(Preliminary Version)

Evolutionary Turing Machines – The Quest for Busy Beavers

Penousal Machado, Francisco B. Pereira

Instituto Superior de Engenharia de Coimbra (ISEC),

Quinta da Nora, 3030 Coimbra, Portugal.

{xico, machado}@dei.uc.pt

Telephone: +351 239 790 000

Fax: +351 239 701 266

Jorge Tavares, Ernesto Costa, Amílcar Cardoso

Centre for Informatics and Systems of the University of Coimbra (CISUC),

Pinhal de Marrocos, 3030 Coimbra, Portugal.

{jast, ernesto, amilcar}@dei.uc.pt

Telephone: +351 239 790 000

Fax: +351 239 701 266

Evolutionary Turing Machines – The Quest for Busy Beavers

Abstract

In this chapter we study the feasibility of using Turing Machines as a model for the evolution of computer programs. To assess this idea we select, as test problem, the Busy Beaver - a well-known theoretical problem of undisputed interest and difficulty proposed by Tibor Rado in 1962.

We focus our research on representational issues and on the development of specific genetic operators, proposing alternative ways of encoding and manipulating Turing Machines. The results attained on a comprehensive set of experiments show that the proposed techniques bring significant performance improvements. Moreover, the use of a graph based crossover operator, in conjunction with new representation techniques, allowed us to establish new best candidates for the 6, 7, and 8 states instances of the 4-tuple Busy Beaver problem.

1 INTRODUCTION

In 1937 Alan Turing, while trying to solve one of the problems posed by Hilbert, developed a theoretical computational model that became known as Turing Machines (Turing, 1937). According to the Church-Turing thesis, these finite state machines, although simple, are able to solve any problem that is solvable by an algorithm. Nowadays this thesis is well accepted, and, as such, the class computable and Turing-computable problems are recognized as equivalent.

The main goal of our research is to test the viability of using Turing Machines (TMs) as a model for the evolution of computer programs. More specifically, we propose a framework for the evolution of TMs, and test its performance in a well-known problem, the Busy Beaver (BB). This problem was proposed by Tibor Rado in 1962 (Rado, 1962) and became one of the most famous in the area of Theory of Computation.

In a colloquial way, this problem can be formulated as follows:

> "What is the maximum number of ones that an N-state halting TM can write when started on a blank tape?"

The N-state machine that writes the maximum number of ones is named Busy Beaver.

The rational for choosing the BB problem lies on some of its properties that make it extremely appealing to study the competence of an Evolutionary Computation (EC) algorithm. Some of these properties are:

- It is an undecidable problem. Most approaches that deal with it, try to perform an exhaustive search over the space of possible solutions. We expect that an EC algorithm can discover good quality candidates just by investigating a small part of the search space.
- The search space is very large. For an instance with N states, there are $(4\times(N+1))^{2N}$ possible solutions. Given that the size of the search space depends on the number of states, we can test the scalability of the used algorithms.
- As far as we know, there are no specific heuristics that can help knowledge-based methods to find TMs with high productivity.
- The fitness landscape defined by the BB problem is highly irregular (Pereira, 2002).

- For non-trivial instances, the optimum is not known. This way, development of new methods can lead to the discovery of new best candidates, which adds an additional motivation to the research that is performed.

The formal description of the BB problem and its variants can be found in the following section, which also includes a synthesis of related research.

In section 3 we present an initial evolutionary approach to the BB problem. We start by analyzing previous approaches in which EC techniques are used in the search for Busy Beavers. Subsequently, we describe our initial approach giving emphasis to representation, genetic operators, and fitness assignment issues. The results achieved with this approach are promising, outperforming previous EC approaches.

Encouraged by the success of the initial approach we made modifications in two key components of the EC algorithm – representation and genetic operators – aiming to improve its performance.

In Section 4 we study the influence of representation in the performance of the EC algorithm when applied to the BB problem. We present three alternative ways to interpret the genotype. The results achieved by the different methods show the importance of the interpretation method in the performance of the EC algorithm. We perform an analysis of the experimental results giving explanation to the differences in performance.

In Section 5, we present a recombination genetic operator suited to the manipulation of individuals with a graph structure. We compare the results achieved through the use of this operator with the ones attained by conventional two-point crossover. The analysis of the results allows us to ascertain that it is advantageous to use this new operator.

We used the developed EC algorithm in the search for new Busy Beaver candidates. In Section 6, we present some of the new BB candidates found, and make a brief description of ongoing research efforts.

Finally, in Section 7, we draw some overall conclusions, referring the main contributions of our research work, and point to future research directions.

2 THE BUSY BEAVER PROBLEM

The Busy Beaver problem is directly connected with key issues of the Theory of Computation (Cuttland, 1980; Hopcroft and Ullman, 1979), namely with the existence of non-computable functions and with the halting problem. Since the problem is defined in terms of Turing Machines, we will start by presenting their formal definition.

Definition 1. A deterministic TM can be specified by a sextuple $(Q,\Pi,\Gamma,\delta,s,f)$, where (Wood, 1987):

- Q is a finite set of states
- Π is an alphabet of input symbols
- Γ is an alphabet of tape symbols
- δ is the transition function
- *s* in Q is the start state
- *f* in Q is the final state.

The transition function can assume several forms, the most usual one is:

$$\delta: Q\times\Gamma \rightarrow Q\times\Gamma\times\{L,R\}$$

where L denotes move left and R move right. Machines with a transition function with this format are called 5-tuple TMs. A common variation consists in considering a transition function of the form:

$$\delta: Q \times \Gamma \rightarrow Q \times \{\Gamma \cup \{L,R\}\}$$

Machines of this type are known as 4-tuple TMs. When performing a transition, a 5-tuple TM will write a symbol on the tape, move the head left or right and enter a new state. A 4-tuple TM either writes a new symbol on the tape or moves its head, before entering the new state.

2.1 Problem Definition

The BB problem was proposed by Tibor Rado in 1962 (Rado, 1962). Rado's goal was to present a series of functions that, although non-computable, could be easily defined. The construction of these functions was based on the principle that any finite set of integers has an upper bound. The demonstration of the non-computability of these functions does not require the use of diagonalization techniques (Davis et al., 1994), which is one of the most interesting aspects of Rado's work.

In the original proposal Rado employs 5-tuple TMs with a tape alphabet equal to {**B**, **1**}. Considering TMs of this type, Rado defines productivity of a TM *M*, $\sigma(M)$, as the number of ones present on the initially blank tape when the machine halts. Machines that do not halt have productivity zero. The question raised by Rado is the following:

> "What is the maximum productivity of an N state TM?"

Rado proceeds by defining $\Sigma(N)$ as the function that returns the maximum productivity of an N-state TM, and then by demonstrating its non-computability (Rado, 1962). The N-state TM of maximum productivity is called the Busy Beaver, BB(N).

The halting state is considered an *anonymous* state, i.e. an N-state TM is a TM with N states plus the halting state. In a more formal way we can define Σ(N) as follows:

Definition 2. Defining E_N as the set of all N-state TMs with Γ = {**B**, **1**} and σ: $E_N \rightarrow \aleph$ as the function that returns the productivity of a TM, we have:

$$\forall M \in E_N, \Sigma(N) \geq \sigma(M) \wedge \exists M \in E_N : \sigma(M) = \Sigma(N) \quad \textbf{(1)}$$

We will also define Ω(*N*) as the function that returns the number of steps performed by the Busy Beaver machine.

2.2 Four-Tuple Variant

Boolos and Jeffrey (Boolos and Jeffrey, 1989) presented an important variant of the original problem, defining it for 4-tuple TMs. In so doing they also introduced some changes in the rules. For this type of machine, productivity is defined as:

> "…the length of the sequence of ones produced by the TM when started on a blank tape, and halting when scanning the leftmost one of the sequence, with the rest of the tape blank. Machines that do not halt, or, that halt on another configuration, have productivity zero." (Boolos and Jeffrey, 1989)

Thus, more restrictions are imposed on the 4-tuple variant:

a) When the machine stops there can only be one sequence of **1**'s on the tape.

b) The machine must halt with the read/write head on the leftmost **1** of this sequence.

c) The machine should stop after reading a **1**.

The first two restrictions are relatively common, the last one may cause some surprise. In the model of TM employed by Jeffrey and Boolos there is not a halting state. The TMs stop when there is no defined transition for the current tape/state configuration.

Using this model meeting requirement a) always ensures requirement c). However, if one uses a more common model of TM, with halting state, the requirement c) must be included in order to allow a fair comparison of the results with approaches that use a model with no halting state.

2.3 The Search for Busy Beavers

Due to its difficulty the BB problem has attracted the attention of many researchers, and several contests were organized in an attempt to find new candidates. Although $\sum$(N) is non-computable, it is still possible to determine its value for specific instances of N. In order to prove that a given N state machine is the BB(N), one must prove that no other machine, of N states, has higher productivity.

The most common approaches to determining new BBs and/or to searching for new candidates involve doing a thorough search of the N-state TM space, and simulating the behavior of each machine. This type of approach poses two problems that become harder with the increase of *N*:

- **Dimension of the Space** – Considering TMs with a binary tape alphabet, there are $(4\times(N+1))^{2N}$ TMs with *N* states.
- **Halting Problem** – It is impossible to build an algorithm that determines, for any TM, if the TM stops (Davis et al., 1994).

The first problem can be tackled by increasing the used computational power, or by building faster TM simulators. The second problem is directly related with the non computability of $\sum$, and, as such, cannot be solved.

When part of the search space is undecided (in the sense that it was not viable to determine if some of the machines belonging to that space halt) it is impossible to show

that the most productive machine found is the BB(*N*), therefore this machine is named as BB(*N*) candidate, and its productivity sets a lower bound for ∑(*N*). For small values of N it is possible to decide the entire search space and thus determine the exact value of ∑.

The majority of the research concerning the BB problem deals with the original variant, 5-tuple TMs. In 1962 Lin and Rado demonstrated that ∑(1) = 1 and that ∑(2) = 4 (Rado, 1962). Latter they also showed that ∑(3) = 6 (Lin and Rado, 1965). Ten years later Brady determined that ∑(4) = 13 (Brady, 1975; Brady, 1983; Brady, 1988). For N>4 the value of ∑ is unknown.

Due to the interest of the problem, several contests were organized aiming to determine the values of ∑ or to find new BB candidates. In 1983 Schult (Ludwig et al., 1983) discovered a 5-state TM, which produced 501 **1**'s in 134467 steps. A year later Dewdney presented the BB problem to a larger audience by describing it in his column *Computer Recreations* of *Scientific American* (Dewdney, 1985). This article caught the attention of G. Uhing, an amateur mathematician, that in December of the same year presented a new BB(5) candidate, showing that ∑(5) ≥ 1915. The machine in question performs 2133492 steps before halting.

This record was only beaten in 1990, by Marxen and Buntrock, that discovered a machine with a productivity of 4098 (Marxen and Buntrock, 1990). The machine in question performs 47176870 steps. Their work has some innovative features:

- **Early detection of non-halting machines** – In spite of the halting problem being non-computable, it is possible to determine, in a large percentage of cases, if a particular machine stops. Marxen and Buntrock employ several techniques that

allow the early detection of non-halting TMs. By using these techniques they were able to decide more than 99% of the 5 state TM space.

