# Evolutionary model with Turing machines

Giovanni Feverati[1,*] and Fabio Musso[2,3,†]
[1]*Laboratoire de Physique Theorique LAPTH, CNRS, UMR 5108, associé à l'Université de Savoie, 9, Chemin de Bellevue, Boîte Postale 110, 74941, Annecy le Vieux Cedex, France*
[2]*Dipartimento di fisica, Università degli Studi di Roma TRE and Istituto Nazionale di Fisica Nucleare, Sezione di Roma TRE, Via della Vasca Navale 84, 00146 Roma, Italy*
[3]*Departamento de Física, Facultad de Ciencias, Universidad de Burgos, 09001 Burgos, Spain*
(Received 3 December 2007; published 3 June 2008)

The development of a large noncoding fraction in eukaryotic DNA and the phenomenon of the code bloat in the field of evolutionary computations show a striking similarity. This seems to suggest that (in the presence of mechanisms of code growth) the evolution of a complex code cannot be attained without maintaining a large inactive fraction. To test this hypothesis we performed computer simulations of an evolutionary toy model for Turing machines, studying the relations among fitness and coding versus noncoding ratio while varying mutation and code growth rates. The results suggest that, in our model, having a large reservoir of noncoding states constitutes a great (long term) evolutionary advantage.

DOI: 10.1103/PhysRevE.77.061901 PACS number(s): 87.10.−e

## I. INTRODUCTION

The "*C*-value enigma" refers to the fact that DNA size variation among eukaryote species show no relation to the number of coding genes or to organismal complexity. The discovery that, in eukaryotes, the great majority of DNA is (protein) noncoding, led to a possible solution of the paradox. Indeed, if there exists "noninformative" DNA, then there is no need for a correlation between organismal complexity and the total amount of DNA. On the other hand, there is no consensus on the actual fraction of the noncoding part that has to be considered "noninformative." Moreover, the reason why eukaryotes should accumulate and maintain such a large amount of noncoding DNA is still debated. At least four different theories have been formulated to answer this question: "junk DNA," "selfish DNA," "nucleoskeletal" and "nucleotypical" theory (for a review, see [1], and the references therein). According to the first two theories, in eukaryotes there is an upward mutation pressure acting to increase DNA content without any direct benefit for the "host" (but, on the contrary, with a slight harm). This increasing tendency will continue until the mutation pressure is balanced by natural selection acting on the phenotypic level. The last two theories state, vice versa, that DNA size is directly related to cell size (in a "coevolutionary" or "causative" way, respectively) and it is consequently adjusted by natural selection in such a way to obtain the optimal dimension of the cells.

A common feature of all these theories is that the actual content of the added extra DNA is insignificant. Therefore such DNA can be freely mutated and it can be viewed as a reservoir of raw material for the production of new genes. The relevance of this process for evolution has been emphasized by Ohno [2] in the "junk DNA" hypothesis context and by [3] in the "selfish DNA" one (however, for an opposite point of view, see, for example, [4]).

Evolutionary algorithms are stochastic search methods that mimic the language of natural biological evolution [5]. They operate on a population of potential solutions to a given problem applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and reproducing them using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation. The formal codification of a solution is called its "genome" (or genotype) while its actual behavior is the "phenotype." Fitness is evaluated on the phenotype.[1] Evolutionary calculations often show the phenomenon of the "code bloat," namely, a major growth of the genome size occurring without a significant improvement in fitness [6]. It manifests itself by the presence of regions of code that do nothing and can be mutated or removed without affecting the fitness. We will call these regions "noncoding"; a popular name for them is also "introns," in analogy with those portions in real genes that are not translated into sequences of amino acids. We will call the active regions "coding"; they are also often called "exons." In the context of evolutionary algorithms, code bloat constitutes a major problem; indeed, it can lead to memory or computational time exhaustion before an optimal algorithm has been obtained. To avoid this phenomenon it is usually necessary to introduce a selective disadvantage against larger genomes; however, the selective disadvantage has to be carefully tuned, since a bad choice can drastically slow down the evolution. Many hypotheses have been formulated to explain

*feverati@lapp.in2p3.fr
†musso@fis.uniroma3.it

[1]There is a slight difference among the concept of fitness in the biological and evolutionary algorithms domains. In the former, fitness is associated to a phenotype by measuring the relative number of offspring it has generated; in the latter, on the contrary, an absolute value of fitness is associated to each phenotype and determines the relative number of offspring it will generate

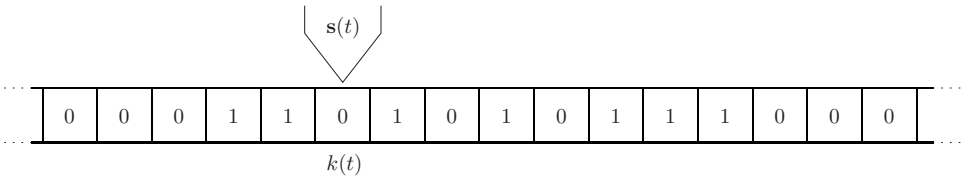



FIG. 1. Graphical representation of a Turing machine at time $t$, in the internal state $\mathbf{s}(t)$, located on the $k(t)$th cell of an infinite tape.

this phenomenon (especially in the framework of "genetic programming") [6]: "hitchhiking," "defense against crossover," "removal bias," and "fitness caused diffusion." The first two impute code bloat to crossover. Indeed, according to the "hitchhiking" hypothesis, noncoding parts attached to "important" parts of code are likely to be propagated in the genome by crossover. On the other hand, for the "defense against crossover" hypothesis, algorithms with large noncoding fractions are selected since they have a smaller probability that their coding parts are disrupted by crossover. The "removal bias" hypothesis ascribes code bloat to the fact that a deletion involving a noncoding part will most probably become disadvantageous if it is larger than the noncoding part itself. On the other hand, an insertion in a noncoding part will always be neutral whatever be the size of the inserted code. So, selection will suppress large deletions but not large insertions, creating a bias in favor of the latter. Finally, up to the "fitness caused diffusion" hypothesis, the number of large-size highly fit programs is much larger than the small-size ones. So code bloat can be described as the evolutionary model going toward its ergodic equilibrium.

While the objects of evolution are very different in the biological and algorithmic contexts, the concepts of survival, reproduction, and selection are very similar so that the name of "artificial life" is given to these "in silico" simulations of life [5]. This analogy makes plausible that some general features can be common to both systems. In particular, the two phenomena of the $C$-value enigma and code bloat appear surprisingly similar: the (genetic or algorithmic) code during its evolution devotes a large part of itself to do, apparently, nothing. With the idea that the evolution of large noncoding parts is a feature of evolutionary models "tout court," we decided to study the relations among fitness and coding versus noncoding ratio for different values of mutation and code growth rates in a particular evolutionary algorithm. The advantage of working with evolutionary algorithms is that we can exactly associate a fitness value to an individual's genotype and we can clearly and unambiguously distinguish its noncoding by its coding parts. We will use Turing machines (TM) to encode our algorithms, identifying the set of their internal states with the genome.[2] Specifically, we start with an initial population of trivial 1-state Turing machines and let them evolve for many generations. At each generation every Turing machine undergoes three processes: mutation (with a rate $p_m$), states increase (with a rate $p_i$), and selection and reproduction[3] (measuring the relative fitness of the machines in accordance with a properly specified task). Each Turing machine state is characterized by two triplets. The triplets of the final population are divided into noncoding and coding triplets. We study how the fitness and the coding versus noncoding ratio vary, changing the values of the mutation ($p_m$) and the states increase ($p_i$) rates. Let us emphasize that ours is a "toy model," namely, it is more a caricature than a description of the real phenomenon of DNA evolution.

