and $\tilde{z}^{(t)}(g)$ its approximation. Suppose $\tilde{z}^{(t+1)}(g)$ is counted using the approximations in generation t without rounding. Then

$$z^{(t+1)}(k) - \tilde{z}^{(t+1)}(k) = \sum_{g,h} p(g,h;k) \Big(z^{(t)}(g)(z^{(t)}(h) - \tilde{z}^{(t)}(h)) - z^{(t)}(h)(z^{(t)}(g) - \tilde{z}^{(t)}(g)) -$$

$$(z^{(t)}(h) - \tilde{z}^{(t)}(h))(z^{(t)}(g) - \tilde{z}^{(t)}(g)) \Big) .$$

This gives

$$\varepsilon_{t+1} \leq \sum_{g,h} \left(z^{(t)}(g)\varepsilon_t + z^{(t)}(h)\varepsilon_t + \varepsilon_t^2 \right) = 2a^m \varepsilon_t + a^{2m} \varepsilon_t^2 .$$

Thus if we round to b bits, we get

$$\varepsilon_{t+1} \leq 2a^m \varepsilon_t + a^{2m}\varepsilon_t^2 + \varepsilon. \tag{8}$$

We shall prove by induction that

$$\varepsilon_t \leq (3a^m)^t \varepsilon, \tag{9}$$

for $t \leq K$. Observe that for $t \leq K$, (9) implies

$$\varepsilon_t \leq (3a^m)^t \varepsilon < \frac{1}{4} 3^{t-K}(a^m)^{t-K-1} \leq \frac{1}{4} a^{-m}, \tag{10}$$

which gives the required precision for $t = K$. We have $\varepsilon_0 = 0$ as the initial frequencies are all 0's and 1. Now suppose that (9) holds for $t < K$. Then we get from (8) and (10)

$$\varepsilon_{t+1} \leq 3a^m \varepsilon_t \leq (3a^m)^{t+1}\varepsilon.$$

Thus it is sufficient to compute with only polynomial precision, namely $b = O(mK^2)$.

Now we can estimate the space s_t needed for computing $\tilde{z}^t(g)$ from the formula

$$\tilde{z}^{t+1}(k) = \sum_{g,h} p(g,h;k)\tilde{z}^t(g)\tilde{z}^t(h).$$

We compute the sum by adding the summands one by one in some order. Thus we need to store 1. the partial sum, i.e., b bits, 2. the last considered pair (g,h), i.e., $O(2m \log a)$ bits, 3. the current $\tilde{z}^t(g)$ and $\tilde{z}^t(h)$, i.e., $2b$ bits. Furthermore we need s_t bits for computing $\tilde{z}^t(g)$ and $\tilde{z}^t(h)$, q bits for computing the coefficients $p(g,h;k)$, $q = m^{O(1)}$, and $O(\log b)$ bits for multiplication. Hence

$$s_{t+1} = O(b + 2m \log a) + s_t + q = s_0 + t \cdot O(b + 2m \log a + q).$$

(s_0 is polynomial, since the encoding of the input in the initial population is trivial.) To add the frequencies of those tapes which have 1 on the first position we need space $s_K + O(b)$. Hence polynomial space is sufficient to compute the frequencies with a sufficient precision. Thus we have proved the first part of the theorem.

2. Now we show that every language in $\mathcal{PSPACE}$ can be simulated by a GTM using polynomially long tape and polynomially many steps.

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 of M working on x and w_3 is an accepting configuration, and $b = 0$ otherwise.
3. On an input pair $(x, 0, i, w_1, w_2), (x, 0, j, w'_1, w'_2)$ it outputs $(x, 0, i, w_1, w_2)$, if $i > j$, and $(x, 0, j, w'_1, w'_2)$, if $i < j$. If $i = j$ and $w_2 \neq w'_1$, then the strings do not interact.
4. On an input pair $(x, 1, i, w_1, w_2), (x, b', i', w'_1, w'_2)$ it produces $(x, 1, i, w_1, w_2)$.

In all other cases the pairs do not interact.

It is clear, how the evolution of the population will look like. Firstly, 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 $(x, 1, i, w_1, w_2)$, with i arbitrary, 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 $(x, 0, i, w_1, w_2)$ 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

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

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 $O(N^2) = n^{O(1)}$ 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 be more precise. Let C be a set of quadruples of finite strings (u_1, v_1, u_2, v_2), $|u_1| = |v_1|$, $|u_2| = |v_2|$, in alphabet A. We shall call C *a set of contexts* and always assume that it is a finite set. *Context sensitive crossing over* determined by a set of contexts C is a transformation of pairs of strings into pairs of strings which works as follows. Let $g, h \in A^m$. 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 right. Otherwise we just move to the next position to the right. Thus if

g_1	g_2	$\dots\ \dots$	g_r	$\dots$	g_i	g_{i+1}	g_{i+2}	$\dots$	g_s	$\dots\ \dots$	g_m
h_1	h_2	$\dots\ \dots$	h_r	$\dots$	h_i	h_{i+1}	h_{i+2}	$\dots$	h_s	$\dots\ \dots$	h_m

and

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

then we get

g_1	g_2	$\dots\ \dots$	g_r	$\dots$	g_i	h_{i+1}	h_{i+2}	$\dots$	h_s	$\dots\ \dots$	h_m
h_1	h_2	$\dots\ \dots$	h_r	$\dots$	h_i	g_{i+1}	g_{i+2}	$\dots$	g_s	$\dots\ \dots$	g_m

Otherwise we just advance

g_1	g_2	$\dots\ \dots$	g_r	$\dots$	g_i	g_{i+1}	g_{i+2}	$\dots$	g_s	$\dots\ \dots$	g_m
h_1	h_2	$\dots\ \dots$	h_r	$\dots$	h_i	h_{i+1}	h_{i+2}	$\dots$	h_s	$\dots\ \dots$	h_m

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.

Context sensitive crossing over is a special kind of a conservative operator. If $(u_1, v_1, u_2, v_2) \in C \Leftrightarrow (v_1, u_1, v_2, u_2) \in C$, then we say that the contexts are *symmetric.* The operator corresponding to symmetric contexts is symmetric.

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 special case where the probabilities are just 0 or 1, thus getting a stronger mathematical result. On the other hand, it would be harder to prove such a result under the condition that the probabilities are *never* sharply 0 or 1, which is closer to reality. We conjecture that our result can be extended in this way.

The rest of the paper will be devoted to the proof of the following theorem.

Theorem 6.1 *Every genetic Turing machine can be polynomially simulated, for a polynomial number of generations, by a genetic Turing machine using context sensitive crossing over.*

Before starting the proof, we give a brief overview. The proof will be an extension of the proof of Proposition 4.1. In that proof we simulated one generation by a round consisting of several steps in which the system developed as a linear system determined by a probabilistic Turing machine and then there was a single step consisting of crossing over in the middle (without any restrictions). Thus it remains to simulate a probabilistic Turing machine. This is done by thinking of a Turing machine as a rewriting system and using some auxiliary tapes as a stock of symbols. To rewrite a tape we replace its part by a homologous part of a suitable auxiliary tape. This can be done by crossing over and we clearly need only a finite set of contexts to ensure that we rewrite according to given rewriting rules.

There are, however, several obstacles to be overcome. Firstly, we cannot force the simulating tapes to mate only with appropriate auxiliary tapes. Thus such a rewriting will be a random process in which only a fraction of tapes will be rewritten. Then some tapes will advance fast, while the others will be slow or do not move at all. To get a correct simulation, we have to prohibit interactions between tapes which simulate different generations. Therefore the information on the number of the simulated generation will be encoded on the tapes. Then we shall use the fact that the "age" of the simulating tapes is very much concentrated around some value, due to the law of large numbers.

The presence of auxiliary tapes and tapes of different age complicates also the simulation of the crossing over step. Again we cannot force tapes to mate only with simulating tapes of the same age, hence only a part of the tapes will cross over in one generation. Therefore we add some marks to control, if the tapes already crossed over, and run this process for several steps, in order to reduce the number of those who did not to minimum.

In order to control the crossing over phase, we equip the tapes with clocks. The clocks will be simulated in the same way as the Turing machine and we shall again refer to the law of large numbers when arguing that most of them show approximately the same time.

As the proof is long, we split it into several parts. In the first two subsections we develop a simulation of Turing machines by crossing over. Then we estimate the rate of mixing in the presence of noninteracting, or weakly interacting, tapes. In the last two subsections we describe the simulation and compute estimates on the frequencies of simulating tapes.

6.1 Simulation of Turing machines by rewriting

Our first goal is to simulate Turing machines by rewriting tapes locally. We shall work with deterministic Turing machines first and then in the next subsection we observe that the argument can be easily extended to probabilistic Turing machines.