- **Equivalence classes** – There are several machines that are equivalent[1]. The identification of sets of equivalent machines enables a vast reduction of the search space.

- **Faster Simulation** – The state-of-the-art TM simulator created by Marxen and Buntrock enables an unprecedented speed in the simulation of TMs. This is mainly achieved by the use of "macro machines" that operate on blocks of symbols, performing in a single step a task that would require several steps to a TM. Additionally the employed tape representation also allows significant speed improvements (Marxen and Buntrock, 1990).

Marxen also applied this set of techniques to the 5-tuple BB(6) problem, consecutively improving the lower bound for $\sum(6)$. In table 1 we make a synthesis of the evolution of the values and lower bounds of $\sum$.

The research related to the BB problem is not limited to the search for new candidates. M. Green proposed a methodology for finding non trivial lower bounds for $\sum$ (Green, 1964). Machlin and Stout (1990) presented a characterization of the behavior of TMs that do not halt when started on a blank tape, identifying several behavioral classes. This type of study enables the development of efficient simulators by allowing the early identification of non-halting machines.

[1] Although 2.56×10^{10} 4-state TMs exist, this number can be reduced to 603712 through the use of equivalence classes.

Table 1. Values and lower bounds for ∑(N), 5-tuple. The second column indicates the number of steps performed by the BB machine, or best candidate, before stopping.

N	Σ(N)	Steps	Authors
1	1	1	Lin and Rado (Rado, 1962)
2	4	6	Lin and Rado (Rado, 1962)
3	6	21	Lin and Rado (Lin and Rado, 1965)
4	13	107	Brady (Brady, 1975)
5	≥ 501	134467	U. Schult (Ludwig et al., 1983)
5	≥ 1915	2133492	G. Uhing, 1984 (Dewdney, 1985)
5	≥ 4098	47176870	Marxen and Buntrock (Marxen and Buntrock, 1990)
6	≥ 136612	13122572797	Marxen and Buntrock (Marxen, 2002)
6	≥ 95524079	86903333816909510	Marxen, 1997 (Marxen, 2002)
6	$\geq 6.427499\times10^{462}$	6.196913×10^{925}	Marxen, 2000 (Marxen, 2002)
6	$\geq 1.29149\times10^{865}$	3.00233×10^{1730}	Marxen, 2001 (Marxen, 2002)

The number of studies focusing on the 4-tuple variant is significantly lower. This can be explained by the variant being more recent, and also by the higher popularity of 5-tuple TMs.

In table 2 we present the known values and lower bounds of ∑ for the 4-tuple variant, at the time our research work started. For $N < 4$ the values of ∑ can be easily found through an extensive search of the space. Due to the low complexity of these machines, the halting problem poses no serious difficulties. For higher values of N this is no longer true, and, as such, only lower bounds are known. We were also unable to determine the authors of the current BB(4) and BB(5) candidates.

Table 2. Values and lower bounds for ∑(N), 4-tuple. The second column indicates the number of steps performed by the BB machine, or best candidate, before stopping.

N	Σ(N)	Steps	Authors
1	1	1	Trivial
2	2	3	Trivial
3	3	7	Trivial
4	≥ 5	16	Unknown
5	≥ 11	52	Unknown
6	≥ 21	125	Cris Nielsen (Bringsjord, 1996)
7	≥ 37	253	Lally, Reineke and Weader (Lally et al., 1997)
8	≥ 86	1511	Norman, Chick and Marcella (Bringsjord, 1996)

The most productive 6 state TM known was found by Cris Nielsen and writes 21 **1**'s in 125 steps (Bringsjord, 1996; Barwise and Etchemendy, 2000). In 1997 Lally, Reineke and Weader set a new lower bound for ∑(7) by discovering a machine with a productivity of 37 (Lally et al., 1997). Their approach involves the creation of an abstract TM representation. They start by performing an analysis of known TMs of high productivity, which are transformed in block diagrams, allowing the discovery of common features. Next, the space of diagram blocks possessing these features is extensively searched. The simulation of the diagram blocks is faster than the simulation of the correspondent TMs, and the search space significantly smaller (Lally et al., 1997). The final stage consists in transforming the diagram blocks of higher productivity in TMs. For some cases it is impossible to construct an equivalent TM with the desired number of states, which is one of the drawbacks of the approach. Additionally, the transformation of the diagram blocks to TMs is performed by hand.

3 AN EVOLUTIONARY APPROACH TO THE BUSY BEAVER PROBLEM

In this section we present our initial approach to the BB problem. We start by making a brief overview of research that is related with the evolution of finite state machines.

Evolutionary Programming (EP) is an EC approach proposed by Fogel, Owens and Walsh in the 60's (Fogel et al., 1966). The goal of this approach was to evolve algorithms that were able to solve sequence prediction problems. Possible solutions were represented as finite state machines (FSM). A FSM is a very simple model of computation that consists of a set of states, a start state, an input alphabet and a transition function that maps input symbols and current states to a next state. A FSM has many similarities with a TM.

EP uses a collection of mutation operators that manipulate specific components of a FSM to generate descendants. They have been successfully applied to a large number of situations, ranging from the original sequence prediction problems to combinatorial optimization problems or evolution of strategies for games. Consult (Fogel, 1995) for a detailed description of application areas.

Mitchell, Crutchfield, and Hraber aimed to evolve Cellular Automata (CA) to perform computations (Mitchell et al., 1994). The main goal of this work was to understand the emergent behavior of CA and to analyze if an EC algorithm could be used as a method for engineering CA, so that they could perform general computations.

A CA can be defined as a two-dimensional organization of simple finite state machines whose next state depends on their own state and the ones of their closest neighbors. Conway's game of life is probably the most well-known instance of a CA

(Berlekamp et al., 1982). In general, the machines can be arranged in meshes of higher or lower dimensions, have larger neighborhoods, or be arbitrarily complex processors.

The lattice starts out with an initial configuration of cell states and this arrangement changes in discrete time steps, in which all cells are updated simultaneously according to the CA rules. A CA computation is composed of a sequence of steps that keep modifying the configuration. The program emerges from the CA rule being obeyed by each cell. The behavior of a CA is often illustrated by space-time diagrams (a plot of lattice configurations over a range of time steps). In their research, Mitchell et al. used a genetic algorithm to evolve CAs in which the actions of the cells are not random-looking, but are coordinated with one another so as to make possible the emergence of sophisticated parallel computations.

3.1 Related Research

To our knowledge there was just one previous attempt to apply EC techniques to the BB problem. In 1993, Jones and Rawlins (Jones and Rawlins, 1993) applied a straightforward EC algorithm to search for good candidates for several instances of the 5-tuple BB. The main goal of this study was to perform a comparison between hill-climbing and genetic algorithms (GAs) and also to analyze the reverse hill-climbing technique. This analysis tool helps to determine the probability that hill-climbing will attain a given point in the fitness landscape. The BB problem was used just as a test-bed to examine the performance of the algorithms. The goal was not to apply EC algorithms to seek for new lower bounds.

In the experiments described in the above mentioned work it is possible to verify that even though both approaches were able to find the optima solutions for the 5-tuple BB(N), $N \leq 4$, hill-climbing was less demanding in terms of computational power. In

table 3 we present a summary of the results. As it can be confirmed, for N=4, the EC algorithm has to evaluate, on average, 186 million individuals to find the optimum (which corresponds to 0.72% of the search space), whilst hill-climbing only evaluates 42 million individuals (0.16% of the search space).

These results allow different interpretations: on one hand, the optimum values were found, which shows the feasibility of applying EC techniques to this problem. On the other hand, the number of evaluations required to find the optimum is relatively high. Taking into account that the size of the search space increases exponentially with the number of states, and that machines with higher states typically present more complex behaviors, thus requiring more time to simulate, it is not viable to apply a similar approach to instances with a higher number of states.

Table 3. Average number of evaluations necessary to find the optimum (from (Jones, 1995)).

N	GA	Hill-Climbing
1	8	6
2	8117	542
3	123404	10606
4	186666666	42372351

3.2 Initial Approach

In spite of our goals being different from the ones of Jones and Rawlins, we decided to start our research by applying a standard genetic algorithm to the 5-tuple version of the BB problem. This would allow us to make a direct comparison of the results.

3.2.1 Representation

The search space of a BB(N) instance is composed of all N-state TMs (plus an anonymous halting state). We need therefore to find an efficient representation for such TMs. As stated before, a TM can be defined by a sextuple (Q,Π,Γ,δ,s,f). Without loss of

generality, we can consider Q={1,2,…,N,N+1}, set 1 as the initial state and N+1 as the final one. Since Π={**1**} and Γ={**B**, **1**}, for a given BB instance we only need to represent the transition function, since this is the only component that differs from TM to TM.

The transition function δ can be represented as a table. For N state TMs with a binary alphabet, this table has 4×N cells: for each state we need to specify the new state when the TM reads a blank, the associated action, the new state when it reads a **1** and the corresponding action.

For a 5-tuple TM there are 4 possible action pairs: {write **1** and move left, write **1** and move right, write blank and move left and write blank and move right}. Figure 1 shows a 5-tuple TM and its transition table.

δ	By blank		By one	
Q	New State	Action	New State	Action
1	2	1,L	2	1,R
2	1	1,R	3	B,R
3	*f*	1,R	4	1,R
4	4	1,L	1	B,L

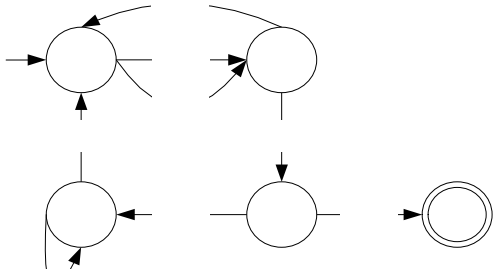

Figure 1. A 5-tuple TM and its transition table. The blank symbol is represented by a **0**. This machine is the 4-state Busy Beaver (Brady, 1975).

In our approach, the chromosome is composed of a sequence of 4×N genes that encodes a transition table. Figure 2 presents a structured representation of such a chromosome.

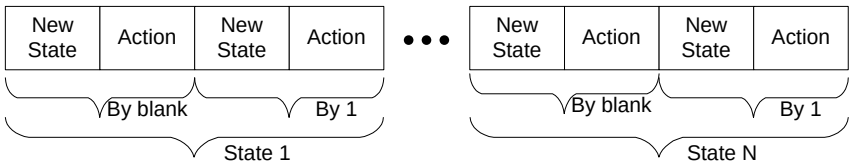

Figure 2. Structure of a chromosome.

During the optimization, we used two-point crossover and a standard mutation operator. When applied to a given gene, mutation changed its value to a new one from its domain.

3.2.2 Simulation and Fitness Assignment

In order to evaluate an individual we need to obtain the TM encoded in the genotype and then simulate it. In these experiments we performed a direct decoding of the information from the chromosome.

Since some of the machines never stop, one must establish a maximum number of steps, *MaxT*, and consider that machines that don't halt after *MaxT* steps will never halt. Additionally, if during simulation the number of ones present on the tape is larger than *MaxOnes* the simulation is also stopped[2]. Since *MaxT* >> *MaxOnes*, significant speed improvements can be achieved.

After running the TM we can determine the fitness of the corresponding individual. In their research, Jones and Rawlins considered that the fitness of an individual should be equal to its productivity (Jones and Rawlins, 1993). This approach implies that a significant number of individuals have the same fitness, which is undesirable. In an attempt to avoid this problem we decided to take other factors into consideration.