## II. WHAT IS A TURING MACHINE?

Turing machines are very simple symbol-manipulating devices, which can be used to encode any feasible algorithm. They were invented in 1936 by Alan Turing [7] and used as abstract tools to investigate the problem of functions computability.

In the following we give a description of Turing machines adapted to our purposes. For a complete treatment of this subject we refer to [8]. The Turing machine consists of a movable head acting on an infinite tape. The tape consists of discrete cells that can contain a 0 or a 1 symbol (Fig. 1). The head has a finite number of internal states. At any time $t$ the head is in a given internal state $\mathbf{s}(t)$ and it is located upon a single cell $k(t)$. It reads the symbol stored inside the cell and, according to its internal state and the symbol read, performs three actions:

(1) "write": writes a new symbol on the $k(t)$ cell,

(2) "move": moves one cell on the right or on the left or stays still $[k(t) \mapsto k(t+1)]$,

(3) "call": changes its internal state to a new state $[\mathbf{s}(t) \mapsto \mathbf{s}(t+1)]$.

Accordingly, a state can be specified by two triplets "write-move-call" listing the actions to undertake after reading, respectively, a 0 or 1 symbol. The following is an example of a possible state:

| | **r** |
|---|---|
| 0 | 1−Right−**3** |
| 1 | 0−Left−**7** |

so that, if the head is in the state $\mathbf{s}(t)=\mathbf{r}$ and reads 0, it writes a 1, moves one cell right, and goes in the state **3**, while

[2]For our aims using TMs is more convenient than using tree based programs as in genetic programming; indeed, we can define independent rates of mutation and code growth, whereas, in genetic programming, mutation also affects genome size.

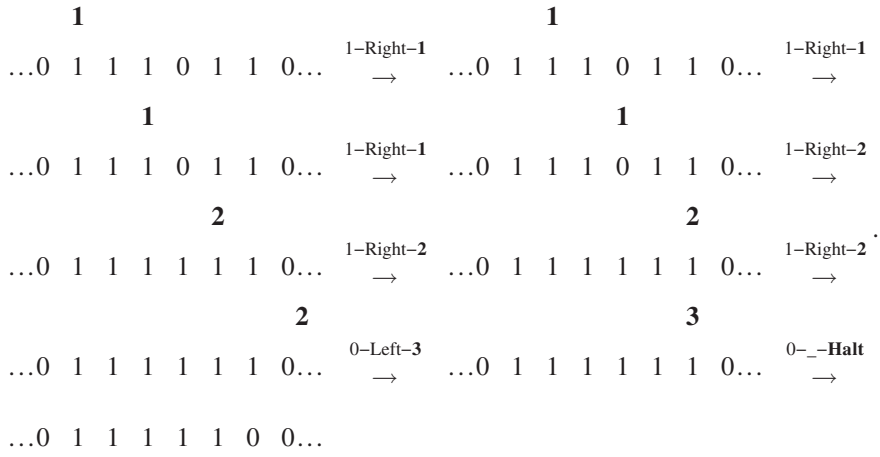
[3]We discarded crossover in this step to keep the model as simple as possible.

if it reads 1, it writes 0, moves left, and goes in the state **7**. There is a distinguished state that causes the machine to halt ("**Halt** state"). The initial tape ($t=0$) specifies the input data for the algorithm encoded by the Turing machine while the final tape (i.e., the one obtained after the machine has halted) gives the output. The head of the machine is initially in the state $\mathbf{s}(0)=\mathbf{1}$ and it is conventionally located upon the cell containing the leftmost 1.

We give here a working example of a Turing machine that performs the sum of two numbers.

| Read**state** | **1** | **2** | **3** |
|---|---|---|---|
| 0 | 1−Right−**2** | 0−Left−**3** | _−_−_ |
| 1 | 1−Right−**1** | 1−Right−**2** | 0−_−**Halt** |

where the underscore symbol means that the corresponding entry can be arbitrarily chosen without affecting the algorithm. A positive integer number $n$ is represented on the tape by a contiguous string of $n$ ones preceded and followed by a zero digit; for example, …01110… represents the number three. Then the sum of 3+2 is performed by the given machine in the following steps:




In the paradigm of artificial life, a Turing machine is an organism; its set of states is the genotype and the output tape is the phenotype. In this frame it is natural to classify as noncoding, those triplets that are never used by the TM, as the third state triplet (_-_-_) corresponding to the read value 0 in the above machine. Notice that whatever input tape is provided, this triplet is never used. In general, the most common case is that the distinction between coding and noncoding triplets depends on the specific choice of the input tape. In this sense the input tape can be interpreted as an extremely powerful epigenetic conditioning that influences the activation of the various triplets of the TM.

The finite state machines used in the context of evolutionary programming (Fogel, 1962 [5]) are a subclass of Turing machines with right only movement. We will not make use of them but cite them as the first appearance of similar devices in the field of artificial life.

## III. MODEL

Since we want to perform computer simulations, we need to use a tape of finite length that we fix to 300 cells. Conventionally, our machines always start from the leftmost cell whatever its content is. If the machine runs out of the tape (both on the left or on the right) it is halted. Since it is quite easy to generate machines that run forever, we also need to fix a maximum number of time steps, therefore we choose to force halting the machine if it reaches 4000 steps.

We begin with a population of 300 1-state TM of the following form:

| | **1** |
|---|---|
| 0 | 0−Still−**Halt** |
| 1 | 1−Still−**Halt** |

and let them evolve for 200 000 generations. At each generation every TM undergoes the following three processes (in this order):

(1) states increase,
(2) mutation,
(3) selection and reproduction.

*State increase***.** In this phase, with a probability $p_i$, the TM passes from **N** to **N+1** states by the addition of the further state

| | **N+1** |
|---|---|
| 0 | 0−Still−**Halt** |
| 1 | 1−Still−**Halt** |

As it will become clear from the definition of the mutation process, this state will be initially noncoding since it cannot be called by any other state. The only way it can be activated is if a mutation in a coding state changes the state call to **N+1**. Notice that, when called, this particular state does not change the tape and halts the machine. Consequently, the activation of this state is mainly harmful or neutral and it can be advantageous only in exceptional cases, so that the TM can benefit from the added states only if they are mutated before their activation.

*Mutation*. During mutation, all entries of each state of the TM are randomly changed with probability $p_m$. The new entry is randomly chosen among all corresponding permitted values, excluding the original one. The permitted values are

(1) 0 or 1 for the "write" entries;
(2) Right, Still, Left for the "move" entries;
(3) the Halt state or an integer from **1** to the number of states **N** of the machine for the "call" entries.

This mechanism of mutation is reminiscent of the biological point mutation. We have not implemented other biological mechanisms such as traslocation, inversion, deletion, etc.

*Selection and reproduction***.** In the selection and reproduction phase a new population is created from the actual one (old population). The number of offspring of a TM is determined by its "fitness" and, to a minor extent, by chance. The fitness of a TM is a function that measures how well the output tape of the machine reproduces a given "goal" tape starting from a prescribed input tape. We compute it in the following way. The fitness is initially set to zero. Then the output tape and the goal tape are compared cell by cell. The fitness is increased by one for any 1 on the output tape that has a matching 1 on the goal tape and it is decreased by 3 for any 1 on the output tape that matches a 0 on the goal tape.