Rewriting rules are defined as a finite set of pairs of strings. A pair (u, v) from the set will be called a *rule* and written as $u \to v$. A *rewriting step* is any transition of the form

$$wuw' \mapsto wvw'$$

where $u \to v$ is a rule. In our simulation rewriting will be deterministic which means that exactly one rule will be always possible to apply (except, possibly, for the final string). Furthermore we need a very special form of rewriting, namely, that always only one letter is rewritten. This means that the rules have the form

$$uau' \to ubu', \tag{11}$$

where a and b are just letters from the alphabet in question.

We shall start with the obvious simulation of Turing machines computations where an instantaneous configuration of the machine is encoded by a string which is the content of the tape except that at the position of the head we have a letter which encodes the original symbol and the state of the machine. In this representation one step of the machine is simulated by one rewriting in which two consecutive letters are changed. We can get a more special way of rewriting in a two element alphabet by representing each letter by a string and simulating one step of computation by a fixed constant number of rewritings.

Lemma 6.2 *Computations of Turing machines can be simulated by one-letter rewriting in a two-element alphabet with at most constant slowdown.*

Proof. Suppose we consider strings in an alphabet A. Let rewriting rules be given such that they always rewrite at most two consecutive letters. We take a larger alphabet B which is the disjoint union of A and $A \times A$. We replace each rule

$$uxyw \to ux'y'w \quad \text{where } x \neq x', y \neq y', \text{ are letters,}$$

by five rules (which we will write as they will be successively applied)

$$uxyw \to u(x, x')yw \to u(x, x')(y', y')w \to ux'(y', y')w \to ux'y'w.$$

Let us note that the new rewriting system has the following property. If (11) is a rule, then the reverse rewriting $ubu' \to uau'$ is not a rule. Now replace B by $\{0, 1\}$ and represent each $a \in B$ as $111011110^{k_a}10^{l_a}$, $k_a + l_a + 1 = |B|$ assigning different numbers to different letters. Then replace each rule (11) by two rules

$$111011110^{k_a}10^{l_a} \to 111011110^{c}10^{d}10^{e} \to 111011110^{k_b}10^{l_b},$$

where $c = min(k_a, k_b), c + 1 + d = max(k_a, k_b), c + 1 + d + 1 + e = |B|$. Due to the property of the intermediate system, for each $111011110^{c}10^{d}10^{e}$ there is at most one rule with this antecedent. ∎

6.2 Simulation of Turing machines by crossing over

An essential part of the simulation of a general GTM by a GTM with crossing over is a simulation of probabilistic Turing machines. Due the above simulation by one-letter rewriting, the task is easy. First we shall consider deterministic Turing machines.

We start with a set of one-letter rewriting rules in the two-element alphabet $\{0, 1\}$. Each word g will be represented twice, once as g and once as its negative image where we switch 0 with 1. These two versions will have equal frequency. Furthermore we insert some fixed distinguishing words, w^+_{pro} for positive versions and w^-_{pro} for negative versions, of constant length between each two consecutive letters of g or its negative image. Thus g will become

$$g_1 w^+_{pro} g_2 w^+_{pro} \dots w^+_{pro} g_m, \text{ or } g_1 w^-_{pro} g_2 w^-_{pro} \dots w^-_{pro} g_m.$$

Such words will be called *proper tapes*. We need also *auxiliary tapes* which will have form

$$h_1 w_{aux} h_2 w_{aux} \dots w_{aux} h_m.$$

Here the word w_{aux} has the same length as w^+_{pro} and w^-_{pro}, but it is different, $h_1, \dots, h_m$ are letters. We will call occurrences of bits which do not belong to the distinguishing words *information bits*. We need not only that w_{aux}, w^+_{pro} and w^-_{pro} are different, but also that there exists a constant c_d such that for every segment of length c_d of homologous parts,

1. we can distinguish proper tapes from auxiliary ones and a proper tape corresponding to some positive word g from a proper tape corresponding to a negative word g',
2. we can determine which part of such a segment of a proper tape are the information bits g_i and which are the bits of the distinguishing words w^+_{pro} or w^-_{pro}.

It is clear that such words can be easily chosen. For instance take $w_{aux} = 01111000$, $w^+_{pro} = 01111001$, $w^-_{pro} = 01111011$ and $c_d = 16$.

The crossing over rules will correspond to the rewriting rules. Rewriting one symbol will correspond to an exchange of an information bit between a proper tape and an auxiliary tape, provided the bits are different. (This requires two contexts.) A pair of proper tapes, or a pair of auxiliary tapes will never interact. Taking a sufficiently large context, any ambiguity can be eliminated. In particular, if we start with a population of some proper and some auxiliary tapes, all subsequent populations will contain only such tapes.

We shall start with 1/4 of tapes corresponding to the positive representation of the initial configuration of the Turing machine, 1/4 corresponding to the negative representation of the initial configuration, 1/4 of auxiliary tapes with all information bits 0 and 1/4 of auxiliary tapes with all information bits 1. The particular choice of auxiliary tapes is not important, we only need that the frequency of information bits at any locus is the same for 0 and 1. This arrangement allows us to estimate exactly how the population develops. The point is that the following two properties will be preserved in all generations:

- the subpopulation of proper tapes is symmetric with respect to switching information bits 0 with 1;
- the frequency of the information bit 0 at a particular locus of auxiliary bits will be equal to the frequency of the information bit 1 at this locus (actually, for the particular choice above, the auxiliary tapes will enjoy the stronger symmetry property of the proper tapes).

This is a direct consequence of the symmetry of the contexts.

The reason for one-letter rewriting is that the speed of simulation will be independent of the content of proper tapes which will be very important later. Namely, in each generation exactly 1/4 of proper tapes will be changed, as if rewritten in the simulated rewriting system, and the rest will remain the same. This corresponds to the Markov's process where we rewrite a tape with probability 1/4 and leave it as it is with probability 3/4.

A probabilistic Turing machine can be described by two sets of rewriting rules, where we apply a rule from the first, resp. second set, if the random bit is 0, resp. 1. If we start with an initial configuration and follow the rules, there will always be exactly one possibility for rewriting for each of the random bits. To simulate probabilistic machines we use two types of each auxiliary tape distinguished by two different words w^0_{aux} and w^1_{aux}; the two types having the same frequency. When rewriting by an auxiliary tape we determine the random bit by the type of the auxiliary tape. As in the case of the simulation of deterministic Turing machines, in each generation exactly 1/4 proper tapes will be rewritten. Thus we get:

Lemma 6.3 *After t generations the relative frequency of proper tapes corresponding to the s-times rewritten initial tape will be* $\mathrm{B}(t, 1/4)$ *(the binomial distribution of dimension t and mean $\frac{1}{4}t$).*

Due to Chernoff-type bounds, the "age" of proper tapes will be very much concentrated around $\frac{1}{4}t$, thus we get a very good simulation of probabilistic Turing machine computations.

6.3 Simulation of uniform mixing

We would like to mix the rewritten tapes in the same way as in the proof of Theorem 4.1, however there is an essential obstacle now. The problem is that there are also the auxiliary tapes which will always mate with proper tapes, thus we can never achieve uniform mixing of proper tapes. The idea, how to overcome this problem, is to label the halves of the proper tapes by several different marks. If we start with equal marks for both halves, we can distinguish those which already crossed-over by observing two different marks at the halves.

We shall make these consideration precise and solve the problem first on an abstract level. Then we shall combine it with the rewriting simulation.

Suppose we have a population $u : A^2 \to [0, 1]$ which we want to mix uniformly. Here we use one letter to encode a half-tape, since we do not care about the structure of the half-tapes. Furthermore suppose that we have another element Ω which does not interact with the elements of A^2. This will correspond to the auxiliary tapes and those proper tapes which are not in the crossing over stage. Again we are not interested in the structure of auxiliary and inactive proper tapes at this moment, so we can all represent by a single element.

We want to define a crossing over like operator so that after a few steps we have a large uniformly mixed subpopulation which can be easily distinguished from the rest. We shall extend the original tapes by adding one of the three labels to each half, i.e., we take $A' = A \times \{0, 1, 2\}$. (We can take 3 or more, 2 are not enough as will be clear from the computation.) We shall simulate the original initial population u by x defined by

$$x((a_1, i), (a_2, i)) = \frac{1}{3} u(a_1, a_2), \tag{12}$$

for $a_1, a_2 \in A$, $0 \leq i \leq 2$, i.e., for other tapes x is 0.

Now we define a conservative operator Φ on $G = A' \times A' \cup \{\Omega\}$ by switching the two parts in $((a_1, i), (a_2, i))$ and $((b_1, j), (b_2, k))$ if $i \neq j$ and $i \neq k$ and requiring that in all other cases the tapes do not interact. To avoid confusion, let us write it explicitly. The conservative operator Φ will be given by a function $F : G^2 \to G^2$ defined as follows. For $i \neq j$ and $i \neq k$

$$\begin{aligned} F(((a_1, i), (a_2, i)), ((b_1, j), (b_2, k))) &= (((a_1, i), (b_2, k)), ((b_1, j), (a_2, i))) \\ F(((a_1, j), (a_2, k)), ((b_1, i), (b_2, i))) &= (((a_1, j), (b_2, i)), ((b_1, i), (a_2, k))) \end{aligned} \tag{13}$$

and all other pairs do not interact (i.e., $F(g, h) = (g, h)$).

