Busy Beaver – The Influence of Representation

Penousal Machado*, Francisco B. Pereira*, Amílcar Cardoso**, Ernesto Costa**

Centro de Informática e Sistemas da Universidade de Coimbra

{machado, xico, amilcar, ernesto}@dei.uc.pt

Abstract. The Busy Beaver is an interesting theoretical problem proposed by Rado in 1962. In this paper we propose an evolutionary approach to this problem. We will focus on the representational issues, proposing alternative ways of codifying and interpreting Turing Machines. These alternative representations take advantage of the existence of equivalent Turing machine sets. The experimental results show that the proposed representations provide improvement over the "standard" genetic codification.

1 Introduction

One of the most important results of theoretical computer science deals with the existence of non computable functions. This fact can be easily established showing that there are functions, which are not Turing computable: there are more functions than Turing Machines to compute them.

In 1962 Tibor Rado proposed one such function based on what is known today as the "Busy Beaver Game or Problem" [8]. It can be described as follows: Suppose a Turing Machine (TM) with a two way infinite tape and a tape alphabet = {blank, **1**}. The question Rado asked was: What is the maximum number of **1**'s that can be written by an N-State[1] halting TM when started on a blank tape? This number, which is function of the number of states, is denoted by $\Sigma(N)$. A machine that produces $\Sigma(N)$ non-blank cells is called a Busy Beaver (BB).

The problem with $\Sigma(N)$ is that it grows faster than any computable function, i.e., $\Sigma(N)$ is non computable. Some values for $\Sigma(N)$, and the corresponding TM's are known today for small values of N. We have, for instance, $\Sigma(1) = 1$, $\Sigma(2) = 4$, $\Sigma(3) = 6$, $\Sigma(4) = 13$. As the number of states increases the problem becomes harder, and, for N >= 5, we have several candidates (or contenders) which set lower bounds on the value of $\Sigma(N)$. This is partially due to the fact that there is, neither a general, nor a particular theory about the structure of a BB. The only available technique for finding such machines is to perform an exhaustive search over the space of all N-state TM.

Due to these difficulties, the Busy Beaver problem has attracted the attention of many researchers and several contests were organised trying to produce the best

---

* Instituto Superior de Engenharia de Coimbra

** Departamento de Engenharia Informática da Universidade de Coimbra

[1] N does not include the final state.

candidates. The used techniques perform a partial search on the solution space, looking for TM which produce the best lower bound for the value of $\Sigma(N)$. Some of the best contenders were obtained be Marxen [6] (e.g., he established that $\Sigma(5) >= 4098$). His approach involves enumeration and simulation of all N-state TMs, using several techniques to reduce the number of inspected machines, accelerate simulation and determine non-termination.

In the original setting, the problem was defined for 5-tuple TMs. With this definition, machines, given a current state and the symbol being scanned in the tape, write a symbol over it, enter a new state and move the read/write head left or right. One of the main variants consists in considering 4-tuples TM. The main difference from the others is that, during the transition to a new state, a TM either writes a new symbol to the tape or moves the head (both actions are not simultaneously allowed).

In our research, we focus on the 4-tuple TMs variant. We use an evolutionary approach, which has already proved to be extremely effective. In a recent work [7], we presented new best lower bounds for 4-tuple BB(7) and BB(8), showing that $\Sigma(7)>=102$ and that $\Sigma(8)>=384$. If we compare these results with previous best candidates ($\Sigma(7)>=37$[5] and $\Sigma(8)>=84$), it clear that our approach can bring significant improvements. These new lower bounds were found in less than one day, using a 300 MHz Pentium II computer. Only (8.5e-11)% of the search space was evaluated.

In this paper, we will focus our attention on representational issues. We will propose and study three different representations for TMs. The results achieved prove that this is a very important issue when searching for good BB candidates. We will use BB(6) as a testbed for the proposed representations. The current best candidate for BB(6) writes 21 **1**'s, and was first proposed by Chris Nielsen. Our goal is to determine which representation allows us to find this solution in a consistent way.

The paper as the following structure: Section 2 comprises a formal definition of five and four tuple TM, and the specification of the rules of the Busy Beaver problem for each of these variants. In Section 3 we present three way of representing and interpreting TMs. Section 4 relates to the simulation and evaluation of TMs. In Section 5 we present the experimental results, which are analysed in Section 6. Finally in Section 7 we state some overall conclusion and suggest some directions for future work.

2 Problem Definition

A deterministic TM can be specified by a sextuple $(Q,\Pi,\Gamma,\delta,s,f)$, where[9]:

- 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.

The original definition, proposed by Rado [8], considered deterministic 5-tuple TMs with N+1 states (N states and an anonymous halt state). The tape alphabet has two symbols, Γ={blank,**1**}, and the input alphabets has one, Π={**1**}.