[2] When the value of $\sum(N)$ is know we make *MaxOnes* equal to $\sum(N)$ and *MaxT* equal to the number of steps made by the BB machine. In the other cases these values are established empirically. The nature of the BB problem ensures that it is impossible to establish an upper bound for $\sum(N)$ (Rado, 1962).

The underlying assumption is that TMs that exhibit a complex behavior should be valued, since good BB candidates will surely have a complex behavior. This way when assigning fitness to an individual, in addition to the productivity of the TM we also take into consideration the number of steps it made before reaching the halting state. We assume that TMs that stop after performing a large number of steps tend to have a more complex behavior than others that make just a few steps. To distinguish between TMs that do not stop we consider the number of steps left on the tape when the simulation is stopped. Following these ideas the fitness of individual *i*, *f(i)*, is given by:

$$f(i) = \begin{cases} \frac{\sigma(i)}{MaxOnes} \times \alpha + \frac{\omega(i)}{MaxT} \times \beta, \text{ if } i \text{ halts} \\ \frac{\sigma(i)}{MaxOnes} \times \gamma, \text{ if } i \text{ does not halt} \end{cases} \quad \textbf{(2)}$$

where $\sigma(i)$ represents the number of ones present on the tape when the machine stops and $\omega(i)$ the number of steps performed.

3.2.3 Experimental Results

We tested our approach in the 4 state instance of the 5-tuple BB problem. The experimental settings were the following: number of evaluations=25000000; population size= 200; two-point crossover with rate=0.7; single point mutation with rate=0.05; roulette wheel selection; elitist strategy; *MaxT*=150; *MaxOnes*=13; α=1, β= 0.3, γ=0.5.

The experiment was repeated thirty times with the same initial conditions and different random seeds. All initial populations were randomly generated. Values for different parameters were set heuristically. Nevertheless, and even though we did not perform an extensive parametric study, we conducted some additional tests and verified that, within a moderate range, there was not an important difference in the outcomes.

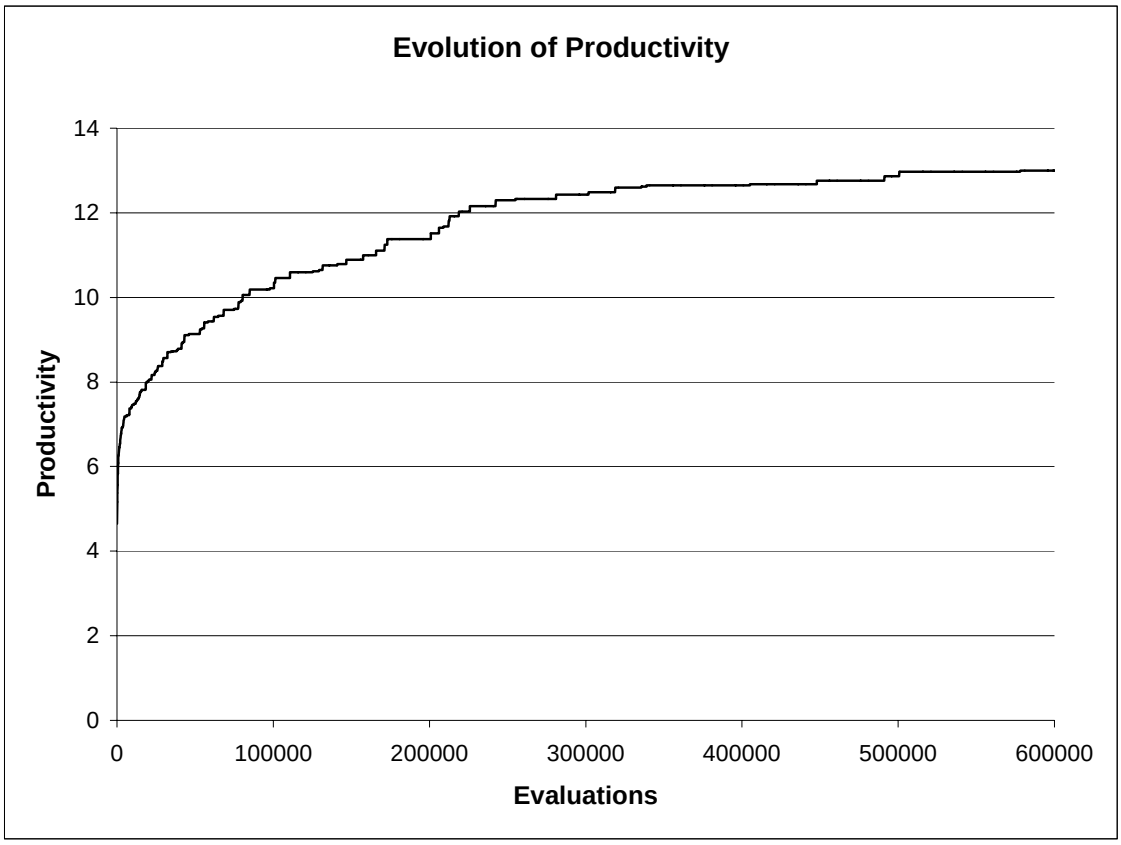

Figure 3. Chart showing the evolution of the productivity of the best individual

In figure 3 we present the evolution of the productivity of the best individual. The presented result is the average of the 30 runs. The BB(4) machine was found in all runs. On average it took 179806.7 evaluations to find it (standard deviation was 137082.7). In the best run, the EC algorithm required just 18574 evaluations to reach the optimum, while in the worst it needed 578266 evaluations. These results are clearly superior to the ones achieved by Jones and Rawlins (Jones and Rawlins, 1993) (average of 186666666 evaluations to find the BB machine). The explanation for this difference lies, mainly, on the fitness function. Additionally, the goal of Jones and Rawlins was to compare two approaches, as such we believe that the optimization of the EC parameters was not considered important.

3.3 Four-Tuple Variant

Even tough we achieved promising results with BB(4) we faced a difficult problem. To apply our approach to BB(5) we had to set MaxT to 47176870, since the current candidate performs this number of steps to reach a productivity of 4098. It is obvious that this would involve a large computational effort.

Moreover, our goal was to establish some lower bounds. Marxen and Buntrock (Marxen and Buntrock, 1990) performed an exhaustive search of the BB(5) solution space, leaving less than 1% of it undecided (they weren't able to determine if these machines eventually halted). Therefore, the chances of finding a new candidate for this instance are relatively slim. Moreover, if this machine exists, it will certainly perform much more than 47 million steps. In their research, Marxen and Buntrock resort to a sophisticated and complex TM simulator that significantly improves simulation speed. Without this sort of tool it is unfeasible to attack any 5-tuple instance greater than BB(4).

Taking these factors into consideration we abandoned the 5-tuple variant and focused our attention on the 4-tuple variant of the problem. When we moved to this alternative we had to modify several features of our algorithm related to the codification of the information in a chromosome and to the fitness assignment. In what concerns the representation, the structure of the chromosome (the number of genes) remains unchanged. Genes related to actions still can take four different values, although they are different from the 5-tuple approach: {write **1**, write **B**, move left, move right}.

Since the 4-tuple variant has different rules for defining the productivity, we had to modify the fitness assignment, even tough we maintained the same basic principle. When assigning fitness to an individual we consider the following factors in decreasing order of importance:

- *h(i)* is equal to 1 if the machine halts before reaching the limit number of steps and 0 otherwise;
- *v(i)* is equal to 1 if the machine follows the 4-tuple variant rules defined in (Boolos and Jeffrey, 1989), 0 otherwise;
- $\theta(i)$ represents the number of transitions used (each TM has 2N transitions, however some of them may never be used);
- $\sigma(i)$ represents the number of ones present on the tape when the machine stops
- $\omega(i)$ represents the number of steps performed.

Given these factors, the equation used to assign fitness is the following:

$$f(i) = h(i) \times \left[\left(1 + v(i) \times \alpha\right) \times \sigma(i) + \left(1 + v(i)\right) \times \theta(i) \times \beta + \left(1 + v(i) \times \gamma\right) \times \omega(i) \right] \quad \textbf{(3)}$$

TMs that do not stop have fitness 0. Constants α, β, γ determine the relative weight of each factor. Values for these parameters are empirically determined.

3.3.1 Experimental results

After modifying our algorithm, we conducted several preliminary tests with the 7 state instance of the problem.

The experimental settings were the following: number of evaluations=25000000; population size= {200, 1000}; two-point crossover with rate=0.7; single point mutation with rate= {0.01, 0.05, 0.1}; roulette-wheel selection; elitist and non-elitist strategy; *MaxT*=50000; *MaxOnes*=500; α=4, β= 2, γ=1.

All experiments were repeated 10 times with the same initial conditions and different random seeds. Initial populations were randomly generated.

In table 4 we present, for all different experiments, the productivity of the best individual of the final generation. Results are averages of the 10 runs.

Table 4. Productivity of the best individual of the final population. Results are averages of 10 runs.

	Strategy	**Elitist**		**Non Elitist**	
	Pop. Size	200	1000	200	1000
Mutation	1%	14	12	13	12
	5%	18	17	17	14
	10%	20	20	18	19

The small number of runs prevents us to determine if there are statistical significant differences between the results achieved by different settings. Anyway, results suggest that a fairly high mutation rate helps to discover good BB candidates. Adopting an elitist strategy also seems to improve the search performance. The EC algorithm was not sensitive to the variation in the population size. A detailed analysis on the results achieved on these experiments can be consulted in (Pereira et al., 1999a).

The most important goal of this experiment was to determine whether it was possible to discover new lower bounds for some instances of the BB using our approach. Although preliminary, results were rather clear. Even though this could be considered a rudimentary approach we were able to discover new candidates for several instances. In the experiments we described we found a 7 state machine with productivity of 102 (Pereira et al. 1999a). If we compare this value with the previous best candidate (which has a productivity of 37 (Lally et al., 1997)), it is evident that our evolutionary method can bring significant improvements.

We also performed some additional tests with other instances. With BB(6), we found on a regular basis TMs with a productivity of 21, solutions which are equivalent to the best known candidate discovered by Cris Nielsen (Bringsjord, 1996). As for BB(8), we found a TM that writes 384 **1**'s in 43368 steps. The previous best known candidate has a productivity of 86 (Bringsjord, 1996). These results show that EC algorithms can be considered efficient methods to search for good candidates for the BB problem.

4 THE INFLUENCE OF REPRESENTATION

An analysis of the attained experimental results indicated that there was room for improvement. We verified that the best results were found infrequently and that, in most of the runs, the algorithm got trapped in local optima. In this section we focus our attention in representational issues. We present and analyze three different ways to represent TMs. Our goal is to determine which representation improves the competence of the EC algorithm when searching for good solutions to the BB problem.

4.1 Representation

It is clear that there are several TMs that, even though they are different, exhibit the same behavior. As an example, consider the following transition: $\delta(1, \mathbf{B}) = (1, R)$. If, without loss of generality, we assume that 1 is the start state, all TMs with the above mentioned transition will loop forever and the other transitions are useless. This way, in what concerns to the BB, all these TMs can be considered equivalent. Assuming that we can construct sets of equivalent TMs, we just need to run one of the machines of a given set to determine the behavior of all the other elements. If we could develop a

representation where all the TMs belonging to a set are encoded in the same way (i.e., they have the same genotype), we would drastically reduce the size of the search space.