When dealing with strings instead of just letters, we shall represent the pairs $((g, i), (h, j))$ as the string $giwjh$, where w is some fixed constant length word marking the middle of the string. Then the above operator will really be given by a context sensitive crossing over.

Let us denote by

$$\begin{aligned} E &=_{df} \{((a_1, i), (a_2, i)) \; ; \; a_1, a_2 \in A, i \in \{0, 1, 2\}\}, \\ U &=_{df} \{((a_1, j), (a_2, k)) \; ; \; a_1, a_2 \in A, \; j, k \in \{0, 1, 2\}, j \neq k\}. \end{aligned}$$

We can think of x as a population on G where $x(g) = 0$ for $g \in G \setminus E$. In order to describe the evolution given by Φ we define two more populations y, z on G. For $i \neq j$

$$y((a, i), (b, j)) = \frac{1}{6} \left(\sum_{b' \in A} u(a, b') \right) \left(\sum_{a' \in A} u(a', b) \right) \tag{14}$$

and $y(g) = 0$ for all other g's.

$$z(\Omega) = 1 \tag{15}$$

and $z(g) = 0$ for $g \neq \Omega$. The projection of y onto $A \times A$ is just the population obtained from u by uniform mixing. Hence we want to get as large as possible portion of the whole population to be equal to y.

Our initial population will consist of elements of E and Ω, more precisely it will be of the form

$$v_0 = \alpha_0 x + \gamma_0 z$$

with $\alpha_0, \gamma_0 > 0, \alpha_0 + \gamma_0 = 1$. In the following lemma we shall show that in all the following generations we will have populations of the form

$$v = \alpha x + \beta y + \gamma_0 z$$

with $\alpha, \beta \geq 0, \alpha + \beta + \gamma_0 = 1$ and the coefficient α will decrease exponentially. Since the share of z does not change, it means that gradually y will replace almost all x.

Lemma 6.4 *Applying the operator Φ to a population of the form $v = \alpha x + \beta y + \gamma z$ with $\alpha, \beta, \gamma \geq 0, \alpha + \beta + \gamma = 1$ produces a population of the form $v = \alpha' x + \beta' y + \gamma z$ with*

$$0 \leq \alpha' < \alpha \frac{2+\gamma}{3}.$$

Proof. Since Ω does not interact with others and is never produced from others the term γz will be preserved.

The fact that we obtain a population of the same type is also intuitively clear, but we shall check it by computation. The pairs of tapes which do not interact will contribute to x, y or z, so we only need to consider those which do interact. These will produce tapes from A'^2. The frequency of such a tape $((a,i),(b,l))$ is an expression with terms of the form

$$\sum_{a',b'} v((a,i),(b',j))v((a',k),(b,l)),$$

which can be written as

$$\sum_{b'} v((a,i),(b',j)) \sum_{a'} v((a',k),(b,l)). \tag{16}$$

Due to symmetry we need to consider only the first term. If $i = j$ then this term comes from the part x. Namely,

$$v((a,i),(b',j)) = \alpha x((a,i),(b',i)) = \frac{\alpha}{3} u(g_1, g_2).$$

If $i \neq j$, then it comes from y, thus

$$\sum_{b'} v((a,i),(b',j)) = \sum_{b'} \frac{1}{6} \left(\sum_{b''} u(a,b'') \right) \left(\sum_{a'} u(a',b') \right) =$$

$$\frac{1}{6} \left(\sum_{b''} u(a,b'') \right) \left(\sum_{b'} \sum_{a'} u(a',b') \right) = \frac{1}{6} \left(\sum_{b''} u(a,b'') \right).$$

Thus in all cases (16) can be written in the form

$$c \cdot \left(\sum_{b' \in A} u(a,b') \right) \left(\sum_{a' \in A} u(a',b) \right)$$

for some constant c. Hence these terms are just a fraction of y.

Now we estimate α'. Note that tapes in E are produced only from mating pairs where at least one element is in E and the two tapes do not interact. For a given $g \in E$, g does not interact with $\frac{1}{3}$ of tapes in E, with $\frac{2}{3}$ tapes in U and it does not interact with Ω. Thus we have

$$\alpha' = \alpha\left(\frac{1}{3}\alpha + \frac{2}{3}\beta + \gamma\right) = \alpha\left(\frac{2}{3} - \frac{1}{3}\alpha + \frac{1}{3}\gamma\right) \le \alpha\frac{2+\gamma}{3}.$$

∎

Starting with a population where E have frequency α and U have frequency 0, after t generations we obtain a population where the frequency of U will be

$$\ge 1 - \gamma - \alpha\left(\frac{2+\gamma}{3}\right)^t.$$

In this way we obtain all except of an exponentially small fraction of proper tapes uniformly mixed (we are disregarding the indices $\{0,1,2\}$).

Unfortunately, this is only a rough description of what will happen in the simulation of GTM's by crossing over. In our simulation not all proper tapes will always be ready for the crossing over stage. They will gradually enter and leave this process. Therefore we need to prove a more complicated statement, whose proof is, however, an easy extension of the above one.

We consider evolution which will be close to the above model, but, strictly speaking, it will not be given by a conservative operator. In fact, the operator will change in time too. We shall describe it by symmetric inheritance coefficients $p_t : G^3 \to [0,1]$, where t runs over the generations. We shall think of them as modified inheritance coefficients of Φ determined by some nonnegative constants c_t, d_t and e_t. As above, we shall use the distributions x, y, z determined by a fixed given distribution u (see (12),(14),(15)). We define

$p_t(\Omega,\Omega;\Omega) = 1 - c_t$;

$p_t(\Omega,\Omega;g) = c_t x(g)$;

for $g \in E$ we put

$p_t(g,\Omega;\Omega) = p_t(\Omega,g;\Omega) = \frac{1}{2} + \frac{d_t}{2} - \frac{e_t}{2}$,

$p_t(g,\Omega;g) = p_t(\Omega,g;g) = \frac{1}{2} - \frac{d_t}{2} - \frac{e_t}{2}$;

for $g,h \in E,\ g \ne h$ we put

$p_t(g,\Omega;h) = p_t(\Omega,g;h) = 0$;

for $g \in E,\ h \in U$, if $g = ((a_1,i),(a_2,i))$, $h = ((a_1,i),(b,j))$, $j \ne i$, we put

$p_t(g,\Omega;h) = p_t(\Omega,g;h) = \frac{3}{4}e_t \sum_{a\in A} x((a,j),(b,j)) = \frac{e_t}{4}\sum_{a\in A} u(a,b)$;

symmetrically, if $g = ((a_1,i),(a_2,i))$, $h = ((a,j),(a_2,i))$, $j \ne i$, we put

$p_t(g,\Omega;h) = p_t(\Omega,g;h) = \frac{3}{4}e_t \sum_{b\in A} x((a,j),(b,j)) = \frac{e_t}{4}\sum_{b\in A} u(a,b)$;

for $g \in E,\ h \in U$ but not of the above form, we put

$p_t(g,\Omega;h) = p_t(\Omega,g;h) = 0$;

also for $g,h \in A'$

$p_t(g,h;\Omega) = 0$;

$g \in U$ and Ω do not interact; finally, for $g,g',h \in A'$ we define the coefficients p_t in the same way as in (13), i.e.,

$p_t(((a_1,i),(a_2,i)),((b_1,j),(b_2,k));((a_1,i),(b_2,k))) = 1/2,$
if $i \neq j, k$ etc.

The meaning of these equations is that a fraction c_t of Ω is moved to E (with the distribution x) and $g \in E$ mating with Ω partly do not interact, partly g becomes Ω (fraction d_t) and partly an element of U is produced as if g interacted with elements $h \in U$ with the distribution y (fraction e_t).

The operators corresponding p_t will be denoted by Ψ_t. We shall show that as long as c_t is kept very small and the frequency of Ω is bounded away from 1, the part of the distribution given by x will still decrease exponentially.