As a selection process, we use what in the field of evolutionary algorithms is known as "tournament selection of size 2". Namely, two TMs are randomly extracted from the old population; they run on the input tape and a fitness value is assigned to them according to their output tapes. The fitness values are compared and the machine which scores higher creates two copies of itself in the new population, while the other is eliminated (asexual reproduction). If the fitness values are equal, each TM creates a copy of itself in the new population. The two TMs that were chosen for the tournament are eliminated from the old population and the process restarts until the exhaustion of the old population.

For our aims, this selection mechanism has many advantages. Since an increase in the fitness is quite rare we are strongly interested in the survival of the best TM, which is automatically granted by the selection mechanism. Moreover, having a very small population (due to computational time reasons) we would like to maintain a maximal "biodiversity," On the other hand, selection implies that TMs with higher fitness have to generate a higher number of offspring, so that they will eventually colonize all the population decreasing the biodiversity. Our selection mechanism ensures that this colonization does not happen too fast. Indeed, the expected number of offspring for the TMs belonging to the best fitness group (those Turing machines that share the best fitness) varies from 1 to 2, depending on the size of this group. The expected number of offspring will be 2 only when there is a single TM in the group and it will progressively decrease to 1 when the group size increases. Finally, it will be exactly one when the best fitness group coincides with the whole population. Another phenomenon limiting biodiversity is genetic drift. In our case such a phenomenon is completely absent, since the tournament preserves both TMs if they score equally.[4] This feature makes this selection procedure also computationally fast, since in the case of an even result the code needs to do nothing.

## IV. SIMULATIONS

In this section we discuss the various choices of parameters adopted for computer simulations with our model. The initial tape was permanently fixed to contain only zeroes. Of course, many other choices are possible, as giving a completely different initial tape or varying it with generations. As we said in Sec. II, the input tape can be interpreted as an epigenetic conditioning. Since we want to keep our model as simple as possible, we decided to keep the input tape fixed. Since we use the symbol 1 to measure the fitness, a tape made entirely of 0 is the most convenient choice. We will define noncoding triplets relative to this choice of input tape. That is, a triplet of a TM will be called noncoding if it is never executed when the TM runs on the input tape made of 300 zeroes. This implies that the values of a noncoding triplet can be arbitrarily changed without affecting the corresponding output tape.

We performed many simulations with different values of the rates $p_i, p_m$, two choices for the "goal" tape and ten choices for the seed of the random number generator.

The states-increase rate has been chosen in the following set of values:

[4]Observe that two TMs can be very different and share the same score.

$$p_i \in \left\{ \frac{1}{10800}; \frac{1}{6000}; \frac{1}{3333}; \frac{1}{1852}; \frac{1}{1029}; \frac{1}{572}; \frac{1}{318}; \frac{1}{176}; \frac{1}{98}; \frac{1}{54}; \frac{1}{30} \right\}. \tag{1}$$

These values were generated starting from the smallest one and requiring an approximate $\frac{5}{9}$ ratio between consecutive numbers. As resulted from some trials, this particular ratio was seen to be the optimal one.

The mutation rate takes the following values:

$$p_m \in \left\{ \frac{1}{20360}; \frac{1}{12339}; \frac{1}{7478}; \frac{1}{4532}; \frac{1}{2747}; \frac{1}{1665}; \frac{1}{1009}; \frac{1}{611}; \frac{1}{371}; \frac{1}{225}; \frac{1}{136}; \frac{1}{83}; \frac{1}{50} \right\} \tag{2}$$

constructed in the same way as $p_i$, but with an approximate ratio of 0.6.

The goal tapes are chosen according to the criterion of providing two difficult and qualitatively different tasks for a TM; in this sense the distribution of the ones on the goal tape has to be extremely nonregular since a periodic distribution is a very easy task for a TM.

We decided to use a goal tape with ones on the cell positions corresponding to prime numbers (with 1 included for convenience) and zeroes elsewhere:

1110101000.1010001010.0010000010.1000001000.1010001000.0010000010.1000001000.1010000010.

0010000010.0000001000.1010001010.0010000000.0000001000.1000001010.0000000010.1000001000,

0010001000.0010000010.1000000000.1010001010.0000000000.1000000000.0010001010.0010000010,

1000000000.1000001000.0010000010.1000001000.1010000000.0010000000.

In the previous expression we inserted a dot every ten cells to facilitate the reading. Our second goal tape is given by the binary expression of the decimal part of $\pi$, namely $(\pi-3)_{\text{bin}}$ as follows:

0010010000.1111110110.1010100010.0010000101.1010001100.0010001101.0011000100.1100011001,

1000101000.1011100000.0011011100.0001110011.0100010010.1001000000.1001001110.0000100010,

0010100110.0111110011.0001110100.0000001000.0010111011.1110101001.1000111011.0001001110,

0110110010.0010010100.0101001010.0000100001.1110011000.1110001101.

Notice that while for prime numbers the ones become progressively rarer so that the task becomes progressively more difficult, in the case of the digits of $\pi$, the ones are more or less equally distributed. Another difference is that prime numbers are always odd (with the exception of 2) so that in the goal tape two ones are separated by at least one zero. On the contrary, the digits of $\pi$ can form clusters of ones of arbitrary length.

According to our definition, the maximal possible value for the fitness is 63 for the prime numbers and 125 for the digits of $\pi$. The program for the simulation has been written in *C* and we used the native random number generator. We tested that its randomness is suitable for our purposes.

We provide an example of a TM obtained after 20 000 generations, with a states-increase rate $p_i=1/3333$, a mutation rate $p_m=1/1009$, and prime numbers task.

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| 0 | 1−Right−**4** | 1−Right−**5** | 1−Right−**6** | 1−Right−**3** | 1−Right−**7** | 0−Right−**2** | 1−Left−**7** | 0−Still−**8** |
| 1 | 1−Still−**3** | 0−Still−**2** | 0−Still−**3** | 1−Right−**3** | 0−Left−**4** | 1−Left−**1** | 0−Left−Halt | 0−Right−**2** |

```
1                                          4
0 0 0 0 0 0 0 0...  1–Right–4 →   1 0 0 0 0 0 0 0...  1–Right–3 →
      3                                    6
1 1 0 0 0 0 0 0...  1–Right–6 →   1 1 1 0 0 0 0 0...  0–Right–2 →
            2                              5
1 1 1 0 0 0 0 0...  1–Right–5 →   1 1 1 0 1 0 0 0...  1–Right–7 →
                  7                        7
1 1 1 0 1 1 0 0...  1–Left–7  →   1 1 1 0 1 1 1 0...  0–Left–Halt →
            H
1 1 1 0 1 0 1 0...
```

We observe that this TM reaches a fitness of 5 with eight states. The eight coding triplets are written in red while the noncoding ones are written in black. Accordingly, the coding triplets amount to 50% of the total triplets. We notice also that this machine writes a 1 in the sixth cell only to come back later and cancel it. So this is clearly not the most economic TM with the same performance. As we will see, this is a typical feature of the TMs obtained through our model, for certain choices $p_i, p_m$.