The productivity of a TM is defined as the number of **1**'s present, on the initially blank tape, when the machine halts. Machines that do not halt have productivity zero. $\sum$(N) is defined as the maximum productivity that can be achieved by a N-state TM. This TM is called a Busy Beaver.

In the 4-tuple variant productivity is usually 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 [1]. Thus, accordingly to these rules in the 4-tuple variant, the machine must halt when reading a **1**, this **1** must be the leftmost of a string of **1**s and, with the exception of this string, the tape must be blank.

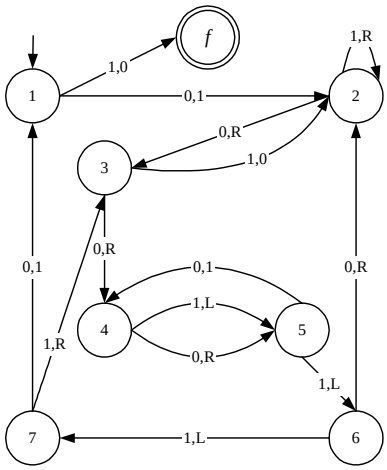
3 Representation

Genetic Algorithms (GAs) are probabilistic search procedures based on the principles of natural selection and genetics [3]. They have been used to solve hard problems (those with a huge and multimodal space to search). Typically they only need to explore a small portion of the space. In simple terms, GAs iteratively evolve a population, that is set of solutions' candidates (points of the search space, called individuals) using genetic operators, until some conditions are met. They start from a random generated set of potential solutions. Each individual is defined by its chromosome.

As stated before, a 4-tuple 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={blank, **1**}, we only need to represent the transition function, δ: Q×{blank,**1**} → Q×{L,R,blank,**1**}. Fig.1 shows a 4-tuple TM and its transition table.

In our approach, the chromosome of each individual will be represented trough a binary string with the following format:

The encoding of each *New State* requires three bits and the encoding of each *Action* requires two bits. This gives 10 bits per state and thus 10*N bits to represent an N state 4-tuple TMs.



δ	By blank		By one	
State	New State	Action	New State	Action
1	2	1	*f*	L
2	3	R	2	R
3	4	R	2	0
4	5	R	5	L
5	4	1	6	L
6	2	R	7	L
7	1	1	3	R

Fig. 1. A seven state 4-tuple TM and its corresponding transition table. The blank symbol is represented by 0. This machine is the best known 4-tuple BB(7) candidate [7], it writes 102 **1**s in 4955 transitions.

It is clear that there are several TMs that exhibit the same behaviour, this machines can be considered equivalent. If we could construct sets of machines with equivalent behaviour we would only need to run one of the machines of the set. The most important of these equivalent classes is known as the *Tree Normal Form* (TNF) [6]. Using a TNF representation ensures that machines differing only in the naming of the states or in transition that never are used, are represented in the same way. We can convert a machine to its TNF by running the machine and numbering the states in the order that they are visited. States that were never visited and unused transitions are deleted. It would be nice to represent the machines in the TNF. Unfortunately, to convert a machine to its TNF (or to know if it is in TNF) we have to run it. There are two possibilities to be considered:

- Directly decoding the chromosome to a TM, and thus not taking advantage of equivalence classes.
- Interpret the machine codified in the chromosome, as if it was in TNF. This can be achieved by using the algorithm presented in Fig.2.

```
Mark all transitions as undefined
Mark all the states as not-visited
N_of_visited_states ← 0
N_of_defined_transitions ← 0
CState ← 1                          //Current State
While (CState ≠ Halting State) and
      (Limit number of transitions not reached)
  Read the symbol on the tape to CSymbol
  If the transition (δ:CState×CSymbol) is undefined
      Mark it as defined
      Increase the number of defined transitions
      If the Current State is not-visited
          Mark it as visited
          Increase the number of visited states
      If state (δ:CState×CSymbol→Q) > (N_of_visited_states+1)
          Set (δ:CState×CSymbol→Q) to N_of_visited_states
      If this is the last undefined transition
          Set (δ:CState×CSymbol→Q) to Halting State
  Perform the action indicated by (δ:CState×CSymbol)
  //i.e. move the head or write a symbol to the tape
  Set Cstate to (δ:CState×CSymbol→Q)
```

Fig. 2. During the process of simulation of the TM we can choose to interpret it as if it where in the Tree Normal Form. The simulation process ends when the machine reaches its halting state, or when a predefined number of transitions is met (this will be explained in Section 4).

A further possibility is to code back the changes induced by the TNF interpretation of the machine, i.e. modify the original chromosome to one with the machine in TNF format. Thus, we have three different options for the representation and interpretation of TMs: using a standard representation (*Standard*), use a tree normal form interpretation of the machine (*TNF*), or the TNF interpretation combined with back coding of the resulting phenotype (*Backcoding*).