The most important of these equivalent classes is known as the Tree Normal Form (TNF) (Marxen and Buntrock, 1990). Using a TNF representation ensures that machines differing only in the naming of the states or in transitions that never are used, are represented in the same way.

The crucial problem with this representation is that it is not possible to directly translate a TM into its TNF. To perform this conversion (or to recognize that it is already in its TNF) we have to run the TM. For example, before the simulation it is impossible to know if all its transitions will be used (otherwise it would be possible to solve the halting problem). Therefore, a simple rule to convert a TM to its TNF is to simulate it and number the states in the order that they are visited. Unused transitions are deleted. Since it is impossible to determine if a given TM is in its TNF without simulating its behavior, it is not viable to directly adopt this form of representation.

In spite of these difficulties, Marxen and Buntrock – that, just like we mentioned before, try to perform an exhaustive exploration of the space when searching for BB – rely on a TNF representation to accelerate the search process. When simulating a TM, they verify if the machine is in its TNF. If, during the simulation they confirm that the machine is not in its TNF they abort the process and immediately proceed to the next candidate (for example, if a TM jumps from state 1 to state 3 without visiting state 2, then it is not in its TNF and simulation can be stopped). This kind of approach is reliable because the search space is fully examined and sooner or later the TNF TM with equivalent behavior will be simulated.

Even though we cannot directly use TNF as a way of representing individuals, we decided to analyze a possible alternative: to interpret the information contained in the

genotype in such a way that the resulting TM is in its TNF. The idea is to change how the genotype-phenotype mapping is performed. With this new approach, the structure of chromosome remains unchanged, whilst the interpretation of the encoded information is modified. During simulation, the chromosome is interpreted in a way that ensures that the resulting TM is in its TNF. This is achieved by performing two straightforward modifications in the TM simulator:

- Consider that states *1* to *m* were already visited and that transition *t* will be used for the first time:
    - If *t* leads to an unvisited state *e* and $e > m+1$ then change transition *t* so that it leads to state $m+1$;
    - If *t* is the last undefined transition from states *1* to *m*, then change transition *t* so that it leads to state $m+1$;

```
Mark all transitions as undefined
Mark all states as not-visited
visited_states ← 0
defined_transitions ← 0
CState ← 1
Steps ← 1
While (CState ≠ N+1) and (Steps < MaxT) Do
  Read the symbol S on the tape
  If the transition δ(CState, S)→(New_state, Action) is undefined Then
     Mark it as defined
     defined_transitions ← defined_transitions + 1
     If CState is not-visited Then
        Mark it as visited
        visited_states ← visited_states + 1
     If New_state > visited_states + 1 Then
        New_state ← visited_states + 1
     If defined_transitions = visited_states × 2 Then
        New_state ← visited_states + 1
     Update δ(CState, S)→(New_state, Action)
  End If
  Execute Action
  Steps ← Steps + 1
  CState ← New_state
End While
```

Figure 4. TNF interpretation algorithm.

In figure 4 we present the TM simulation algorithm that ensures that the genotype is interpreted as if it was in TNF. A careful analysis shows that TMs only reach the final state after visiting all other states.

It is now possible to conduct a set of tests with different representational approaches, so that we can compare their efficiency. We consider three different options for the representation and interpretation of TMs:

- Standard: Direct decoding of the genotype into a TM.
- TNF: Interpretation of the genotype as if it was in TNF. This is achieved using the algorithm from figure 4.
- IC: Just like in the previous option, the TNF simulator is used. After the simulation, the resulting TM is directly coded in the genotype of the individual.

4.2 Experimental Results

We used BB(6) as the test-bed in a comprehensive set of experiments where we tried to determine which representation allows us to find good solutions in a consistent way. When the tests were performed the best candidate for this instance had a productivity equal to 21 reached in 125 steps.

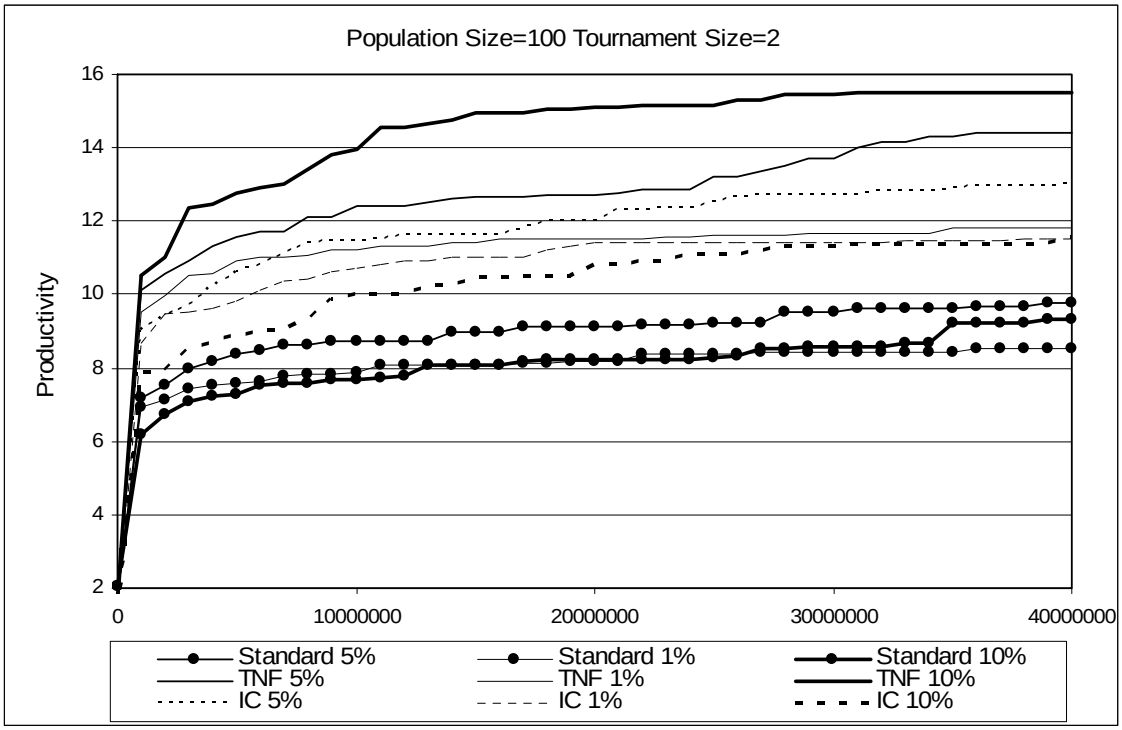

Figure 5. Evolution of the productivity of the best individual in experiments with population size=100 and tournament size=2. Results are averages of 30 runs.

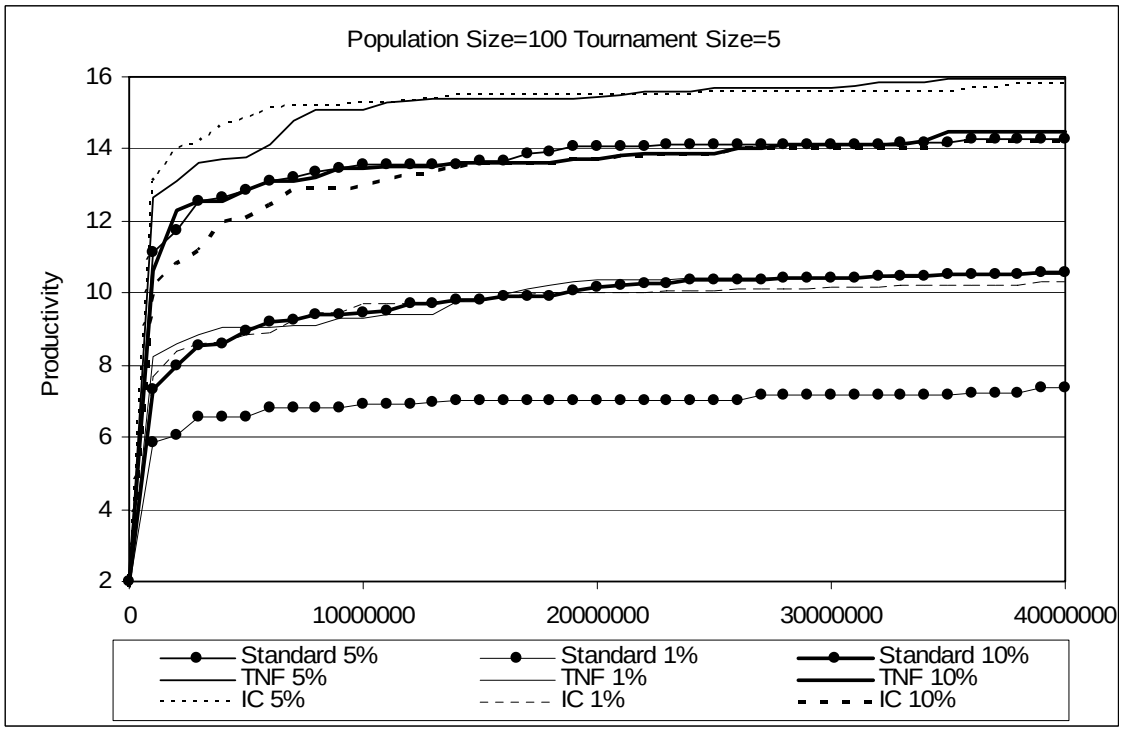

Figure 6. Evolution of the productivity of the best individual in experiments with population size=100 and tournament size=5. Results are averages of 30 runs.

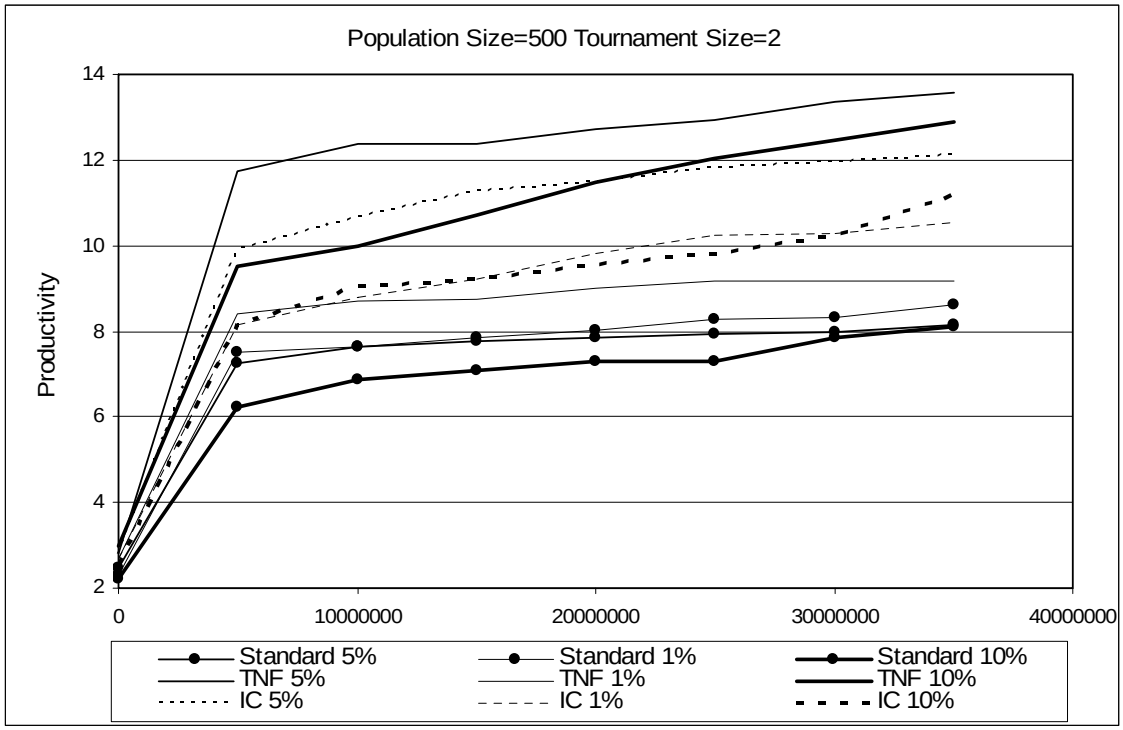

Figure 7. Evolution of the productivity of the best individual in experiments with population size=500 and tourney=2. Results are averages of 30 runs.