Lemma 6.5 *Applying the operators* $\Psi_0, \ldots, \Psi_{s-1}$ *to a population of the form* $v_0 = \alpha_0 x + \beta_0 y + \gamma_0 z$ *with* $\alpha_0, \beta_0, \gamma_0 \geq 0$, $\alpha_0 + \beta_0 + \gamma_0 = 1$ *produces a population of the form* $v_s = \alpha_s x + \beta_s y + \gamma_s z$, $\alpha_s, \beta_s, \gamma_s \geq 0$, $\alpha_s + \beta_s + \gamma_s = 1$. *If* $\gamma_t \leq \gamma \leq 1$ *for* $0 \leq t < s$, *then*

$$\alpha_s \leq \alpha_0 \left(\frac{2+\gamma}{3}\right)^s + \sum_{t=0}^{s-1} c_t.$$

Proof. The proof that the operators preserve the type of the distribution is almost the same as in Lemma (6.4). The only essential difference is that $h \in U$ may be produced from $g \in E$ and Ω. The coefficients, however, ensure that the distribution of $h \in U$ produced in this way is just y, provided that the distribution of $g \in E$ is x, which is trivially ensured. So it remains to estimate α_s. As above, we get by induction

$$\alpha_{t+1} = \alpha_t \left(\frac{1}{3}\alpha_t + \frac{2}{3}\beta_t + \gamma_t - d_t\gamma_t\right) + c_t\gamma_t^2 \leq$$

$$\alpha_t \left(\frac{1}{3}\alpha_t + \frac{2}{3}\beta_t + \gamma_t\right) + c_t \leq \alpha_t \frac{2+\gamma_t}{3} + c_t \leq \alpha_t \frac{2+\gamma}{3} + c_t \leq$$

$$\left(\alpha_0 \left(\frac{2+\gamma}{3}\right)^t + \sum_{i=0}^{t-1} c_i\right) \frac{2+\gamma}{3} + c_t \leq \alpha_0 \left(\frac{2+\gamma}{3}\right)^{t+1} + \sum_{i=0}^{t} c_i.$$

■

6.4 Simulation of general genetic Turing machines by crossing over – description

Now we are ready to start the proof of Theorem 6.1. We shall use the proof of Proposition 4.1, namely, instead of simulating the original GTM we shall simulate the GTM M constructed in that proof slightly modified. Namely we shall omit the first bits that were used in order to avoid randomness in the computation. Instead, we shall use probabilistic Turing machines. Furthermore we shall assume that the tape of M has length $2m$ and whenever it encodes $gg\vec{\#}$, then one occurrence of g is on one half and the other is on the other half. Then the mixing is obtained by crossing over the tapes in the middle. We assume that the alphabet of M is $A = \{0,1\}$. Thus the GTM M is determined by a probabilistic Turing machine M_0 which works on tapes A^{2m}, uses only alphabet $A = \{0, 1\}$, and for some constant T, (more precisely T depends on the input size, i.e., on m, but not on g) it stops on each tape g after exactly $T - 1$ steps producing another tape in A^{2m}. M works in rounds of length T. The 0-th round is a single crossing over operation (with no restrictions, so the halves are uniformly mixed). Then, in each next round, it works as the linear operator given by M_0 for $T-1$ generations, then it applies the quadratic operator of crossing over in the middle. For sake of symmetry, we shall assume that the tapes in crossing over steps are of the form $g\vec{\#}h\vec{\#}$, $|g\vec{\#}| = |h\vec{\#}| = m$, instead of $gh\vec{\#}$ as in Proposition 4.1.

Moreover we shall think of M_0 as a rewriting system, rather than a Turing machine, which rewrites always only one symbol. It is clear from Lemma 6.2 and the proof of Proposition 4.1 that it suffices to simulate only such operators.

We shall denote the simulating machine by N. First we shall describe the structure of the tapes which will appear in the simulation (with nonzero frequency). The tapes will be in alphabet $A = \{0, 1\}$ and will have length n, an even integer bounded by a polynomial in length of the simulated tape m. We shall simulate a polynomial number of rounds K.

There will be two main types of tapes – proper and auxiliary – as described in Section 6.2. We shall further split each auxiliary tape into two (thus we have four types altogether): tapes of the first kind will be used to rewrite the left halves of the proper tapes and the tapes of the second kind will be used for the right halves. According to this we shall use four distinguishing words for auxiliary tapes. This will ensure that only one of the halves of a proper tape is rewritten, even if there are places on both halves which can be rewritten. We shall use the simulation of Turing machines described above, thus always at most one letter of the proper tape is rewritten and at each information bit the frequency of 0's and 1's is $1/2$ and the frequency of auxiliary tapes is $1/2$, equally split between the two types.

The proper tapes will be divided into two segments called *left half-tape* and *right half-tape* which will have the same length and similar structure. The border between them will be called *the crossing over locus.* This will be determined by special subwords *the middle markers.* The two half-tapes will have structure symmetric with respect to the crossing over locus; the actual content may be different. Each half-tape will have two versions where one is obtained from the other one by replacing 0 by 1. The versions of the left half-tapes can mix with the versions of the right half-tapes arbitrarily, thus we have four versions of each proper tape, each of the four with the same frequency. The contexts will be symmetric with respect to the four versions, therefore we need only to count the total frequency of all four, or just concentrate on one of them.

Now we describe the structure of a half-tape. It contains parts called *the simulated tape, the clock, the number of a round, the garbage flag, the crossing over flag, the type flag* and the middle marker. The part for storing the number of a round will be big enough so that it can store numbers up to $8K$ (this requires only $O(\log m)$ bits).

We require that the flags and, of course, the middle markers, be in a constant distance from the crossing over locus. In the simulation information must be passed from one simulated tape segment to the other one and from clocks to flags. Since a finite set of contexts cannot be used to jump over more than constant length segments we arrange the clock bits and simulated tape bits so that they interleave regularly. In order to distinguish these two kinds of bits we use two different versions of each of the distinguishing words w^+_{pro} and w^-_{pro}. Otherwise the particular layout of proper tapes does not matter.

Simulated tapes will be encoded in the two parts reserved on the half-tapes. Namely the left part encodes the left half-tape of the simulated tape and the right part encodes the right half-tape of the simulated tape. Let $g \in A^n$ be a proper tape with g_1 the left half-tape and g_2 the right half-tape, then we denote by $H(g)$ the tape $h \in A^m$ coded by g and by $H_1(g_1)$, respectively $H_2(g_2)$, the left, respectively right, half-tape of coded by g_1, respectively g_2. We assume that the bits of $H_i(g_i)$ are just certain bits of the part called simulated tape i of g.

The garbage and cross-over flags will have two values each, 0 meaning *off* and 1 meaning *on*. The type flags will have three values $0, 1, 2$, which will be encoded by the $\{0, 1\}$ alphabet. Initially on each proper tape the two types are equal and all three possibilities occur with the same frequency.

The main complication is that we cannot enforce that all the tapes will simulate the same generation of the original tapes. So we have to encode the information about the generation into each simulating tape. This information will be used to avoid interactions between simulating tapes which simulate different rounds of M. This is the reason for using the flags and the numbers of rounds.

Another complication of a similar nature is that the crossing over is not so efficient if some tapes are not allowed to cross-over. Thus we shall simulate each single crossing over step by several generations and use estimates derived in Section 6.3.

According to this plan we shall distinguish tapes as being in two possible phases: *rewriting phase* and *crossing over phase*. As auxiliary tapes do not interact, we only need to describe interactions of proper tapes with proper tapes and interactions of proper tapes with auxiliary tapes. The interaction will depend mainly on the phase. In the rewriting phase a proper tape interacts only with auxiliary tapes and it does it in such a way that it simulates a Turing machine. Thus we shall describe it as a work of a Turing machine on the tape. In the crossing over phase a proper tape interacts both with proper tapes and auxiliary tapes. Again the interaction with auxiliary tapes is a simulation of a Turing machine. Let us recall that we shall use a conservative GTM, hence we can think of the system as if the tapes evolved.

In both phases we shall use clocks. The clocks are simply Turing machines which make a certain number of steps and then they switch flags. The number of steps is a constant depending only on m whose value will be determined below in the computation. Note that each clock is entirely on one half-tape and it advances only using some auxiliary tapes, hence the running time of the clock is not influenced by crossing over in the crossing over locus. Due to the use of two separate classes of auxiliary tapes the two clocks on a proper tape in a rewriting phase run independently. The precise meaning of these intuitive statement will be explained in the next subsection.

Here is a description in more details. Fix positive integer parameters Δ and C whose value will be determined later. In the initial population all proper tapes simulate the initial simulated population. Namely the relative frequency of proper tapes g such that $H(g) = h$ is equal to the frequency of h in the initial simulated population. The numbers of rounds and the clocks are set to 0. The garbage flags are off, the crossing over flags are on. The two type flags are equal on each proper tape and the three possibilities occur with the same frequency independently of the remaining content of the tapes. The simulation starts with a crossing over phase and then the crossing over and rewriting phases alternate.

Rewriting phase. Let g be a proper tape with half-tapes g_1 and g_2. The phase will start when both crossing over flags are turned off. Switching the second crossing over flag off will initiate rewriting which will simulate a Turing machine N_0 which does the following:

1. runs left clock for 16Δ time units (=number of bit rewritings); this will be called *the 1-st synchronization phase*;
2. it checks if the two round numbers are the same;
3. it checks if the types of the half-tapes are different;
4. if both true, then continues, otherwise it puts the garbage flags on and stops;
5. it simulates one step of the computation of the machine M_0 on the tape $(H_1(g_1), H_2(g_2)) \in A^{2m}$;
6. it increments the number of a round by one in both halves;
7. it sets the clocks to zero;
8. it rewrites both type flags to k, where k is the unique element of $\{0,1,2\}$ different from the types of the two half-tapes;
9. it runs the left clock for 2Δ time units; this will be called *the 2-nd synchronization phase*;
10. sets the crossing over flags on.