## V. RESULTS

We start examining the best fitness reached by the population of Turing machines, as shown in Figs. 2 and 3. The most evident effect is the mainly monotonic growth of fitness with the states-increase rate $p_i$ [see, in particular, Figs. 2(c) and 3(c)], occurring at almost all values of the mutation rate. This effect is particularly interesting in combination with the observation that the total number of states of a TM grows approximately proportional to $p_i$ [actually, it is approxi-

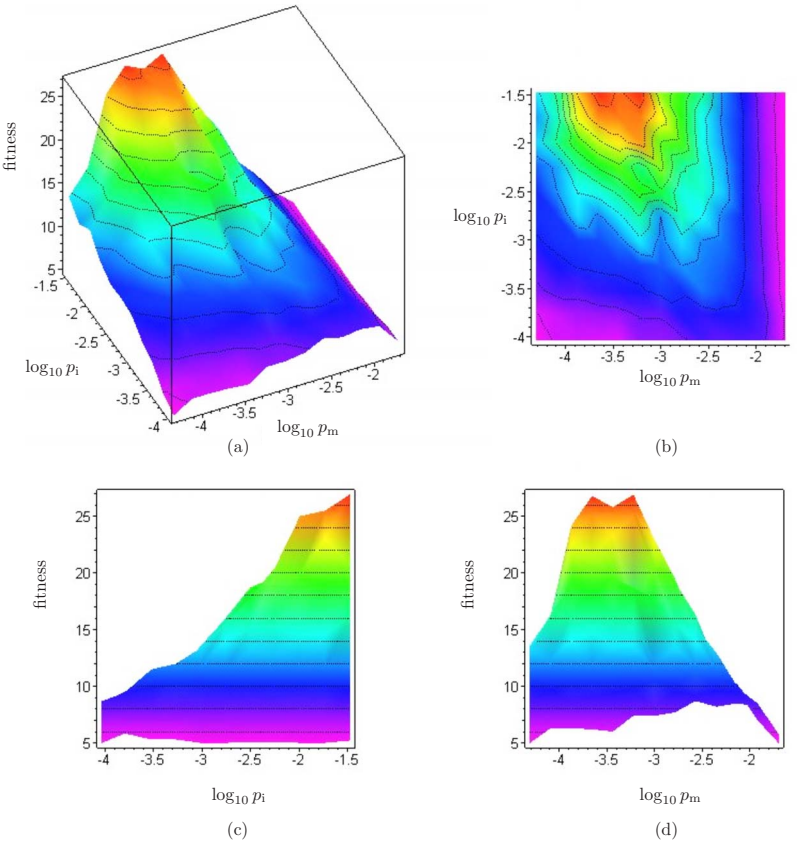



FIG. 2. (Color online) For the case of prime numbers, in (a) we show the 3D plot of the best fitness value in the population, averaged on the ten different seeds, as a function of the states-increase rate $p_i$ and of the mutation rate $p_m$. The three orthogonal projections of (a) are also shown.

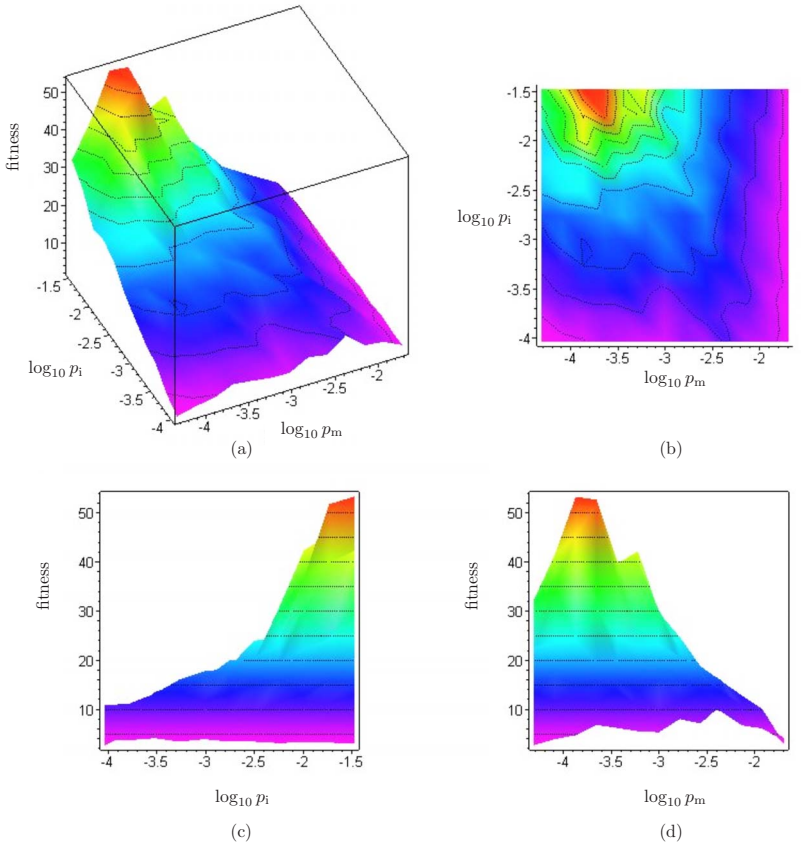



FIG. 3. (Color online) For the case of digits of $\pi$, in (a) we show the 3D plot of the best fitness value in the population, averaged on the ten different seeds, as a function of the states-increase rate $p_i$ and of the mutation rate $p_m$. The three orthogonal projections of (a) are also shown.

mately given by the number of generations times the states-increase rate; see later, Eq. (3)], therefore we state that populations of machines with a larger number of states are those where a higher best fitness can be obtained. Similarly, Figs. 4 and 5 indicate that populations of machines with a larger number of states, i.e., obtained with a larger value of $p_i$, reach a better fitness at all times, namely, after any number of generations. Indeed, for both tasks, the various curves are ordered according to increasing values of $p_i$ and they never intersect. These are strong indications that, in our model, the rate of evolution of a population of Turing machines is directly related to the rate of increase of the number of its states.

Since the maximum values of fitness are obtained at the largest examined values of $p_i$, from Figs. 2 and 3 we cannot state if they are true maxima (either absolute or relative) or

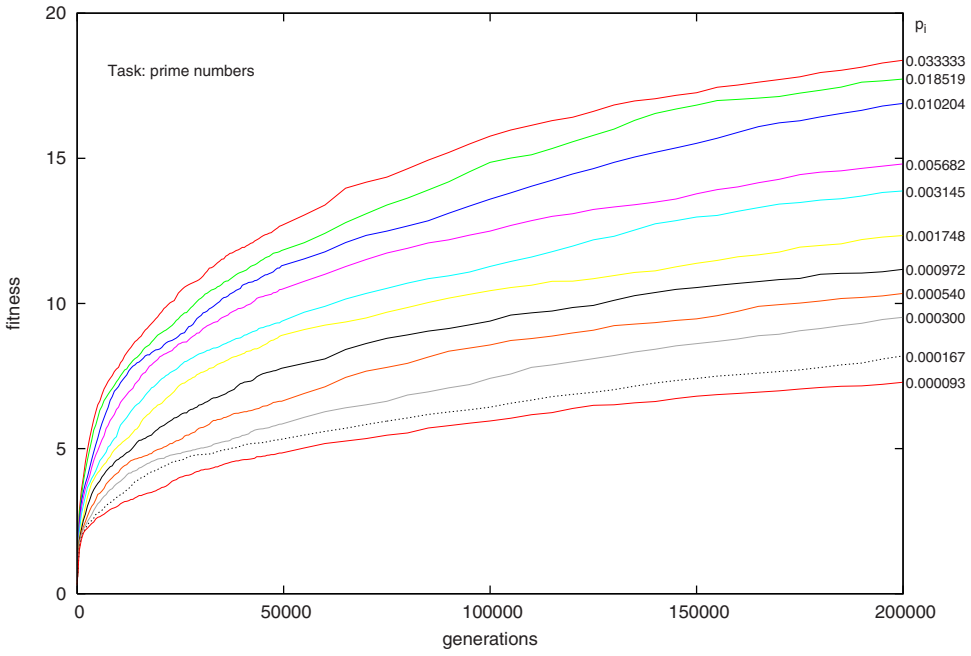



FIG. 4. (Color online) Evolution of the population best fitness with generations, averaged on the seeds and the mutation probabilities.