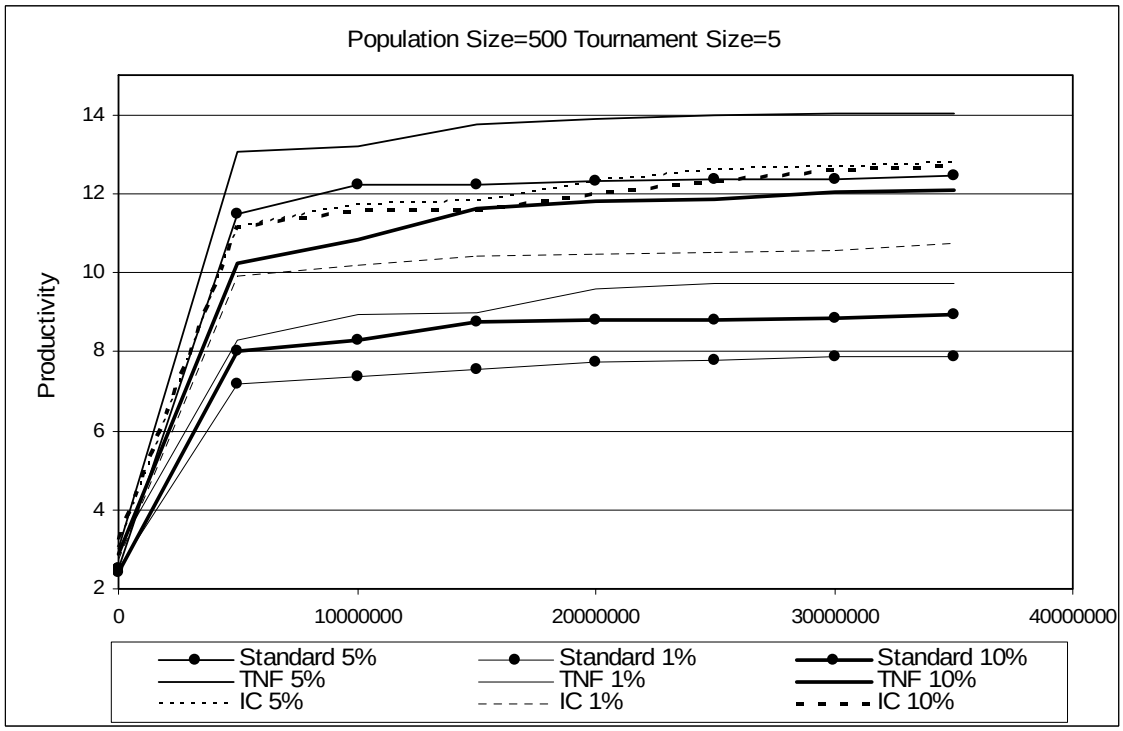

Figure 8. Evolution of the productivity of the best individual in experiments with population size=500 and tourney=5. Results are averages of 30 runs.

The experimental settings were the following: number of evaluations=40000000; population size= {100, 500}; two-point crossover with rate=0.7; single point mutation with rate= {0.01, 0.05, 0.1}; Tournament selection with tourney size = {2, 5}; elitist strategy; Interpretation={Standard, TNF, IC}; MaxT=250; MaxOnes=100; α=3, β= 2, γ=1. All experiments were repeated 30 times with the same initial conditions and different random seeds. Initial populations were randomly generated. The charts on figures 5 to 8 illustrate the evolution of the productivity of the best individuals. Results are averages of the 30 runs.

A brief perusal of the graphs shows that TNF and IC outperform Standard interpretation. It is also visible that a fairly high mutation rate is desirable. Results achieved when using a 1% mutation rate are clearly worse than when using 5% or 10% mutation rates. The difference between 5% and 10% mutation rates is less significant, though 5% gives better results. Tendentiously, small populations (100 individuals) perform better.

Table 5. Productivity of the best individual of the final population. Results are averages of 30 runs.

		Standard				**TNF**				**IC**			
	Pop. Size	100		500		100		500		100		500	
	T. Size	2	5	2	5	2	5	2	5	2	5	2	5
Mutation	**1%**	8.5	7.4	8.6	7.9	11.8	10.5	9.2	9.7	11.5	10.3	10.7	10.7
	5%	9.8	14.3	8.2	12.6	14.4	15.9	14.0	14.1	13.0	15.8	12.3	13.0
	10%	9.3	10.6	8.1	9.0	15.5	14.5	13.0	12.2	11.6	14.2	11.2	12.7
	Totals	9.2	10.8	8.3	9.8	13.9	13.6	12.1	12.0	12.0	13.4	11.4	12.1
		9.98		9.07		13.77		12.03		12.73		11.77	
		9.53				12.90				12.25			
		11.56											

In table 5 we present the average number of ones written by the best individual of the final population, for all possible configurations. Results are averages of 30 runs.

The average productivity of the 360 runs performed with the Standard interpretation is 9.53. Experiments performed with TNF and IC interpretation achieved, respectively, average productivities of 12.9 and 12.25. Differences between the productivities of experiments that adopted a TNF interpretation of the chromosome (TNF and IC) and the experiment that relied on standard representation are statistically significant (significance level: 0.05).

In table 6 we present, for all configurations, the difference between the productivity of the best individual in experiments with a Standard interpretation and experiments with TNF or IC. Bold entries represent statistical significant differences (significance level: 0.05). The results presented on this table show that significant differences exist for most of the configurations. On the other hand, the difference between the results achieved by TNF and IC interpretation is not statistically significant.

Table 6. Differences in productivity between standard and TNF interpretations.

		TNF				**IC**			
	Pop. Size	100		500		100		500	
	T. Size	2	5	2	5	2	5	2	5
Mutation	**1%**	**3.3**	**3.1**	0.6	**1.8**	**3.0**	**2.9**	**2.1**	**2.8**
	5%	**4.6**	**1.6**	**5.8**	**1.5**	**3.2**	1.5	**4.1**	0.4
	10%	**6.2**	**3.9**	**4.9**	**3.2**	**2.3**	**3.6**	**3.1**	**3.7**
	Totals	**4.7**	**2.9**	**3.8**	**2.2**	**2.8**	**2.7**	**3.1**	**2.3**
		3.8		**4.0**		**2.8**		**2.7**	
		3.4				**2.7**			

Table 7. Number of runs in which a BB(6) candidate with productivity 21 was found. Blank cells indicate that none of the 30 runs found this TM.

		Standard				**TNF**				**IC**			
	Pop. Size	100		500		100		500		100		500	
	T. Size	2	5	2	5	2	5	2	5	2	5	2	5
Mutation	1%												
	5%					1	6	2			4	1	
	10%		1			4	3	1				2	
	Totals		1			5	**9**	3			4	3	
		1				***14***		3		4		3	
		1				***17***				**7**			

In table 7 we present the number of runs in which the BB(6) candidate with productivity 21 was found. Using a Standard representation, the EC algorithm found a machine with productivity 21 once in 360 runs. This result shows the difficulty of the problem and how the space can be hard to search. When using TNF, candidates with productivity of 21 were found in 17 runs and with IC in 7 runs. Bold entries highlight statistical significant differences (significance level: 0.05) between experiments with a Standard interpretation and experiments with TNF or IC. Results from table 7 confirm the superiority of TNF over the standard interpretation.

If we consider this criterion, it is also possible to determine statistical significant differences between experiments that use TNF and IC interpretations. Italic entries in table 7 highlight the existence of such a difference.

TNF interpretation is achieved through the modification of the standard simulation. Even though these modifications do not change the genotype space, they considerably reduce the number of distinct phenotypes. The most important phenotype reduction is due to the restriction that specifies that states belonging to the TM should be visited in order. Anyway, it is important to notice that this also reduces the number of possible solutions. When using a Standard representation there are several isomorphic solutions,

while in TNF they are reduced to just one. Therefore, the advantage of TNF interpretation cannot be explained just by this reduction on the phenotype space.

There is, however, another type of space reduction when TNF interpretation is used. A careful analysis of the algorithm presented in figure 4 reveals that TMs are only allowed to enter the final state after visiting all other states. Thus, in TNF, machines that halt, always visit all states. This situation might allow the development of solutions with complex behavior in the early stages of the evolution process. Since TMs that do not stop have fitness 0, descendants of the machines that visit all states before halting will dominate initial populations. One justification for the advantage of TNF is that this kind of machines can be considered as a good starting point to the evolution of high quality BB candidates.

On the contrary, with a Standard interpretation, a machine can halt after visiting a small number of states. Although these machines may have a simple behavior, they will still have a fitness score higher than most of the individuals from their generation and will tend to dominate the populations, hindering the formation of good candidates.

The chart in figure 9 shows the evolution of the average number of visited states for Standard, TNF and IC. A logarithmic scale is used to allow an easier visualization of the early stages of the simulation.

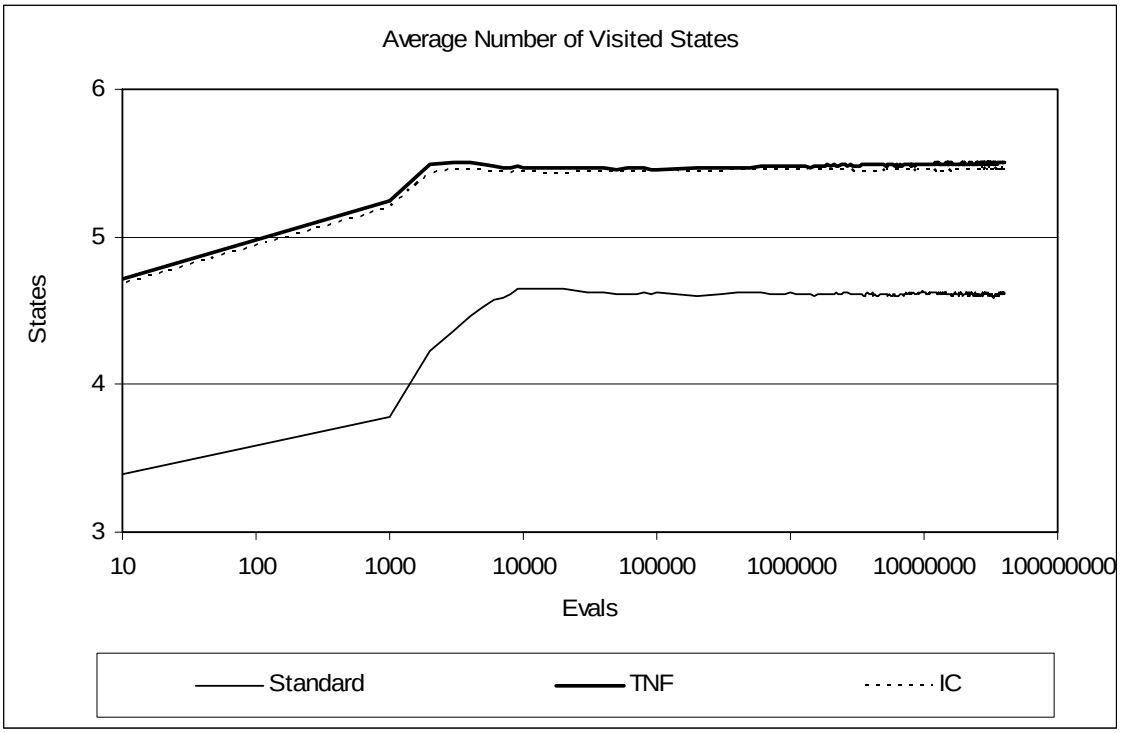

Figure 9. Evolution of the average number of visited states (logarithmic scale).