Note that the machine N_0 must be designed so that it starts its computation in a constant distance from the crossing over locus and the last action of it is to set the second crossing over flag on. In this way it is ensured that there is always exactly one bit on a tape in a rewriting phase which can be rewritten. We shall run the simulation only for a limited number of steps, so we can take size of the number of rounds registers so big that the machine never reaches the maximal value.

Crossing over phase. The phase will start when both crossing over flags are turned on. The half-tapes contain images of simulated half-tapes. Switching the crossing over flag off will initiate rewriting which will simulate a Turing machine $N_{1,l}$ on the left half-tape and a Turing machine $N_{1,r}$ on the right half-tape. Each of the machines does the following:

1. it advances its clock,
2. when the time C (the time reserved for crossing over) is reached on the clock, then it switches the crossing over flags off.

Again we assume that rewriting of the clocks starts and ends near the crossing over locus, so that there are always exactly one bit on the left half-tape and exactly one bit on the right half-tape which can be rewritten.

Furthermore proper tapes will cross-over if certain conditions are satisfied. The conditions for crossing over are given by appropriate flag settings of the flags of the two tapes:

1. the garbage flags of both tapes are off,
2. the crossing over flags of both tapes are on,
3. the type flags are as described in Section 6.3, i.e., on one tape they are i, i and on the other j, k with $i \neq j, k$.

As the flags are in constant distance from the crossing over locus which is determined by the crossing over marker, these conditions can be defined as a finite set of contexts.

Garbage tapes. Once some garbage flag is set on, the proper tape will not interact with other tapes.

We require that switching the garbage and crossing over flags is done always in one step. This is easy to accomplish by one rewriting (i.e., crossing over with an auxiliary tape), since these flags can be coded by single bits. When switching from a rewriting phase to a crossing over phase we need to switch both crossing over flags, so this is done in two steps.

Note that we use two crossing over flags and two garbage flags only in order to have some symmetry between the half-tapes. We could also do with only a single crossing over flag and a single garbage flag.

We denote by R the number of rewritings needed to complete a rewriting phase for the machine N_0. C is the number of rewritings needed to complete a crossing over phase for the machines $N_{1,l}$ and $N_{1,r}$; we assume that both machines need the same time. These numbers depend only on the input size, they do not depend on the round of the computation.

Let us observe that always 1/8 of the proper tapes in a rewriting phase and 1/8 of the half-tapes in a crossing over phase will be rewritten (the factor is 1/8 instead of 1/4 as we have different auxiliary tapes for different halves).

6.5 Simulation of general genetic Turing machines by crossing over – computation

First we shall prove that the simulation is correct in the sense that the relative frequencies of tapes with fixed additional information are the same as the frequencies of simulated tapes at some stage. Then we shall show that in each generation of the simulation almost all proper tapes simulate original tapes of some particular generation. However, to prove that we have to show that also half-tapes simulate the corresponding halves in crossing over generations. Recall that given a population of tapes, the frequency of a right (resp. left) half-tape g is the sum of the frequencies of tapes of the form (g, h) (resp. (h, g)).

We call a proper tape *synchronized*, if both numbers of rounds are the same. A proper tape is *simulating* if it is synchronized and not garbage. A half-tape is simulating, if it is a part of a non-garbage proper tape in a crossing over phase; we do not require that it is synchronized. In fact, unsynchronized tapes may cross-over to produce synchronized tapes again. Note that once an unsynchronized tape enters rewriting phase its garbage flag will be switched on before it can enter another crossing over phase. An unsynchronized tape has necessarily different types, therefore, according to the rules, it can cross-over only with a tape with both types equal, hence synchronized.

We will define a parameter of a proper tape, resp. half-tape, which determines the simulated generation and which also enables us to separate the information about the simulated from the rest. The *age* of a simulating tape g in a rewriting phase is the triple $(0, r, j)$ where r is the number of the round and j, $0 \leq j < R$ is the number of steps that N_0 needs to produce g from a tape obtained in a crossing over phase. The age of a left (resp. right) half-tape in a rewriting phase (means rewriting flag *on*) is a triple $(1, r, j)$ where r is the number of the round and j, $0 \leq j < C$ is the number of steps that $N_{1,l}$ (resp. $N_{1,r}$) needs to produce g from a tape obtained in a rewriting phase, (i.e., j is essentially the time on the clock). Let us note that this is a correct definition, since $N_0, N_{1,l}$ and $N_{1,r}$ always use the same time on any initial configuration before they stop. Hence they cannot reach an intermediate configuration using computations of different lengths.

Recall that we have two versions for each half-tape – the positive one and the negative one, thus four versions of proper tapes. All four versions occur with the same frequency, so we can ignore the distinction between them. Furthermore, we have three types for each half-tape. So each half-tape g has six possible types storing the same information. The crossing over rules ensure that the symmetry between positive and negative tapes is preserved in each generation. In the same way the symmetry is preserved for types in the sense that we can permute the types $\{0, 1, 2\}$ without affecting the frequency. Note, however, that the ratio of those tapes with both types equal to those with different types on half-tapes will vary. By the definition of the simulating operator the proper tapes which will appear in the simulation with nonzero frequency will be only tapes which can be produced from initial tapes using computations of the machines N_0, $N_{1,l}$ and $N_{1,r}$ and the crossing over described above.

We describe explicitly how the types are changed during the rewriting phase. To change a pair (i, j) to (k, k) (where $\{0, 1, 2\} = \{i, j, k\}$) we first mark k to a separate place, then erase successively i and j, then write (k, k) to the appropriate position and finally erase the extra stored k. Until both i and j are erased, the information about them is present, so we shall think of the tape as being of type (i, j). After that we shall say that it is of type (k, k).

Observe that each rewriting either simulates rewriting of the simulated tape, or advances a clock, or switches a flag. Thus we get:

Fact *A simulating tape g in a rewriting phase is uniquely determined by the simulated tape $H(g)$, the age, the types of the half-tapes $(0/1/2)$, and the versions of the half-tapes (positive/negative). Left (resp. right) half-tape g_1 (resp. g_2) in a crossing over phase is uniquely determined by the simulated half-tape $H_1(g_1)$ (resp. $H_2(g_1)$), the age, the type and the version.*

In order to have simpler correspondence between the age of a simulating tape and the number of the simulated generation, we assign the *age* to the *simulated generation* of

M in a similar way. Thus the round r will have ages $(0,r,0),(0,r,1),(0,r,2),\ldots,(0,r,T-2)$ in the rewriting phase, which correspond to the computation steps of M_0, then there will be just one crossing over age $(1,r,0)$. After crossing over follows age $(0,r+1,0)$ and so on. Since the simulating machine N_0 does more than M_0 (it checks the flags and the numbers of rounds), it will need more steps to simulate M_0, however only polynomially more. Let s be the function such that $s(j)$ is the number of steps of N_0 which have been simulated after j steps of N_0. E.g. during the checking and synchronization periods the function will be constant.

We shall show that if we fix all parameters (age, types, versions) of a simulating tape in a rewriting phase, then its frequency is equal to the frequency of the simulated tape in the corresponding simulated generation. For half-tapes, we need a slightly stronger statement, namely that this is true even if we fix the other half-tape.

Lemma 6.6 *Fix integers $1 \leq r \leq K, 0 \leq t \leq 8K$. Consider only positive half-tapes and proper tapes with both half-tapes positive (the same holds true for the other versions of half-tapes and proper tapes).*
(1) Let furthermore an age $(0,r,k)$, $0 \leq k < R$, and a pair of types (i,j) be fixed. Suppose that in the t-th generation the frequency of such simulating tapes is nonzero. Then the relative frequency of such a tape $g \in A^n$ among all tapes of this age and type in generation t is equal to the frequency of the simulated tape $H(g) \in A^{2m}$ in generation $(0,r,s(k))$.
(2) The same holds for an age $(1,r,k)$, $0 \leq k < C$ and tapes of type (i,i).
(3) Fix a right half-tape g_2 in a crossing over phase, a type i different from the type of g_2 and an age $(1,r,k)$, $0 \leq k < C$. Suppose that in the t-th generation the tapes which are in crossing over phase and whose left half-tape has age $(1,r,k)$ and whose right half-tape is g_2 occur with a nonzero frequency. Then the relative frequency of a half-tape g_1 from such proper tapes among all such half-tapes is equal to the frequency of the left half-tape $H_1(g_1)$ in the simulated population in the generation of age $(1,r,0)$.
(4) The same holds for right half-tapes.
(5) Let furthermore an age $(1,r,k)$, $0 \leq k < C$, and a pair of types (i,j) be fixed, where $i \neq j$. Suppose that in the t-th generation the frequency of such simulating tapes is nonzero. Then the relative frequency of such a tape $g \in A^n$ among all tapes of this age and type in generation t is equal to the frequency of the simulated tape $H(g) \in A^{2m}$ in generation $(0,r+1,0)$.

(Let us remark that the distribution of left half-tapes is the same as the distribution of right half-tapes in the simulated system, so the same will be true about the simulating system. However we shall not use this property in proving the correctness of the simulation.)