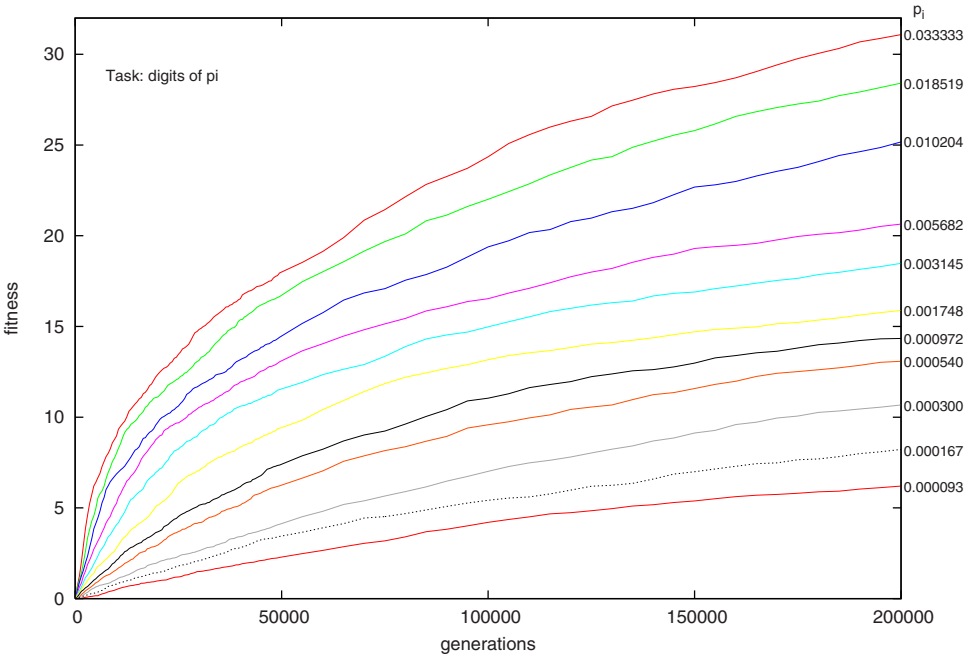



FIG. 5. (Color online) Evolution of the population best fitness with generations, averaged on the seeds and the mutation probabilities.

possibly if a saturation will occur. To understand this would require one to obtain new data with values of $p_i$ larger than $\frac{1}{30}$, but this would largely increase the needed computational time. Indeed, a run with $p_i=\frac{1}{30}$ requires about 100 times the computational time needed for a run with $p_i=\frac{1}{10\,800}$. This great difference is caused by the fact that the number of states of a TM grows approximately as the number of generations times the states-increase rate. In Sec. V A we will test for the existence of a maximum at large values of $p_i$ by performing a simplified simulation.

Another way to test the role of the states-increase rate on fitness is to look at the final number of states. Indeed, since the states-increase procedure alone does not affect the fitness, one could expect that the number of states is not subject to selection. On the other hand, if machines with more states evolve faster, one should expect that the number of states has to be positively selected. We stress the fact that this selection mechanism is completely indirect, in the sense that there is a greater probability that a TM with more states develops an advantageous mutation. In the following Table I we report the observed final number of states $\bar{N}_{\mathrm{obs}}$ averaged on the population, the mutation rate, and the seeds, for all values of the states-increase rate. We also report the number of states $N_{\mathrm{exp}}$ expected under the assumption that it is not a selected character so that it is determined only by chance.

$$N_{\mathrm{exp}} = 200\,000 p_i + 1. \tag{3}$$

Finally, we also indicate the relative difference among the two values,

$$(\%\ \mathrm{diff}) = 100\frac{\bar{N}_{\mathrm{obs}} - N_{\mathrm{exp}}}{N_{\mathrm{exp}}}.$$

From Table I we observe for both tasks that $\bar{N}_{\mathrm{obs}}$ is significantly larger than $N_{\mathrm{exp}}$, for $p_i$ small. This indicates that, in such cases, there is a positive selection on the number of states.[5]

Moreover, it is evident that the relative difference between $\bar{N}_{\mathrm{obs}}$ and $N_{\mathrm{exp}}$ decreases progressively as $p_i$ increases. This decrease is an expected phenomenon, since also the relative standard deviation of the distribution of $N_{\mathrm{exp}}$ decreases while $p_i$ increases.

$$\frac{\sigma}{N_{\mathrm{exp}}} \simeq \frac{1}{\sqrt{200\,000 p_i}}.$$

Finally, we observe that for the primes task the relative differences are larger than those of the digits of the $\pi$ task. This effect is probably due to the fact that it is initially easier for a TM to increase its fitness for the primes task than for the digits of the $\pi$ one, as can be clearly seen from Figs. 4 and 5; this creates a strong selective bias toward TMs with more states for the first generations. This initial easiness basically follows from the three consecutive ones at the very beginning of the primes goal tape.

In Fig. 6 we give a three-dimensional view of the number of coding triplets averaged on the fittest machines in the population and on the seeds, for every $p_i, p_m$. These plots closely resemble those for the fitnesses Figs. 2(a) and 3(a), showing a strict correlation between the two values. In particular, the maximum fitness and maximum number of coding triplets occur at intermediate values of $p_m$, while for both large and small ones the fitness and number of coding triplets rapidly decrease. We give some explanations for this behavior.

[5]Simultaneous mutations in a given TM are allowed. This means that the growth in the number of states cannot act as a shield against harmful mutations.

TABLE I. For the two tasks, we report the observed and expected number of states and their relative difference.

| Task | $p_i$ | $\frac{1}{10\,800}$ | $\frac{1}{6000}$ | $\frac{1}{3333}$ | $\frac{1}{1852}$ | $\frac{1}{1029}$ | $\frac{1}{572}$ | $\frac{1}{318}$ | $\frac{1}{176}$ | $\frac{1}{98}$ | $\frac{1}{54}$ | $\frac{1}{30}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $N_{\mathrm{exp}}$ | 19.5 | 34.3 | 61.0 | 109.0 | 195.4 | 350.7 | 629.9 | 1137.4 | 2041.8 | 3704.7 | 6667.7 |
| Primes | $\bar{N}_{\mathrm{obs}}$ | 23.5 | 37.9 | 64.2 | 113.5 | 199.4 | 355.1 | 633.0 | 1144.2 | 2045.0 | 3701.2 | 6670.1 |
| | % diff | 20.48 | 10.49 | 5.18 | 4.12 | 2.05 | 1.27 | 0.49 | 0.60 | 0.15 | −0.09 | 0.04 |
| $\pi$ | $\bar{N}_{\mathrm{obs}}$ | 21.8 | 36.2 | 63.3 | 111.1 | 197.5 | 356.4 | 631.0 | 1134.6 | 2045.4 | 3710.0 | 6665.6 |
| | % diff | 11.66 | 5.48 | 3.71 | 1.96 | 1.10 | 1.63 | 0.17 | −0.25 | 0.17 | 0.14 | −0.03 |

If the mutation rate is too small, it is clear that there is not enough variation between a machine and its offspring for selection to work on. On the contrary, high mutation rates exert a limiting effect on the maximum number of coding triplets allowed to a TM. Indeed a TM with many coding triplets will most probably undergo a mutation affecting its output tape. Since mutations in coding triplets are probably nefarious, such a TM is likely doomed to extinction.