Another interesting characteristic of TNF interpretation is that it induces an ordering of the states. States that are directly connected have a higher probability of being close in the chromosome. Thus, there is a higher similarity between genotype neighborhood and phenotype neighborhood. It is reasonable to assume that this situation helps the evolutionary process, since there is a decrease in the average number of transitions broken by crossover.

There is an interesting effect related to the TNF interpretation. Consider that we have a chromosome whose gene value for *New State* of the transition of state 2 by blank is 3; admit also that, when this transition is first used, state 3 was not yet visited. Even if mutation changes this value to 5, there will be no changes on the phenotype, since the TNF interpretation will still consider it as a transition to state 3. This simple example

shows that TNF interpretation allows the accumulation of neutral mutations (i.e., changes in the genotype that do not lead to any modification in the phenotype).

The way in which these neutral mutations may influence the behavior of the evolutionary algorithm depends whether a TNF or an IC interpretation are used. With the first option, even though the neutral mutations are not expressed in the phenotype (and as a consequence do not affect the fitness of the individual), they remain in the genotype. Later on, due to other alterations in the chromosome they may become effective. As for IC, the transfer of the phenotype to the genotype eliminates all neutral mutations.

Results presented in this section show that TNF clearly outperforms a Standard representation, enabling the discovery of good candidates for the BB problem. In what concerns the advantages/disadvantages of coding back to the genotype the TM derived from the TNF interpretation, results presented are somewhat inconclusive. Even though experiments that relied on a TNF interpretation achieved results that are slightly better than those achieved by IC, differences are not too big. Statistical significant differences were only visible when we considered the number of runs in which the BB(6) candidate with productivity 21 was found.

5 GENETIC OPERATORS

The performance of an EC algorithm depends, to a large extent, on the adopted genetic operators. The relation of vicinity between points of the search space is established by these genetic operators, which, in conjunction with the fitness function, also defines the topology of the space.

Due to their importance a significant portion of the research in EC focuses on the study and development of operators. The choice, or development, of the operators is based on the characteristics of the problem. The type of structure being manipulated is, usually, one of the determinant factors.

Two-point crossover is one of the most common recombination operator for the manipulation of linear structures. The manipulation of structured representations, such as the ones employed in Genetic Programming (GP), is typically carried out through operators particularly suited to these structures. In GP, for instance, the individuals are represented by trees, as such the typical recombination operator (Koza, 1992) promotes the exchange of sub-trees between individuals. This type of approach is based on the notion that it is advantageous to manipulate the individuals in a way that is consistent with their structure, and that respects the underlying syntactic restrictions (Angeline, 1993).

As previously stated, our goal is the evolution of TMs. The natural representation of a TM is a graph. In the experiments described in the previous sections we resorted to two-point crossover. However, since the individuals are graphs[3], they should be manipulated as such, hence the development of a graph based crossover operator. In this section we present this operator, the results achieved through its use, and a comparison with the ones achieved by two-point crossover.

5.1 Graph Based Crossover

The basic idea of our operator is to promote the exchange of sub-graphs or, from a functional point of view, the exchange of sub-machines. The inspiration for this operator was the standard GP crossover operator that exchanges sub-trees.

[3] Internally the chromosome is represented as a vector. However, this is only an implementation issue and, therefore, not relevant from a conceptual viewpoint.

The primary goal is to improve the performance of our algorithm in the BB problem. Additionally, we aim at developing a generic operator that is suited for other problems in which the natural representation is a graph.

In order to explore the structure of the TMs, and since any sub-set of nodes and arcs is a sub-graph, we must impose constraints to the sub-graphs being exchanged. A TM, especially one with a complex behavior, is usually formed by several sub-machines. In the current context, we can define sub-machine as a set of functionally dependent nodes, and corresponding transitions, that perform a simple and well-defined task. Thus, the concept of sub-machine is similar to the concept of sub-routine. An analysis of high productivity TMs shows that their complex behavior is attained through the interaction of simple sub-machines.

Due to the nature of TMs, functional dependency is typically connected to the distance between nodes (considering as distance the minimum path length between nodes). Thus, the probability of two nodes being functionally dependent is higher if they are directly connected. Therefore, sub-machines tend to be composed of a set of neighbor nodes. Consequently, the sub-graphs should be composed of closely linked nodes, and by the transitions among them. From here on we name the transitions between nodes of a sub-graph as internal transitions.

Next we describe our graph based crossover operator, which, although independently developed, shares many characteristics with the one proposed by A. Teller and M. Veloso (1996).

5.1.1 Implementation

Considering two individuals, *A* and *B*, the application of the graph based crossover operator can be divided in two stages. In the first, we select the sub-graphs to be

exchanged. In the second, we perform the exchange of genetic material, generating two descendents.

The Selection of the sub-graphs is performed as follows:

- Randomly select crossover points, P_A and P_B, for each of the parents, and a crossover size, X. The value of X is randomly chosen from the interval (1, *MaxSize*), where *MaxSize* is a user specified constant.
- For each of the parents define the list of nodes, L_A and L_B, belonging to the corresponding sub-graphs:
    - The crossover point is the first node of the list. Then, through breadth first search, nodes of increasingly higher distance from the starting point are added to the end of the list. The order of points located at the same distance is randomly determined.
    - Next, the lists are truncated making their length equal to X.
    - When the lists have different lengths – which can happen when it is impossible to reach X-1 nodes from the crossover point – the larger is truncated making their lengths equal.
- The sub-graphs, S_A and S_B, will be composed of the nodes of L_A and L_B and by the internal transitions among these nodes.

The next step is the exchange of genetic material between the parents. Due to the existence of external transitions, i.e. transitions between nodes of the sub-graphs and nodes that do not belong to them, it is necessary to establish a correspondence between the nodes of S_A and S_B. The exchange of genetic material is performed in the following way:

- Based on the order of the elements in the lists L_A and L_B a correspondence between nodes is established.
- The internal transitions of S_A and S_B are mutually swapped. The external transitions remain unchanged. When the swapping operation leads to the existence of two transitions (an internal and an external one) from the same node and by the same tape symbol, the internal transition is deleted.

In Figure 10 we present an example of the crossover operation between individuals *A* and *B* at points P_A and P_B. The crossover size is three, L_A={1, 2, 4} L_B={2, 3, 6}, yielding the following correspondence table {1_A-2_B, 2_A-3_B, 4_A-6_B}.

When the number of internal transitions in S_A and S_B is different, there are transitions that won't be exchanged, since they don't have an equivalent. In the example shown in Figure 10, the transition 2×**0**→4×R of individual *A* is not replaced by 3×**0**→4×**1** since this transition is not internal.

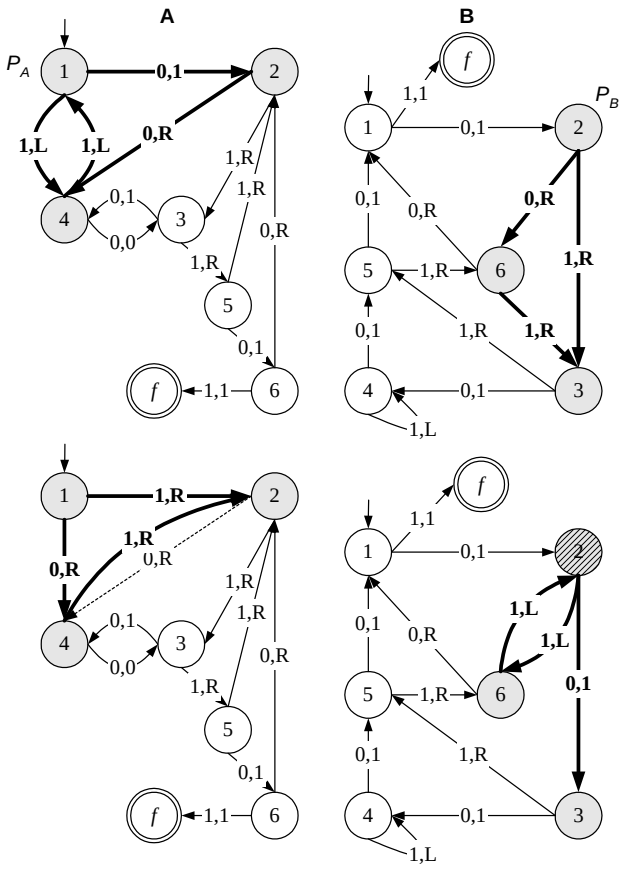

Figure 10. Example of Crossover. Nodes belonging to the sub-graphs are depicted in gray and transitions in bold. A dashed line represents internal transitions that weren't replaced.

5.2 Graph vs. Classical Two-Point Crossover

The experiments presented in this section concern the search for 4-tuple BB(6). Since the previously best known candidate wrote 21 **1**'s in 125 steps we set MaxT to 250.

The parameters of the EC Algorithm were the following: TM representation = {Standard, TNF, IC}; Number of Evaluations = {40 000 000}; Population Size= {100, 500}; Generation Gap=1; Crossover Operator = {Two-point, Graph}; Crossover rate = 0.7; Single point mutation; Mutation rate = {0.01, 0.05, 0.1}; Elitist strategy; Tournament selection; Tournament size = {2, 5}. Two-point crossover was restricted to gene boundaries. MaxSize = 3. A particular experimental configuration can be defined

as an instantiation of the following set {TM representation, Mutation Rate, Population Size, Tournament Size}. For each configuration we performed thirty runs with the same initial conditions and different random seeds.

Table 8 shows the average number of ones written by the best individual, of the final population, for all considered configurations. A brief perusal of the results indicates that graph crossover consistently outperforms two-point crossover, improving the results for all representations. When we use two-point crossover the average productivity is 11.56; with graph crossover it is 13.5. In addition to reaching higher global results, it also sets new best averages for specific configurations. The best configuration for two-point crossover {TNF, 5%, 100, 5} achieves a 15.9 average productivity. With graph crossover the best configuration, {IC, 5%, 500, 5}, achieves 18.6 average productivity.