Proof. We shall prove the lemma by induction on t.

For $t = 0$ all proper tapes of N code the tapes of the machine M in the initial configuration. Then the relative frequency of a g among proper tapes is equal to the frequency of $H(g)$ by definition.

Let $t > 1$. Let an age $(0,r,k)$ and a pair of types (i,j) be fixed. If $r = 1, k = 0$, then tapes of age $(0,r,k)$ are the initial tapes. As they cannot be produced from others, they are the remainder of those which there were at the 0-th generation. Since rewriting does not depend on the content of the tape their frequencies will decrease at the same speed.

Now consider an age $(0, r, k)$ with $k > 0$. Using the fact that rewriting does not depend on the content of the tapes we infer that the frequency of such proper tapes is $7/8$ of their frequencies in generation $t-1$ and $1/8$ of the frequencies of the proper tapes of age $(0, r, k-1)$ in generation $t-1$. Thus the statement (1) follows from the induction assumption for the ages $(0, r, k)$ and $(0, r, k-1)$. If $k = 0$ and $r > 1$ we use the induction assumption for $(0, r, k)$ and $(1, r-1, C-1)$.

The same argument proves the induction step for an age $(1, r, k)$, $0 \le k < C$ and tapes of type (i, i), i.e., statement (2).

Similar argument can be applied to half-tapes in crossing over phase. Fix an age $(1, r, k)$ and a right half-tape g_2. Let g_1 be a left half-tape of age $(1, r, k)$. Then the tape $g_1 g_2$ was produced from tapes of generation $t-1$ in one of the following four ways:

1. by crossing over $g_1 h$ with $g g_2$, for some g, h;
2. by rewriting the left clock in some $g_1' g_2$;
3. by rewriting the right clock in some $g_1 g_2'$;
4. from the same tape $g_1 g_2$ which did not interact.

In all four cases the operation does not depend on the simulated half-tape $H_1(g_1)$. Thus we only need to check that the frequency of the half-tape is correct in generation $t-1$. In the first case, if the type of $g_1 h$ is (i, i), it follows from the induction assumption (2). If the type is (i, j), for some $j \neq i$, and in all other cases we use the statement (3).

(4) follows by symmetry.

As shown above (Section 6.3), the type flags ensure uniform mixing of half-tapes. This means that in the subpopulation of tapes in crossing over phase with different types on the half-tapes, the frequency of a tape is the product of the relative frequencies of the half-tapes. Clearly this ensures that the subpopulation of synchronized tapes will also be uniformly mixed. This and the induction assumption give the statement (5) exactly in the same way as above. ■

Now we have to set parameters of the construction and prove that simulating tapes have frequency bounded from below by a positive constant.

First we observe that the ratio of the number of the simulated rounds K to the length of the rewriting phase R can be an arbitrary polynomial. This is because, on the one hand, we can artificially increase R by letting the machine N_0 just to count to a given number, or, on the other hand, we can simulate more rounds and thus increase K. We set

$$\Delta = K^4, \quad R = 64K^4, \quad C = 4K^4.$$

A proper tape will need $8R$ generation for rewriting in the average; a half-tape will need $8C$ generation to switch the crossing over flag off. Thus a *typical* number of generations for a round will be $8(R + C)$. However we need to know that *most* of the proper tapes are in some definite state. Therefore we shall split the time scale differently. Let

$$\begin{aligned} t_r &=_{df} 8(R+C)(r-1), \\ t_r^{(0)} &=_{df} t_r + 8\Delta, \\ t_r^{(1)} &=_{df} t_r + 8R - 8\Delta, \\ t_r^{(2)} &=_{df} t_r + 8R + 8\Delta, \\ t_r^{(3)} &=_{df} t_{r+1} - 8\Delta. \end{aligned}$$

Furthermore we define error parameters:

$$\Delta_r^{(i)} =_{df} \frac{4(r-1)+i+1}{4} K^3, \quad r = 1, \ldots, K, \quad i = 0, 1, 2, 3.$$

Note that $\Delta_r^{(i)} \leq \Delta$ for all r, i in the given range. We shall say that a tape has an age $(i, r, t \pm d)$, if it has age (i, r, s) for some s such that $t - d \leq s \leq t + d$.

Lemma 6.7 *There exists an $\varepsilon > 0$ such that for every sufficiently large n and*

$$\varepsilon_r^{(i)} =_{df} \exp(-\varepsilon K^2 + 2(4r + i))$$

the following holds for $r = 1, \ldots, K$.

(1) In generation $t_r^{(0)}$ at least $1 - \varepsilon_r^{(0)}$ proper tapes simulate generation $(0, r, 0)$ and have age $(0, r, \Delta \pm \Delta_r^{(0)})$.

(2) In generation $t_r^{(1)}$ at least $1 - \varepsilon_r^{(2)}$ proper tapes simulate generation $(1, r, 0)$ and have age $(0, r, R - \Delta \pm \Delta_r^{(1)})$.

(3) In generation $t_r^{(2)}$ at least $1 - \varepsilon_r^{(2)}$ proper tapes have both half-tapes of age $(1, r, \Delta \pm \Delta_r^{(2)})$ and simulate half-tapes of generation $(0, r+1, 0)$.

(4) In generation $t_r^{(3)}$ at least $1 - \varepsilon_r^{(3)}$ proper tapes simulate generation $(0, r+1, 0)$ and both their half-tapes have age $(1, r, C - \Delta \pm \Delta_r^{(3)})$.

We shall use the following trivial corollary of Chernoff's bound.

Lemma 6.8 *Let $X, Y_1, \ldots, Y_t$ be independent random variables, $Y_1, \ldots, Y_t$ Bernoulli variables with mean α. Let $a_1 \leq a_2$ and $\Delta > 0$ be given. Then*

$$\mathrm{Prob}(a_1 + \alpha t - \Delta < X + \sum_{i=1}^{t} Y_i < a_2 + \alpha t + \Delta) \geq \mathrm{Prob}(a_1 < X < a_2) - 2\exp\left(-c_\alpha \frac{\Delta^2}{t}\right).$$

where the constant c_α depends only on α.

Proof.

$$\mathrm{Prob}(a_1 + \alpha t - \Delta < X + \sum_{i=1}^{t} Y_i < a_2 + \alpha t + \Delta) \geq$$

$$\mathrm{Prob}(a_1 < X < a_2 \text{ and } \alpha t - \Delta \leq \sum_{i=1}^{t} Y_i \leq \alpha t + \Delta) =$$

$$\mathrm{Prob}(a_1 < X < a_2) \cdot \mathrm{Prob}(\alpha t - \Delta \le \sum_{i=1}^{t} Y_i \le \alpha t + \Delta) \ge$$

$$\mathrm{Prob}(a_1 < X < a_2)\left(1 - 2\exp\left(-c_\alpha \frac{\Delta^2}{t}\right)\right) \ge \mathrm{Prob}(a_1 < X < a_2) - 2\exp\left(-c_\alpha \frac{\Delta^2}{t}\right).$$

■

Proof of Lemma 6.7.

Let $\varepsilon > 0$ be sufficiently small and n sufficiently large. The actual bounds can be easily computed from the bounds below. We shall use induction. As in the statement of the lemma, we shall count the relative frequencies among the proper tapes.

1. Consider the generation $t_0^{(0)} = 8\Delta$. Then all proper tapes are still in the first rewriting phase, so they have ages of the form $(0, 1, t)$. As shown above, t has binomial distribution $B(8\Delta, 1/8)$. Hence the frequency of those which are in the interval $\Delta \pm \Delta_0^{(0)}$ is at least

$$1 - 2\exp\left(-c_{1/8}\frac{(\Delta_0^{(0)})^2}{\Delta}\right) = 1 - \exp\left(-\frac{c_{1/8}}{16}K^2 + \ln 2\right) \ge 1 - \exp\left(-\frac{c_{1/8}}{16}K^2 + 2\right),$$

which is at least $1 - \varepsilon_0^{(0)}$, if ε is sufficiently small. All these tapes are still in the first synchronization phase (i.e., only the clock is running), hence, by Lemma 6.6, they simulate the initial population.

2. Consider the generation $t_r^{(1)}$. Suppose the statement (1) holds true for $t_r^{(0)}$. As above, we can think of tapes in a rewriting phase as being randomly independently rewritten with probability 1/8. One rewriting means advancing the age by one unit. Thus it suffices to estimate the contribution of the proper tapes whose age was $(0, r, \Delta \pm \Delta_r^{(0)})$ in the generation $t_r^{(0)}$ to the frequency of tapes of age $(0, r, R - \Delta \pm \Delta_r^{(1)})$ in the generation $t_r^{(1)}$. Using Lemma 6.8 we get that this frequency is:

$$\ge 1 - \varepsilon_r^{(0)} - 2\exp\left(-c_{1/8}\frac{(\Delta_r^{(1)} - \Delta_r^{(0)})^2}{t_r^{(1)} - t_r^{(0)}}\right) \ge 1 - \varepsilon_r^{(0)} - 2\exp\left(-\frac{c_{1/8}}{1984}K^2\right). \tag{17}$$

If ε is sufficiently small, it is

$$\ge 1 - 3\varepsilon_r^{(0)} = 1 - 3\exp(-\varepsilon K^2 + 2 \cdot 4r) = 1 - \exp(-\varepsilon K^2 + 2 \cdot 4r + \ln 3) \ge$$

$$1 - \exp(-\varepsilon K^2 + 2 \cdot 4r + 2)) \ge 1 - \varepsilon_r^{(1)}.$$

This gives us the statement (2).