For the sake of clarity, we report the fitness and the mean number of coding states for the maximum value of the states-increase rate $p_i=\frac{1}{30}$, averaged on the seeds and on the TM with best fitness in the population.

| Task | $p_m$ | $\frac{1}{20\,360}$ | $\frac{1}{12\,339}$ | $\frac{1}{7478}$ | $\frac{1}{4532}$ | $\frac{1}{2747}$ | $\frac{1}{1665}$ | $\frac{1}{1009}$ | $\frac{1}{611}$ | $\frac{1}{371}$ | $\frac{1}{225}$ | $\frac{1}{136}$ | $\frac{1}{83}$ | $\frac{1}{50}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Primes | Fitness | 13.6 | 16.3 | 24.2 | 26.8 | 25.8 | 26.9 | 20.8 | 19.2 | 16.3 | 11.9 | 9.7 | 7.0 | 5.2 |
| | $N_{\mathrm{c}}$ | 85.6 | 130.9 | 257.8 | 291.8 | 239.5 | 239.3 | 140.8 | 94.6 | 61.9 | 35.1 | 22.5 | 13.0 | 11.0 |
| $\pi$ | Fitness | 32.4 | 41.5 | 53.3 | 52.7 | 39.6 | 42.2 | 29.7 | 24.8 | 18.8 | 15.8 | 12.8 | 10.0 | 3.0 |
| | $N_{\mathrm{c}}$ | 237.0 | 338.4 | 487.8 | 424.9 | 337.4 | 273.9 | 171.7 | 104.9 | 66.1 | 40.7 | 26.2 | 25.4 | 19.8 |

It is interesting to read this table together with Fig. 6. This confirms that a TM with a higher mutation rate needs to limit the number of its coding states. Moreover, the table and the figure make evident that equal fitness machines can largely differ in the numbers of coding states (even by a factor larger than 2).

So far we have examined the relation among the number of coding triplets and the fitness. Now we want to elucidate the relation among the number of coding triplets $N_c$ and the number of total triplets $N_t=2N$ (where $N$ is the total number of states). In Table II we report the ratio $\frac{N_c}{N_t}$. The values of $N_c$ and $N_t$ have been obtained averaging on the four values of mutation rate corresponding to the four best fitness scores for any value of $p_i$. We excluded the other values of $p_m$ from the averaging to avoid the already discussed limiting effect on the number of coding triplets that occur at extreme values of the mutation rate. As it is evident from the table, on the

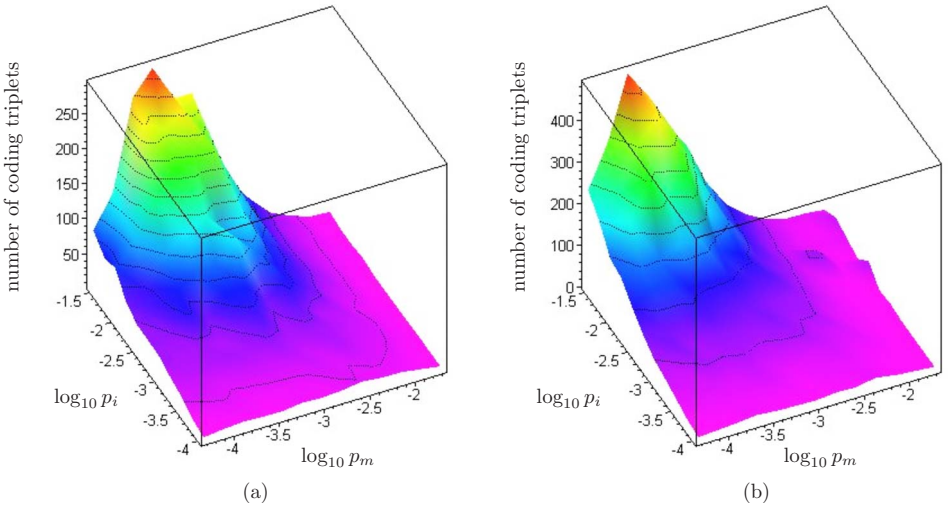



FIG. 6. (Color online) For the prime numbers task (a) and digits of $\pi$ task (b), we show the number of coding triplets averaged on the fittest machines in the population and on the seeds, for every $p_i, p_m$.

TABLE II. For the two tasks, we report the observed ratio of coding triplets versus the total number of triplets.

| Task | $p_i$ | $\frac{1}{10\,800}$ | $\frac{1}{6000}$ | $\frac{1}{3333}$ | $\frac{1}{1852}$ | $\frac{1}{1029}$ | $\frac{1}{572}$ | $\frac{1}{318}$ | $\frac{1}{176}$ | $\frac{1}{98}$ | $\frac{1}{54}$ | $\frac{1}{30}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Primes | $\frac{N_c}{N_t}$ | 0.262 | 0.255 | 0.183 | 0.139 | 0.091 | 0.098 | 0.041 | 0.044 | 0.030 | 0.021 | 0.019 |
| $\pi$ | $\frac{N_c}{N_t}$ | 0.403 | 0.243 | 0.258 | 0.231 | 0.154 | 0.077 | 0.081 | 0.066 | 0.054 | 0.032 | 0.026 |

contrary of what happens for the fitness, the ratio $\frac{N_c}{N_t}$ basically decreases with the states-increase rate.

By plotting the final number of coding triplets versus the states-increase rate on a log-log scale (in Fig. 7), a power-law relation clearly emerges. By fitting the data, we obtained the following relation:

$$N_c = 1.5 \times 10^3 p_i^{0.49} \quad \text{for prime numbers,} \tag{4}$$

$$N_c = 2.5 \times 10^3 p_i^{0.53} \quad \text{for digits of } \pi. \tag{5}$$

On the other hand, the number of states has roughly a linear growth [see Eq. (3)]. This explains the decreasing behavior of $\frac{N_c}{N_t}$ observed in Table II.

The last datum we want to present is the mean fitness increase $\bar{f}_{\text{inc}}$, namely, the final fitness divided by the number of increments. After averaging on the mutation rate, the states-increase rate, and the seeds, we finally obtain

$$\bar{f}_{\text{inc}} = 1.017 \quad \text{for prime numbers,}$$

$$\bar{f}_{\text{inc}} = 1.073 \quad \text{for digits of } \pi. \tag{6}$$

Notice that the minimum theoretical value is 1, meaning that fitness always increases by one. Our values imply that jumps in the fitness larger than one occur but are extremely rare. On this basis, we would say that the evolution of our TMs is more gradualist than saltationist. This is, in some way, surprising since mutations occurring in the state-call entry of the coding triplets almost always give rise to a radical mutation of the TM.