Table 8. Results achieved in the 4-tuple BB(6). Productivity of the best individual of the final population. Each experiment was repeated 30 times. The results are the averages.

		Standard				**TNF**				**IC**			
	Pop. Size	100		500		100		500		100		500	
	T. Size	2	5	2	5	2	5	2	5	2	5	2	5
Mutation	1%	14.6	10.4	13.7	9.1	15.3	13.4	14.4	12.4	14.3	12.0	15.4	12.7
	5%	12.1	16.1	10.5	16.9	14.4	18.4	13.7	17.4	13.0	17.8	12.9	18.6
	10%	9.9	12.2	8.6	10.8	14.2	14.4	12.0	12.8	13.3	13.8	11.3	13.8
	Totals	12.2	12.9	10.9	12.3	14.7	15.4	13.4	14.2	13.5	14.5	13.2	14.9
		12.56		11.61		15.03		13.76		14.04		14.05	
		12.08				14.39				14.04			
		13.50											

It is also important to consider the evolution of productivity of the best individual over time. In Figure 11 we present the charts with the results of the experiments using TNF representation and a 5% mutation rate. We chose these settings because they are the ones where the best known contender is more frequently found. It is interesting to notice

that, for the majority of the experimental settings, the differences in productivity are substantial throughout the evolutionary process. The configuration {TNF, 5%, 100, 2} is one of the rare cases where this does not happen.

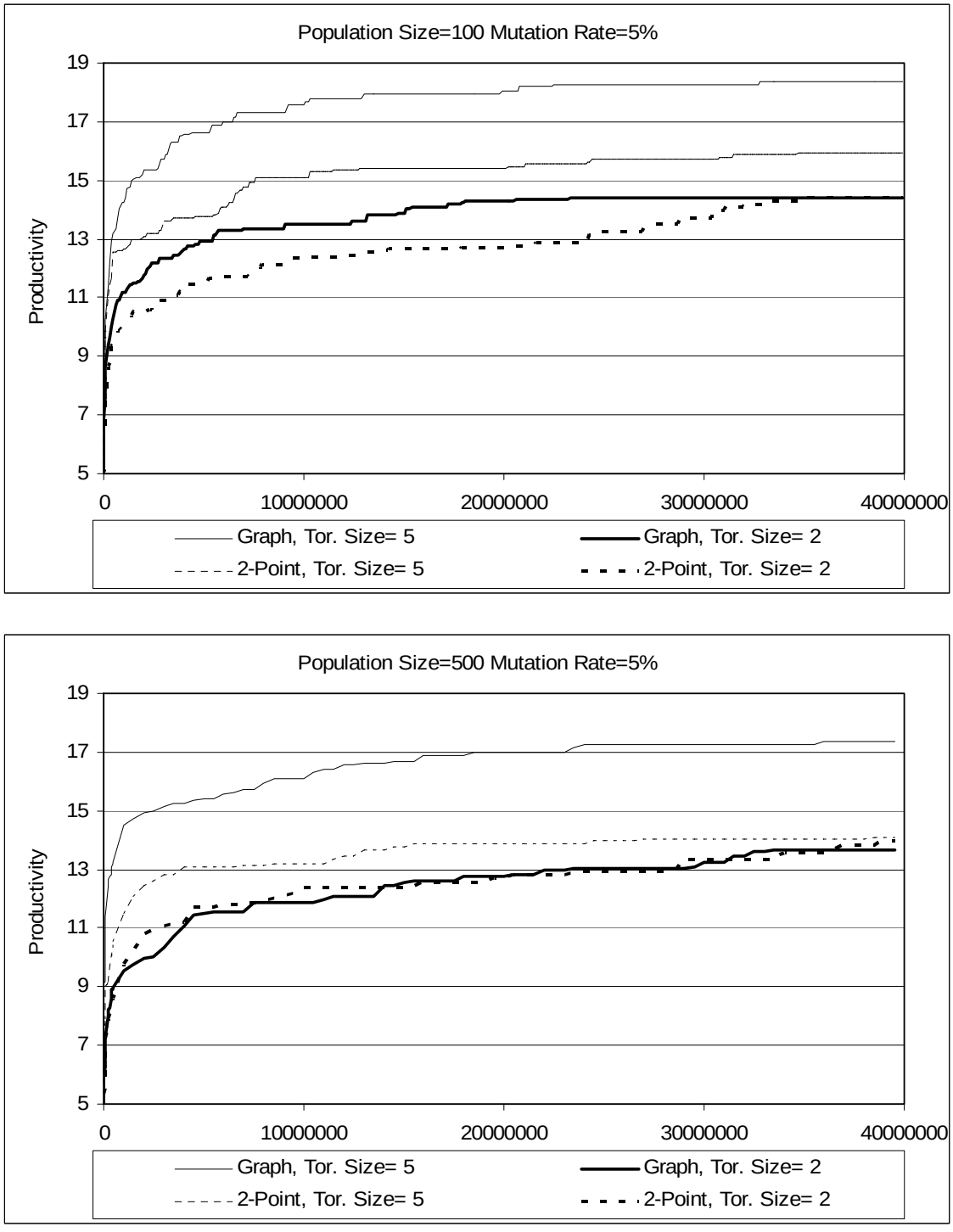

Figure 11. The charts show the number of ones written by the best individual using a *TNF* interpretation. The results are averages of series of 30 runs.

Table 9. Difference in productivity between Graph and Two-Point crossover for the best individual of the final population. The entries in bold represent statistically significant differences (significance level: 0.05).

		Standard				**Totals**	**TNF**				**Totals**	**IC**				**Totals**
	Pop. Size	100	100	500	500		100	100	500	500		100	100	500	500	
	T. Size	2	5	2	5		2	5	2	5		2	5	2	5	
Mutation	1%	**6.13**	**2.97**	**5.13**	1.20	**3.86**	**3.50**	**2.90**	**5.17**	**2.67**	**3.56**	**2.77**	1.73	**4.73**	1.57	**2.70**
	5%	**2.33**	**1.80**	**2.33**	**4.27**	**2.68**	0.03	**2.47**	-0.33	**3.27**	1.36	0.03	**2.00**	0.60	**5.60**	**2.06**
	10%	0.60	1.60	0.47	**1.83**	1.13	-1.27	-0.07	-1.00	0.57	-0.44	1.70	-0.40	0.10	1.10	0.63
	Totals	**3.02**	**2.12**	**2.64**	**2.43**		0.75	**1.77**	1.28	**2.17**		**1.50**	1.11	**1.81**	**2.76**	
		2.57		**2.54**			1.26		**1.73**			1.31		**2.28**		
		2.56					**1.49**					**1.79**				
		1.95														

The advantages of using graph crossover diminish as the mutation rate increases. This is shown clearly in Table 9, which presents the difference between the results achieved by two-point crossover and graph crossover, for the same configurations. For a mutation rate of 1%, the average increase in productivity is 3.37, for 5% mutation 2.03, and for 10%, the average gain is 0.44. These facts indicate that graph crossover overcomes the tendency to converge to sub-optimal solutions. Two-point crossover between individuals with resembling genotypes will result in an individual that is also similar (e.g. two-point crossover between the same individual doesn't produce any change). With graph crossover this is not necessarily true. In late generations, when a few individuals tend to dominate the population, graph crossover may promote diversity. A 10% mutation rate is too high for this type of crossover. This is especially visible in the results of *TNF* representation. An expected result since *TNF*, by itself, enables higher population diversity (Machado et al, 1999).

In Table 10 we indicate the number of times that the best 4-tuple BB(6) candidate (21 **1**'s in 125 steps) was found. These results confirm and emphasize the previous ones. The difference between the two crossover operators is impressive. Using a 1% mutation rate and two-point crossover we were unable to find the best contender, while with the

new crossover operator this candidate was found 9 times. Conversely, with 10% mutation it was found 11 times with two-point and 7 with graph crossover, which confirms our previous remarks. The table also stresses the increase in performance for the best configuration {TNF, 5%, 100, 5}, from 6 to 11. The number of runs performed does not render statistically significant differences for specific configurations. Nevertheless, the overall difference (25 vs. 47) is statistically significant (significance level: 0.05).

The idea behind the proposed crossover operator is to take advantage of the inherent relations in the sub-structures that compose a TM. These sub-structures (or sub-machines) can be considered as the building blocks of our problem. Two-point crossover is “blind” with regard to the structure of a TM, being, therefore, ineffective in the combination of such blocks. Graph crossover is aware of the structure of the individuals, enabling effective discovery and recombination of building blocks of increasing order. Our results prove that solutions obtained by successive recombination of building blocks enable the discovery of TMs with complex behavior and credible candidates for the BB problem.

In the previous section we showed that using a TNF interpretation of the TMs (with or without IC) significantly improves the results. This was partially explained by the reduction of the search space, and by the fact that TNF induces an ordering of the states. With TNF, states that are directly connected have a higher probability of being close in the chromosome, resulting in a higher similarity between genotype and phenotype neighborhood. Accordingly, the probability of functional dependent states being separated by two-point crossover is greatly reduced. The use of the proposed graph crossover operator implicitly redefines genotype proximity, making it a step closer to

phenotype proximity. Thus, this further reduces the probability of breaking sub-programs.

Table 10. Number of runs in which the maximum was reached. Blank Cells indicate that none of the 30 runs reached the maximum.

		Standard				TNF				IC			
	Pop. Size	100		500		100		500		100		500	
	T. Size	2	5	2	5	2	5	2	5	2	5	2	5
Mutation	**1%**		1	1		2		2	1		1	1	
	5%	1	1		2		11		5		3		8
	10%					3	1	1		1	1		
	Totals	1	2	1	2	5	12	3	6	1	5	1	8
		3		3		17		9		6		9	
		6				26				15			
		47											

6 SETTING NEW LOWER BOUNDS

In the previous sections we made an analysis of the influence of representation and genetic operators in the performance of the EC algorithm. In this section we describe a series of experiences that aimed to establish new lower bounds for $\Sigma(N)$.

A straightforward modification of the TM simulator enables us to attack, simultaneously, BB(N) and BB(M) for all $M<N$. During simulation, when a state is visited for the first time, it is considered as the final state and the productivity of the machine is calculated according to the 4-tuple rules. Then, this modification is discarded and simulation proceeds in the normal way. This is equivalent to raising the question:

"What would be the productivity of the machine if this was the final state?"

The productivity of the machine after visiting M states (with $M<N$) does not influence the fitness value. This optimization allows a more efficient use of computational

resources, enabling the simultaneous exploration of the problem's lower instances search spaces.

Using this version of the simulator we conducted several experiments attacking the 7-state instance of the problem. This allowed us to discover a new BB(7) candidate, showing that $\Sigma(7) \geq 164$. Surprisingly, we also discovered a new BB(6) candidate, presented in figure 12, which produces 25 **1**'s in 256 steps.

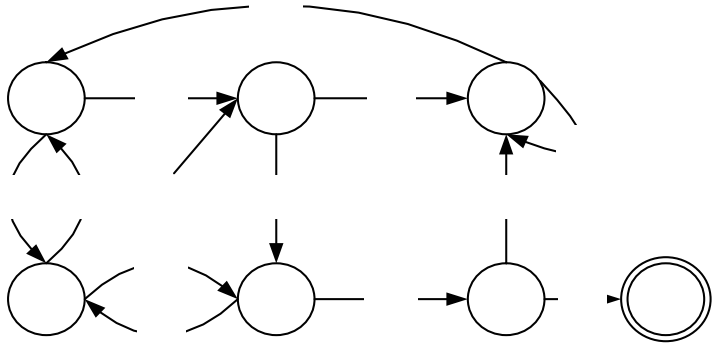

Figure 12. Current BB(6) candidate. This machine produces 25 **1**'s in 256 steps.