3. We would like to use the same argument as above for $t_r^{(2)}$, but the age of a half-tape is not defined during the rewriting phase. We are interested only in tapes which are descendants of the proper tapes which had age $(0, r, R - \Delta \pm \Delta_r^{(1)})$ in generation $t_r^{(1)}$ and we want to investigate them in the interval $[t_r^{(1)}, t_r^{(2)}]$. Such tapes are in the second synchronization phase of the rewriting phase (i.e., only the left clock is running) and they gradually enter the crossing over phase. In this period the *simulated tape* parts of the half-tapes do not change. For the left half-tapes of the tapes which are in the

second synchronization phase we can easily extend the concept of the age, since the left clock is used in the synchronization phases. Namely, it will be the age of the proper tape whose part they are. To get the statement (3) for left half-tapes we consider the projection of the population to the left half-tapes and argue exactly in the same way as we did in 1 and 2 in case of the proper tapes.

To handle right half-tapes we shall mentally assign a clock to the right half-tapes which are parts of proper tapes in the second synchronization phase. The clock will be identical with the clock on the left half-tape. Then we use the same argument as for the left half-tapes. Mathematically it means that we simulate the population of the right half-tapes by pairs consisting of the half-tape g_2 and and a number t which is the time on a clock. The frequency of a half-tape g_2 is the sum of the frequencies of pairs (g_2, t) for t running over all possible values. g_2 is constant and t is determined by a binomial distribution. When the proper tape enters the crossing over phase, we replace the pair by the actual half-tape.

4. Consider the generation $t_r^{(3)}$. By Lemma 6.6 (5) we know that proper tapes of age $(1, r, C - \Delta \pm \Delta_r^{(3)})$ whose half-tapes have different types simulate tapes of generation $(0, r+1, 0)$. Thus we only need to estimate their frequency.

We shall apply Lemma 6.5. Let u be the distribution of the simulated tapes in generation of the age $(1, r, 0)$, i.e., just before crossing over. Then x is the same distribution with types added, both types the same; y corresponds to the uniformly mixed population of the generation $(0, r+1, 0)$ with two different types added. These distributions will be simulated by the proper tapes which encode this information. Put otherwise, we identify proper types which have the same parts *simulated tape* (we identify different versions with respect to the 0–1 interchange) and the same types of the half-tapes.

Ω, i.e., z, corresponds to the simulating tapes which do not encode tapes of the first two kinds. The meaning of the coefficients c_t, d_t and e_t of Lemma 6.5 is as follows:

- c_t is the fraction of tapes which will reach age $(1, r, 0)$ in in generation t;
- d_t is is the fraction of tapes which in generation t will reach age $(0, r+1, 0)$ while still having the two types equal (these are the tapes which do not manage to cross-over in the time given by the clock) plus the fraction of tapes obtained by crossing over a proper tape in round r and equal types with a proper tape in round r', $r' \neq r$;
- e_t is the fraction of proper tapes of type (j, i) (resp. (i, k)) with both numbers of rounds r which in generation t will result from crossing over tapes of a type (i, i) and rounds (r, r) with tapes of a type (j, k) and rounds (r, r') (resp. (r', r)) with $r \neq r', j \neq i \neq k$.

We shall check that $p_t(g, \Omega; h)$ and $p_t(\Omega, g; h)$ for $g \in E$, $h \in U$, have the form required by Lemma 6.5. Let $\tilde{g}_2$ be a fixed right half-tape in a crossing over phase, with a type $j \neq i$. By Lemma 6.6 (3), the distribution of tapes $(g_1, \tilde{g}_2)$ where the type of g_1 is i and has the number of a round r and $(g_1, \tilde{g}_2)$ is in a crossing over phase, is equal to the frequency of the simulated half-tape $h_1 = H_1(g_1)$ in the generation $(1, r, 0)$, which is $\sum_{h_2} u(h_1, h_2)$. Note that this fact does not depend on the number of round of $\tilde{g}_2$. Fix a proper tape $(\tilde{g'}_1, \tilde{g'}_2)$ in a crossing over phase which has type (i,i). Then we get that the frequency of tapes of the form $(g_1, \tilde{g'}_2)$ obtained by crossing over tapes $(\tilde{g'}_1, \tilde{g'}_2)$ with $(g_1, \tilde{g}_2)$ of the given form, is a constant times $\sum_{h_2} u(h_1,h_2)$. Hence summing over all such $\tilde{g}_2$ we get that the coefficient p_t corresponding to this types is proportional to $\sum_{h_2} u(h_1,h_2)$, as required. The other case follows by symmetry.

We do not have to estimate c_t, d_t and e_t, we only need bounds on γ and $\sum_{t=t_r^{(2)}}^{t_r^{(3)}-1} c_t$. By the induction assumption at least $1-\varepsilon_r^{(2)}$ half-tapes have age $(1,r,\Delta \pm \Delta_r^{(2)})$ in generation $t_r^{(2)}$. Thus for $t_r^{(2)} \le t \le t_r^{(3)}$ at least

$$1-\varepsilon_r^{(2)} - 2\exp\left(-c_{1/8}\frac{\Delta^2}{t}\right) \ge 1-2\varepsilon_r^{(2)}$$

half-tapes have an age $(1,r,t\pm\Delta)$, hence are still in rewriting phase of the r-th round, (assuming ε is sufficiently small and using the same computation as in (17)). Thus the frequency of tapes which are not of this form, *including auxiliary tapes,* is at most $1/2+2\varepsilon_r^{(2)}$, (now we have to consider the frequency among all tapes in order to to compute γ). This is an upper bound on γ, hence for n sufficiently large we have $\gamma \le 2/3$.

Similarly, we can estimate $\sum_{t=t_r^{(2)}}^{t_r^{(3)}-1} c_t$ by the frequency of proper tapes that are not in the cross-over phase of the r-th round in generation $t_r^{(2)}$, since each tape can enter the r-th round only once. Thus

$$\sum_{t=t_r^{(2)}}^{t_r^{(3)}-1} c_t \le \varepsilon_r^{(2)}.$$

Using Lemma 6.5 we can estimate from below the frequency of proper tapes which have unequal types in generation $t_r^{(3)}$ by

$$\ge 1-\varepsilon_r^{(2)} - \left(\frac{2+2/3}{3}\right)^{8(C-2\Delta)} = 1-\varepsilon_r^{(2)} - \exp(-\Omega(K^4)).$$

By the induction assumption and Lemma 6.8 at most $2\varepsilon_r^{(2)}$ half-tapes have the age outside $(1,r,C-\Delta\pm\Delta_r^{(3)})$ in generation $t_r^{(3)}$ (use the same computation as in 1 and 2 and assume that ε is sufficiently small). Hence the frequency of proper tapes one of whose half-tape has age outside $(1,r,C-\Delta\pm\Delta_r^{(3)})$ in generation $t_r^{(3)}$ is at most $4\varepsilon_r^{(2)}$. Thus the frequency of the proper tapes satisfying the condition in the statement (4) of the lemma is

$$\ge 1-5\varepsilon_r^{(2)} - \exp(-\Omega(K^4)) \ge 1-6\varepsilon_r^{(2)} \ge 1-\varepsilon_r^{(3)},$$

for n sufficiently large (and using the fact that $2 \ge \ln 6$ in the last inequality).

5. It remains to prove statement (1) for rounds bigger than 1. Assume that (4) is true for a round r. We shall prove (1) for the round $r+1$.

We would like to use the same argument as we have used above several times. But again there is a problem with the definition of the age. Consider a tape in a crossing over phase. If one of the clocks reaches the time C reserved for crossing over, it will stop. The other may still run and then the age of the tape where the clock has stopped is not defined. As in 3, we assign a virtual clock to such a half-tape. This time the clock will run independently of the other one. Again, the time will have binomial distribution $B(1/8, t)$ after t steps. It will start when the real clock stops and end when the tape reaches the rewriting phase, i.e., when the other clock stops too. The age will be $(0, r+1, t)$, where $r+1$ is the next round and t is the time on the clock. When the tape reaches the next rewriting phase, the time on the virtual clock will be added to the time on the real clock. So the half-tape will have age $(0, r+1, t+s)$ where t is the time on the virtual clock at the moment when the tape starts the rewriting phase and s is the time on the real clock; the other half-tape will have age $(0, r+1, s)$. We are interested only in the descendants of the half-tapes which in generation $t_r^{(3)}$ have age $(1, r, C-\Delta \pm \Delta_r^{(3)})$ and we shall consider them only in the interval $[t_r^{(3)}, t_{r+1}^{(0)}]$. So all such tapes will still be in the crossing over phase of the r-th round, or in the first synchronization phase of the $r+1$-st round. (Recall that the first synchronization phase is 16Δ rewritings long, while $t_r^{(3)} - t_{r+1}^{(0)} = 16\Delta$. Thus half-tapes of the age $(1, r, C - \Delta \pm \Delta_r^{(3)})$ in generation $t_r^{(3)}$ may have descendants in generation $t_{r+1}^{(0)}$ of age at most $(0, r, 15\Delta + \Delta_r^{(3)})$ and $15\Delta + \Delta_r^{(3)} < 16\Delta$.)