### A. Comparative run

There are two basic arguments that could lead the reader to think that the fitness has to always be a monotonic increasing function of $p_i$ (for all values of $p_i$, regardless of the choice of the target tape).

The first argument is that the fitness growth with $p_i$ has to be considered expected and obvious [6] because the higher the number of states is, the higher is the number of TMs that solve the given task (goal tape). Moreover, each TM can be thought to be contained in a bigger one just by adding some noncoding states while the opposite is clearly false.

The second argument is that, since there is no direct cost associated to the accumulation of noncoding triplets, while, as we will argue in the next section, there is an indirect advantage, again the fitness should always be a monotonic increasing function of $p_i$.

The considerations in the first argument are absolutely correct, but they apply also to bad genomes: there are more large bad genomes than small bad genomes. Given that the system can only experiment a limited (and fixed) number of trials, having a larger set of TMs within which to search just implies that a larger number of trials will fail. In other words, increasing the number of states means that there are more good solutions but does not mean that it is easier to find them.[6]

Our answer to the second argument is that there is also an indirect cost associated with a too fast accumulation of noncoding triplets. This cost is strictly related to the argument given before. Indeed, suppose that there is an optimal solution represented by a TM with $M$ coding triplets. If, at a given generation, the maximum number of states in the TMs population is $N$, during the successive generations the evolutive algorithm will test TMs with a number of coding states distributed between 1 and $N$. If $N<M$, there is no chance of finding the optimal solution. On the other hand, from some value of $N>M$ on, the number of TMs with $M$ coding triplets that will be tested at each generation, will progressively decrease. In other words, the value of $p_i$ determines the "time" that the system has to explore the set of TMs with a given number of coding triplets. If $p_i$ is very large, the system will try a lot of TMs with many coding triplets, risking missing some optimal solutions with fewer ones. We think that this should be a general phenomenon in our evolutive model for any choice of the goal tape. In other words, we think that there is both an indirect advantage and an indirect cost associated with the accumulation of noncoding triplets, whose relative weights depend on $p_i$. Since the maximum fitness is bounded, the indirect advantage must decrease from some value of $p_i$ on; vice versa, the indirect cost will reasonably increase. In conclusion we expect that, in our evolutionary model for TMs, the fitness will have a maximum for some particular value of $p_i$, depending on the particular choice of the goal tape. Here we present the result of some simulations where the goal tape and the number of TMs in the population have been chosen in such a way that the maximum of the fitness is obtained for a value of $p_i$ that we can computationally afford.

In particular, we choose a goal tape with ones in the multiples of 5 positions. There is an obvious 5-states TM that solves this task.

[6]On the contrary, it is natural to think that from some number of states on, it will be extremely more difficult to find them.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 0−Right−**2** | 0−Right−**3** | 0−Right−**4** | 0−Right−**5** | 1−Right−**1** |
| 1 | _−_−_ | _−_−_ | _−_−_ | _−_−_ | _−_−_ |

We also took a very small population of 20 individuals, we limited calculations to 1000 generations, and machines were stopped at 400 time steps. The results of our simulations are summarized in Table III.

As is evident from Table II, the fitness initially grows for small values of $p_i$, then saturates and then decreases at higher values. Similar results have been obtained using as goal tapes the multiples of 2 and 4 (data not shown).

We observe that, for the multiples of 5 goal tape, the TMs use a small number of coding triples (on average, from 5 to 10 triplets), quite unrelated to the fitness or to the mutation and states-increase rates. The reason is that the TMs tend to imitate the 5-states machine indicated earlier (or an equivalent one). Contrarily, in the other simulations we observed a clear relashionship between fitness and the number of coding triplets (see Fig. 7), which indicates that, in general, the TMs that we obtain do not encode periodic (or other compact) algorithms, but try to "guess" a 1 after another [as follows also by the fact that the mean fitness increase is very near to 1; see Eq. (6)].

## VI. CONCLUSIONS AND DISCUSSION

We developed an abstract model, mimicking biological evolution, to understand if there is an "evolutive" advantage in maintaining noncoding parts in an algorithm. We tried to keep the model as simple as possible, while being complicated enough not to allow easy predictions. Moreover we required a model where it would be easy to distinguish between coding and noncoding states in an unambiguous way, with a simple mechanism for the accumulation of noncoding states, where the mutation rate and the states-increase rate should be independent, with a simple mechanism of state activation through mutation. The use of TMs fit perfectly with these requirements.

For the sake of simplicity, we imposed various restrictions on our model that can be relinquished to make the model more realistic from a biological point of view. In particular, we decided that

(1) noncoding states accumulate at a constant rate (determined by the states-increase rate $p_i$) without any deletion mechanism;

(2) there is no selective disadvantage associated with the accumulation of both coding and noncoding states;

(3) the only mutation mechanism is given by point mutation and it also occurs at a constant rate (determined by the mutation rate $p_m$);

(4) there is a unique ecological niche (defined by the target tape);

(5) population is constant; and

(6) reproduction is asexual.

Because of the second point, the TMs we have obtained are not economical in the use of coding triplets. Indeed it is very easy to think of TMs with many fewer coding triplets reaching a better fitness. It could be interesting to make further studies trying to relinquish some of these restrictions. However, letting fall any of these restrictions, would have introduced further free parameters in our model and this was undesirable for two reasons. First, the number of needed simulations increases with the number of free parameters, and second, it becomes more difficult to interpret unambiguously the results.

We decided to use asexual reproduction since crossover would have introduced other possible causes for the accumulation of noncoding states (as in the "hitchhiking" and "defense against crossover" explanations of code bloat) blurring the final conclusions. Analogously, the processes of insertion

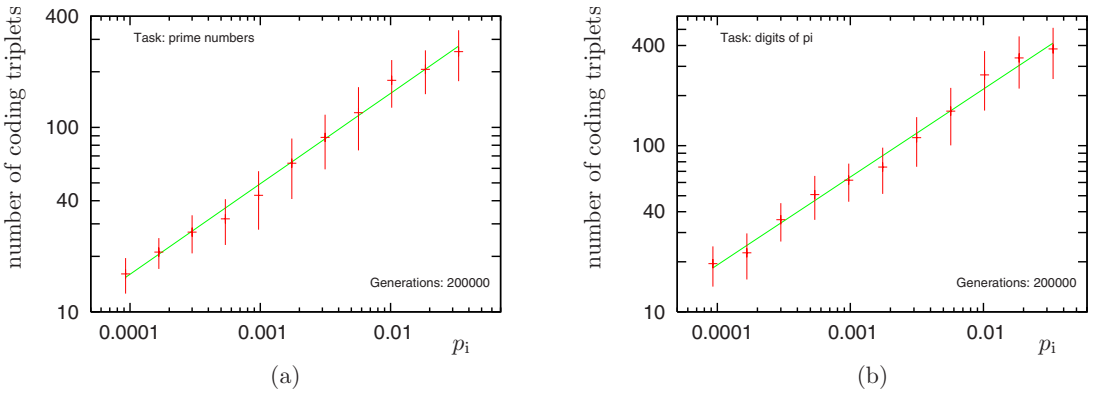



FIG. 7. (Color online) In a log-log scale, we present the relation between the average number of coding triplets $\bar{N}_c$ and the states-increase rate at the last generation. We average $N_c$ on the four values of the mutation rate corresponding to the four best fitness values for each $p_i$.