In the experiments described in the previous sections *MaxT* was set to 250, which partially explains why this candidate was not found. We conducted additional experiments for the BB(6) using a higher value of *MaxT* in order to allow the discovery of this machine. However, in these experiments the EC algorithm failed to discover this candidate. By itself this result is not anomalous, since EC approaches are mainly function satisfiers and not optimizers. On the other hand, this machine was repeatedly found in experiences concerning the BB(7) instance. Additionally, the current BB(7) and BB(8) candidates, presented in figure 13, were found when attacking BB(8) and BB(9), respectively. These facts suggest, at least in what concerns the discovery of new lower bounds, that it is advantageous to consider higher instances of the problem.

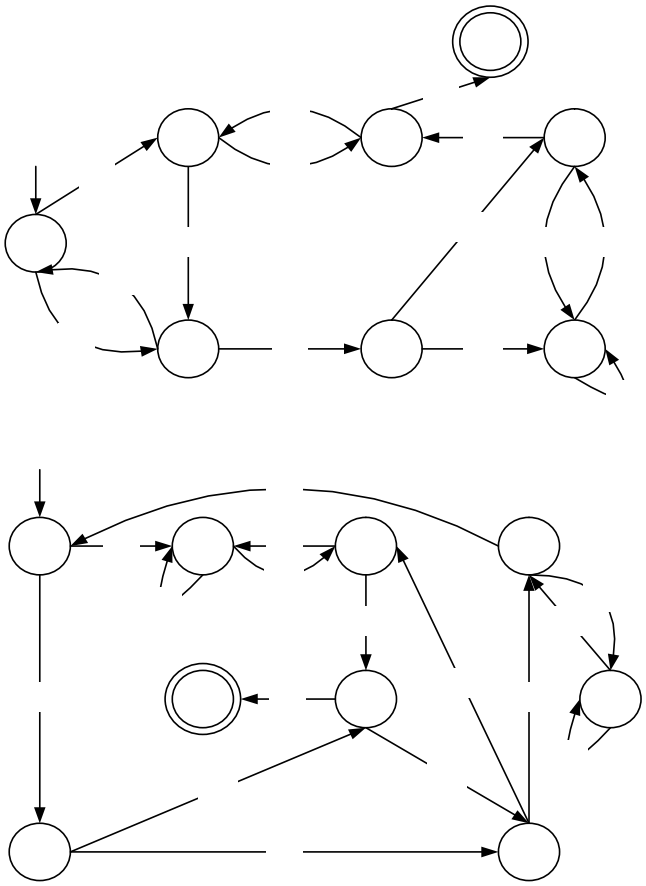

Figure 13. Current BB(7) and BB(8) candidates**.** These BB(7) candidate writes 196 **1**'s in 13683 steps, while the BB(8) candidate writes 673 **1**'s in 198340 steps.

1

We conclude this section by presenting a table showing the lower bounds for Σ(N) before and after the application of EC techniques to the BB problem. The improvements in productivity demonstrate that EC is a viable approach for the discovery of new candidates.

Table 11. Evolution the lower bounds of Σ(*N*).

	Before the Application of EC			After the Application of EC		
N	**Σ(N)**	**Steps**	**Authors**	**Σ(N)**	**Steps**	**Authors**
6	≥ 21	125	Cris Nielsen (Bringsjord, 1996)	≥ 25	256	Pereira and Machado (Pereira et al., 1999b)
7	≥ 37	253	Lally, Reineke and Weader (Lally et al., 1997)	≥ 196	13683	Pereira and Machado (Pereira, 2002)
8	≥ 86	1511	Norman, Chick and Marcella (Bringsjord, 1996)	≥ 672	198340	Pereira and Machado, 2002

7 CONCLUSIONS

The main goal of this research was to test the viability of using Turing Machines as a model for the evolution of computer programs. With this purpose in mind, we chose a problem that allowed us to test this idea, the Busy Beaver. This theoretical problem of established interest and difficulty was proposed by Tibor Rado in 1962.

Our first approach to the problem, although rudimentary, gave promising results, which can be compared favorably with previous EC approaches. This can be explained by the method used to assign fitness, which tries to estimate the complexity of the machines. An analysis of the attained experimental results allowed us to identify two key issues that influence the performance of the algorithm: representation and used genetic operators.

Following this line of reasoning, we studied different methods of interpreting the information encoded in the genotype. We tested three different genotype to phenotype mappings using the 4-tuple BB(6) instance of the problem. The experimental results show significant differences in performance, in terms of average productivity of the best individual, and of the frequency of discovery of the best known candidate.

The natural representation for a Turing Machine is a graph. As such, a graph based crossover operator specifically designed to manipulate Turing Machines was proposed. The motivation was twofold: the improvement of performance in the Busy Beaver problem; it was considered a fundamental step in the development of a consistent framework for the evolution of Turing Machines, which was the original goal of our research work.

The attained experimental results show that graph based crossover clearly outperforms standard two-point crossover, reinforcing the idea that it is advantageous to manipulate the individuals in a way that is consistent with their natural representation.

In addition to the contributions of potential relevance for Evolutionary Computation, such as the study of alternative representations and development of genetic operators, there are also significant contributions in the scope of the Busy Beaver problem. The discovery of new candidates for the 4-tuple BB(6), BB(7), and BB(8) instances of the problem is, by itself, important. Furthermore, we showed that Evolutionary Computation techniques are a viable and competitive approach to attack this problem.

An analysis of the outcome of the research in light of the initial goals reveals some shortcomings. In order to assess the feasibility of applying Turing Machines as a basis for the evolution of computer programs, it is necessary to apply the proposed model to a widespread range of problems. This is, undoubtedly, one of the obvious directions for future research.

8 BIBLIOGRAPHY

Angeline, P. J. (1993). *Evolutionary Algorithms and Emergent Intelligence*. PhD thesis, Ohio State University.

Barwise, J. and Etchemendy, J. (2000). Busy Beaver Problem. http://wwwcsli.stanford.edu/hp/Beaver.html.

Berlekamp, E., Conway, J. H., Guy, R. (1982). *Winning Ways for Your Mathematical Plays*, Volume 2, Academic Press.

Boolos, G. S. and Jeffrey, R. C. (1989). *Computability and Logic*. Cambridge University Press, Cambridge, third edition.

Brady, A. H. (1975). Solution of the non-computable "Busy Beaver" game for k=4. In *ACM Computer Science Conference*, pag. 27, Washington DC. ACM.

Brady, A. H. (1983). The Determination of the Value of Rado's Noncomputable Function Σ(*k*) for Four-State Turing Machines. *Mathematics of Computation*, 40(162):647–665.

Brady, A. H. (1988). The busy beaver game and the meaning of life. In Herken, R., editor, *The Universal Turing Machine: A Half-Century Survey*. Oxford University Press.

Bringsjord, S. (1996). Turing Machines (in "Turing's World"). http://www.rpi.edu/~brings/SL/tms.html.

Church, A. (1936). A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1:40–41 and 101–102.

Cutland, N. (1980). *Computability: an introduction to recursive function theory*. Cambridge University Press, Cambridge, UK.

Davis, M. D., Sigal R., and Weyuker, E. J. (1994). *Computability, Complexity, and Languages: fundamentals of theoretical computer science*. Academic Press, San Diego, CA, USA.

Dewdney, A. K. (1985). A Five-state Busy Beaver Turing Machine Contender. *Scientific American*, 252(4).

Fogel, D. B. (1995). *Evolutionary Computation: Toward a new Philosophy of Machine Intelligence* . IEEE Press.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York.

Green, M. W. (1964). A lower bound on Rado's sigma function for binary Turing machines. In *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pag. 91–94, Princeton, New Jersey. IEEE.

Hopcroft, J. E., and Ullman, J. D. (1979). *Introduction to automata theory, languages and computation*. Addison-Wesley Publishing Company, Reading, Massachusetts.

Jones, T. (1995). *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, NM.

Jones, T. and Rawlins, G. J. E. (1993). Reverse Hillclimbing, Genetic Algorithms and the Busy Beaver Problem. In Forrest, S., editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pag. 70–75, San Mateo, CA. Morgan Kaufman.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA.

Lally, A., Reineke, J., and Weader, J. (1997). An Abstract Representation of Busy Beaver Candidate Turing Machines. Technical report, Rensselaer Polytechnic Institute.

Lin, S. and Rado, T. (1965). Computer Studies of Turing Machine Problems. *Journal of the ACM*, 12(2):196–212.

Ludwig, J., Schult, U., and Wankmueller, F. (1983). Chasing the Busy-Beaver — Notes and Observations on a Competion to Find the 5-State Busy Beaver. Technical Report IB-B84700, University of Dortmund, Dortmund.

Machado, P., Pereira, F. B., Cardoso, A., and Costa, E. (1999). Busy Beaver — The Influence of Representation. In Poli, R., Nordin, P., Langdon, W. B., and Fogarty, T. C., editors, *Proceedings of the Second European Workshop on Genetic Programming (EuroGP'99)*, pag. 29–36, Goteborg, Sweden. Springer Verlag.

Machlin, R. and Stout, Q. F. (1990). The Complex Behavior of Simple Machines. *Physica*, D(42):85–98.

Marxen, H. (2002). Busy Beaver. http://www.drb.insel.de/˜heiner/BB/.

Marxen, H. and Buntrock, J. (1990). Attacking the Busy Beaver 5. *Bulletin of the European Association for Theoretical Computer Science*, 40:247–251.

Mitchel, M., Crutchfield, J. P., Hraber, P.T. (1994). Evolving cellular automata to perform computations: Mechanisms and Impediments. *Physica D* 75:361-391.

Pereira, F. B. (2002). *Estudo das Interacções entre Evolução e Aprendizagem em Ambientes de Computação Evolucionária*. PhD thesis, Universidade de Coimbra, Coimbra, Portugal.

Pereira, F. B., Machado, P., Costa, E., and Cardoso, A. (1999a). Busy Beaver — An Evolutionary Approach. In Ochoa, A., Ortiz, M., and Santana, R., editors, *Procs. of the Second International Symposium on Artificial Intelligence and Adaptive Systems (CIMAF'99)*, pag. 212–217, Havana, Cuba.

Pereira, F. B., Machado, P., Costa, E., and Cardoso, A. (1999b). Graph Based Crossover — A Case Study with the Busy Beaver Problem. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of*

the Genetic and Evolutionary Computation Conference, volume 2, pag. 1149–1155, Orlando, Florida, USA. Morgan Kaufmann.

Rado, T. (1962). On non-computable functions. *The Bell System Technical Journal*, 41(3):887–884.

Teller, A. and Veloso, M. (1996). PADO: A New Learning Architecture for Object Recognition. In Ikeuchi, K. and Veloso, M., editors, *Symbolic Visual Learning*, pag. 81–116. Oxford University Press.

Turing, A. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, Series 2(42):230–265.

Wood, D. (1987). *Theory of Computation*. Harper and Row, New York.