Now we can use the same computation (and similar assumptions on ε and n) as above and get that at least $1 - 2\varepsilon_r^{(3)}$ of left half-tapes have age $(0, r+1, \Delta \pm \Delta_{r+1}^{(0)})$ in the generation $t_{r+1}^{(0)}$. The same holds for right half-tapes. Moreover we know from 4, that this is a bound on the frequency of the half-tapes which belong to tapes with the two types different (recall that such tapes cannot cross-over so that the types become the same). Consequently, we have at least $1 - 4\varepsilon_r^{(3)}$ of proper tapes with age $(0, r+1, \Delta \pm \Delta_{r+1}^{(0)})$ in the generation $t_{r+1}^{(0)}$ and which have the two types different. But these are tapes which simulate the generation $(0, r+1, 0)$. A similar argument as above gives

$$1 - 4\varepsilon_r^{(3)} \geq 1 - \varepsilon_{r+1}^{(0)}.$$

Thus we have proved the statement (1) for $r+1$, which finishes the proof of the lemma. ■

To conclude the proof of Theorem 6.1 observe that we only need to simulate the generations $(1, r, 0)$, since only these generations were used to simulate a general GTM in the proof of Proposition 4.1. The simulation of the generations $(1, r, 0)$ have been proved in Lemma 6.8 (2), thus we are done.

7 Conclusions

The computational model of genetic Turing machines is a natural model of parallel computing. We have shown (Theorem 5.1) that it has the power of space bounded Turing machines which is a criterion for a model to be considered parallel. This implies, assuming the plausible conjecture $\mathcal{BPP} \neq \mathcal{PSPACE}$, that in general it is impossible to compute efficiently samples from large populations in genetic like systems. This certainly depends on a given mating mechanism, and for particular systems such a sampling is possible. Here we have considered only one concrete way of mating, context sensitive crossing over, and we have shown that such systems can be as complex as general ones. For further research of this model it is important to find natural phenomena that it can explain.

The most interesting question is, if computation of this kind occur on DNA's in living organisms. For that it would be necessary that crossing over is controlled and depend on the structure of the DNA's in the neighbourhood of the crossing over locus. It is well known that crossing over is not completely random, namely there are sites (the hot spots) where crossing over is much more likely to occur than elsewhere. It is also conceivable that there could be enzymes which could control crossing over using the context. Context sensitive crossing over does not seem to require a process more complex than the synthesis of proteins using the genetic code. The place where we should look for such computations are sequences outside the protein coding DNA ("junk" DNA). On the other hand, crossovers are not very frequent in one meiosis, so such a computational process would be very slow.

Can we at least argue that the use of computation on the molecular level would bring some survival advantage? We do not know. Let us give at least one example. Suppose a species lives in environment that has two possible states A and B. Suppose that there is a gene a that is necessary in order to survive in the state A for long time, but it is deleterious in the state B. Let b an allele with the opposite function. The species needs a to be expressed in environment in the state A. When environment changes to B, gene a will be quickly selected out and, if B lasts long enough, it may become extinct. Then, changing to A will kill the whole species. A good strategy for a is to switch so that it is expressed very seldom, when the environment changes to B and vice versa. This can be achieved by changing the dominance relation between a and b, so that a is recessive in environment B and dominant in A. A phenomenon of this kind has been observed in the classical example of the *industrial melanism* of the peppered moth in England. There the selection favouring darker color not only caused an increase of the frequency of genes for the dark color, but also the genes for dark color became dominant.

A computational explanation of such adaptations could be based on an assumption that a somehow recognizes that the environment is not favorable for a. Namely, in Proposition 3.3 we have shown that an efficiently computable survival operator may influence the population so that the frequency of some gene (which is meanwhile kept constant) is encoded in the genome with high precision. This alone is not sufficient information for a to switch, but the proof, in fact, gives that also the *history of the frequencies* can be encoded in this way. Then a can switch to recessive if the frequencies are decreasing and, vice versa, switch to dominant if they start to increase again. In this example, however, there is a much simpler and much more likely explanation. Namely, there is another pair of genes which controls the dominance relation between a and b. When the selection favours b, it necessarily also favours heterozygotes ab whose control genes make b dominant. Thus in environment B also the frequency of the gene for b's dominance increases. Let us observe that for this explanation we need that the control genes and the pair a, b are not linked (are on different chromosomes) or crossing over occurs frequently on the DNA between them. This might be an explanation for hot spots. This idea has been tested experimentally in a computer simulation in [10]. In any case the mechanism of crossing over and its influence on gene expression is a very interesting subject which deserves more attention.

Most commonly held opinion is that crossing over is inherently random, in spite of the fact that at some loci it is more likely to occur. If so, then the distributed computation on the molecular level is controlled only by the selection operator. This is also the philosophy of genetic algorithms, where mutations and crossing over is usually completely random. This leads to different models, where the stress is put on the selection operator. It is likely that such models have also the power of $\mathcal{PSPACE}$. This is an area of research that we want to pursue in the future.

Genetic Turing machines can be interpreted as a parallel computer architecture with a large number of simple processors where every two processors are connected and in each computational step they are randomly matched into pairs which exchange information. The simulation of $\mathcal{PSPACE}$ in polynomial time on GTM's shows that even such random architecture is very powerful. However this is only a theoretical result. The choice of the random matching requires a lot of random bits, which is considered to be expensive, and it would be very difficult to realize connections between all pairs of processors.

One of the natural phenomena to which our model is also related is the evolution of ideas in human society, say, in a scientific community. There the problems are solved by individuals which for some period of time work independently and then exchange information and this is repeated on and on. The way they choose partners is far from being completely random, but, no doubt, there is a strong element of randomness involved. Also they do not interact always only in pairs. We think that these differences are not very important and that our result hints that there is a big computational power in such a process. Of course, this is only a speculation which can hardly be tested experimentally.

Acknowledgment. I would like to thank to P. Clote, P. Dyan, J. Hartmanis, R. Impagliazzo, T. Pitassi and O. Ritter for their remarks and suggestions concerning an earlier version of the paper.

References

[1] L.M. Adleman, *Molecular computation of solutions to combinatorial problems.* Science 266, 1994, 1021-1024.

[2] S. Arora, Y. Rabani, U. Vazirani, Simulating quadratic dynamical systems is PSPACE-complete (preliminary version). Proc. 26-th ACM Symp. on Theory of Computing, 1994, 459-467.

[3] P. Clote, R. Backofen, Evolution as a computational engine, preprint.

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

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

[6] R.M. Karp, *Mapping the genome: Some combinatorial problems arising in molecular biology.* Proc. 25-th ACM Symp. on Theory of Computing, 1993, 278-285.

[7] S.A. Kauffman, *Origins of Order: Self-organization and Selection.* Oxford Univ. Press 1993.

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

[9] P. Pudlák, *Complexity theory and genetics, (preliminary version).* Proc. 9-th IEEE Symposium on Structure in Complexity Theory, 1994, 383-395.

[10] Pavel Pudlák, Petr Pudlák, *Does dominance-recessiveness have a suvival value?* http://www.ms.mff.cuni.cz/ ppud7490/mysi.ps

[11] Y. Rabani, Y. Rabinovich, A. Sinclair, *A computational view of population genetics (preliminary version).* Proc. 27-th ACM Symp. on Theory of Computing 1975, 83-92.

[12] Y. Rabinovich, *Quadratic Dynamical Systems and Theoretical Aspects of Genetic Algorithms.* PhD thesis, Hebrew University, Jerusalem, 1993.

[13] Y. Rabinovich, A. Sinclair, A. Wigderson, *Quadratic dynamical systems.* Proc. 33-rd IEEE Symp. on Foundations of Computer Science 1992, 304-313.

[14] Y. Rabinovich, A. Wigderson, *Analysis of a simple genetic algorithm.* Proc. 4-th International Conference on Genetic Algorithms, 1991, 215-221.

[15] P. Savický, *Bent functions and random boolean formulas.* Discrete Mathematics 147, 1995, 211-234.

[16] W.D. Stansfield, *The Science of Evolution.* Macmillan, 1977.

[17] D.S. Thaler, *The evolution of genetic intelligence.* Science 264, 1994, 224-225.

[18] L.G. Valiant, *Short monotone formulae for the majority function.* Journal of Algorithms 5, 1984, 363-366.

[19] 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.