TABLE III. We report the average fitness obtained after 1000 generations, with a population of 20 individuals, 20 different choices of the seeds, and multiples of 5 as goal tape.

| | | 0.0025 | 0.005 | 0.01 | 0.02 | 0.04 | 0.06 | $p_m$ |
|---|---|---|---|---|---|---|---|---|
| | 0.0022 | 0.1 | 0.1 | 0.3 | 0.2 | 0.5 | 0.1 | |
| | 0.005 | 0.1 | 0.2 | 2.3 | 0.6 | 0.5 | 0.1 | |
| | 0.022 | 0.0 | 3.1 | 3.2 | 2.8 | 0.7 | 0.1 | |
| | 0.05 | 0.0 | 3.2 | 14.0 | 7.4 | 0.3 | 0.0 | |
| | 0.1 | 0.0 | 0.1 | 10.2 | 2.9 | 2.1 | 0.0 | |
| | 0.22 | 0.1 | 0.0 | 5.5 | 6.4 | 0.2 | 0.0 | |
| | 0.5 | 0.0 | 2.1 | 3.1 | 0.2 | 0.1 | 0.0 | |
| $p_i$ | 1 | 0.0 | 4.5 | 3.5 | 0.1 | 0.0 | 0.0 | |

and deletion would have brought the "removal bias" effect. None of this processes has any role in our model. Needless to say, also the processes advocated by the biological explanations ("junk DNA," "selfish DNA," "nucleoskeletal" and "nucleotypical" theory) that we briefly sketched in the Introduction, are ineffective in our model.

In our simulations we started from an initial population completely unadapted to its ecological niche and we observed that, for fixed $p_m$, higher values of $p_i$ correspond to a faster evolution. This behavior is observed for both tasks on the whole range of generations (see Figs. 4 and 5). The same behavior is observed also for the number of coding triplets; the striking similarity between Figs. 2(a) and 6(a), and Figs. 3(a) and 6(b) strongly suggests a direct link between the number of coding triplets and the fitness reached at the end of the evolutive runs. Since the number of coding states is a monotonic increasing function of $p_i$ (see Fig. 7), the natural conclusion is that TMs evolving under higher values of the states-increase rate $p_i$ reach a better fitness since they succeed in generating a larger number of coding triplets. From Table II we see that even if the number of coding triplets grows, they progressively become a very small fraction of the genome as $p_i$ increases (this fact also easily follows from the observation that the number of coding triplets grows approximately as the square root of $p_i$ [see Eqs. (4) and (5)], while the total number of states grows as a linear function of $p_i$ [see Eq. (3)]. So, we argue that the dependence of the fitness on the states-increase rate $p_i$ is realized through the following chain of implications: larger values of $p_i$ imply a larger availability of noncoding triplets that, in its turn, implies a greater probability of enlarging the number of coding triplets. We give the following explanation of the latter statement. During evolution, a mutation occurring in the state-call entry of a coding state can activate a certain subset of noncoding states. Since this activation is probably nefarious, such mutations will almost always lead to the extinction of the TM. Having a population of TMs with a large number of noncoding states allows the system to try to activate a large number of different subsets of them (by means of different mutations in the state-call entries) until a good subset is found. Such an explanation is supported by the fact that the final number of states is a selected character (at least for low values of $p_i$) as discussed in Sec. V.

As we discussed in Sec. V A, we expect that the fitness should have a maximum around some value of $p_i$ and it would be very interesting to determine its actual position for the primes and $\pi$ goal tapes. In such cases, the maximum must lie outside the range of values of $p_i$ that we have considered. Moreover, it is worthwhile to notice that from Figs. 2(c) and 3(c), we seem to be still quite far from this maximum. This means that the optimal coding versus noncoding ratio, in our model, is probably much less than 2%. Exploring the region $p_i > 1/30$ could provide interesting information but is computationally quite expensive.

### A. Back to biology

In this section we put forward some biological speculations inspired by our model. There are two ways of identifying TMs with biological entities and they suggest two ways to which the accumulation of noncoding free to mutate DNA can play a role for "evolvability." In the first one we identify TMs with organisms and coding states with genes. We have to stress that the mechanism of transcription is different in the two contexts. For TMs transcription is serial, so that states must be transcribed, one at a time, in a prescribed order, while in biological organisms transcription of genes can happen in parallel. We can interpret TM states as genes accomplishing both a structural and regulatory function, since a coding state both affects the output tape and specifies which state has to be subsequently transcribed. From this point of view, we can think of TMs in our simulations as organisms increasing their gene pools by the addition of new genes assembled from junk DNA. If the organisms possess more junk DNA it is possible to test more "potential genes" until a good one is found.[7]

On the other hand, we can identify the TMs with single genes and their states as sequences of nucleotides. From this point of view, transcription of states is as serial as the transcription of the nucleotides composing a gene in DNA. The difference now is that transcription of states can jump forward and backward (with respect to the natural order of the states) and passes also more than once through a given state, while transcription of a gene proceeds always in the $5'-3'$ direction. Letting these differences apart, in this framework we think of our simulations as the assembly of a new gene by the addition of new nucleotide sequences. This interpretation enhances the similarity between the activation mechanism in our model and in biological organisms (a similarity that in the previous interpretation is quite loose). Indeed, we can compare our activation mechanism with point mutations or with a deletion of all the stop codons of a gene. In biological organisms the activation of noncoding states can occur through many other mechanisms that are not taken into

[7]This long-term advantage is granted by the accumulation of noncoding states; even if this accumulation is indirectly positively selected in our model (see Table I), it is mainly obtained by increasing the $p_i$ rate, therefore it is not an adaptative phenomenon, namely, is not the effect of natural selection.

account by our model. In our explanation of why TMs with a large number of noncoding states evolve faster, the possibility of jumping in the transcription is essential. Indeed, without this possibility, the TMs could test only the subset of noncoding states just following the last coding state. We notice that this is exactly what happens in prokaryotes, while in eukaryotes the splicing of introns allows one to have jumps in the transcription (at least in the forward direction). So, in the framework of this second interpretation, our model suggests that the mechanism of splicing could have a very significant role for the evolvability of eukaryotes.

**ACKNOWLEDGMENTS**

It is a pleasure to thank M. Caselle, A. Parmeggiani, G. Satta, and S. Pagnotta for their helpful comments regarding this work and for having read the manuscript. G.F. thanks INFN for financial support. F.M. acknowledges support by the LAPTH and by the research training network "EUCLID. Integrable Models and Applications: From Strings to Condensed Matter," Contract No. HPRN-CT-2002-00325. We thank the "Centre de calcul IN2P3-CNRS" and the "EUMEDGRID" INFN project for having allowed us to run our simulations on their grid facilities.

[1] T. R. Gregory, Biol. Rev. Cambridge Philos. Soc. **76**, 65 (2001).
[2] S. Ohno, *Evolution by Gene Duplication* (Springer-Verlag, New York, 1970).
[3] H. K. Jain, Nature **288**, 647 (1980).
[4] T. H. Bestor, Genetica **107**, 289 (1999).
[5] D. B. Fogel, Proc. SPIE **6228**, 622801 (2006).
[6] S. Luke, in *Evolutionary Computation and the c-Value Paradox*, Proceedings of GECCO'05 (ACM, Washington, D.C., USA, 2005).
[7] A. M. Turing, Proc. London Math. Soc. (2) **42**, 230 (1937).
[8] M. Davis, *Computability and Unsolvability* (Dover, New York, 1982).