4 Simulation and Evaluation

The evaluation phase involves the interpretation of each chromosome and simulation of the resulting TM. Due to the Halting Problem we must establish a limit for the number of transitions. Machines that don't halt before this limit are considered non-halting TMs. We keep track of the maximum number of transitions (MaxT) made

by a TM before halting and set the limit of transitions to ten times this number. Thus, when the GA finds a machine halting after a number of transitions higher than MaxT the limit is increased.

To assign fitness we consider the following factors [7], in decreasing order of importance:

1. Halting before reaching the predefined limit for the number of transitions.
2. Accordance to the rules [1].
3. Productivity.
4. Number of used transitions.
5. Number of steps made before halting.

This seems to yield better results than using the productivity, alone, as fitness function [7]. The idea is to establish and explore the differences between "bad" individuals, e.g., a machine that never leaves state 1 is considered worse than one that goes trough all the states, even if they have the same productivity.

5 Experimental results

The experiments were performed using GALLOPS 3.2 [2]. The parameters of the GA were the following: Number of Evaluations={40 000 000}; Population Size={100, 500};Two-point crossover restricted to gene boundaries; Crossover rate=70%; Single point mutation; Mutation rate={1%,5%,10%}; Elitist strategy; Tournament selection; Tourney size={2,5}. Each experiment was repeated thirty times with the same initial conditions and different random seeds.

A brief perusal of the result graphs in Fig.3 shows that *TNF* and *BackCoding* clearly outperform *Standard* interpretation. It is also visible that *TNF* is usually better than *BackCoding*, although in a lesser scale. Another interesting result is the need for a fairly high mutation rate. The 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 perform better.

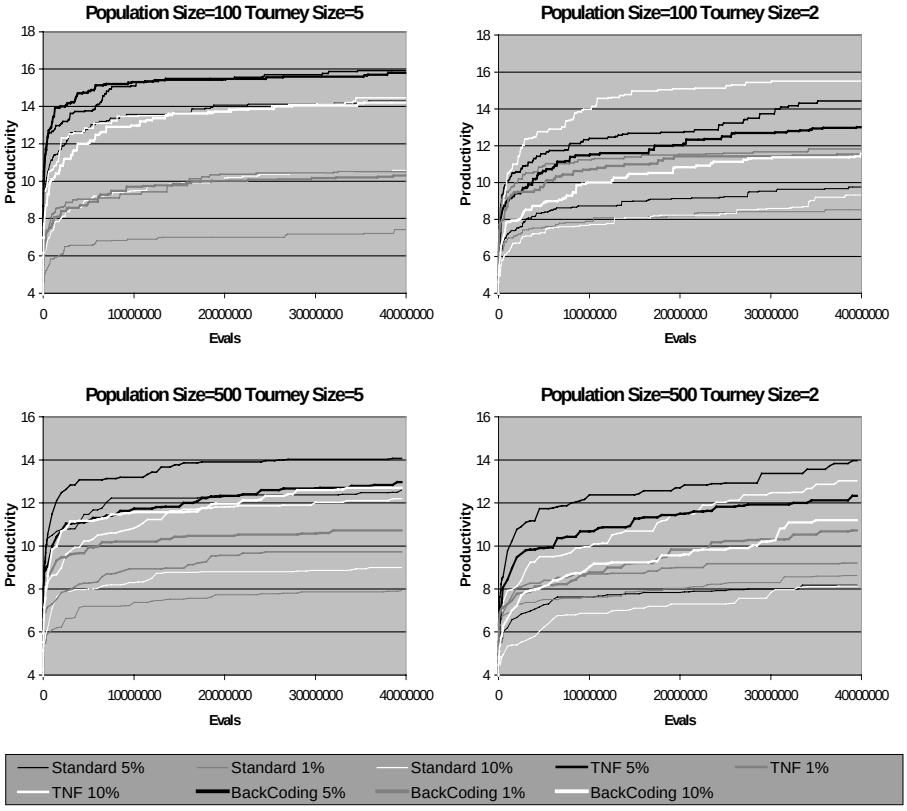


Fig. 3. The charts show the number of ones written by the best individual. The results are the average of series of 30 runs.

6 Analysis of the Results

In Table 1 we indicate the number of times that the best 4-tuple BB(6) candidate (21 **1**s in 125 transitions) was found. This table confirms and emphasizes the results shown in the previous section. Using a *Standard* representation the GA only reached the maximum once in 360 runs. This result is in accordance to the ones presented in [4] for the 5-Tuple BB(4), showing that the busy beaver can be a complex problem, and that the space is difficult to search by an evolutionary approach when using standard representation. When using *TNF* the maximum was reached 17 times, approximately twice as much then when using *BackCoding*.

		Standard				TNF				BackCoding			
	Pop. Size	100		500		100		500		100		500	
	Tourney 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			

Table 1. Number of runs in which the maximum was reached. Blank cells indicate that none of the 30 runs reached the maximum.

Using *TNF* reduces significantly the search space since isomorphic machines will have the same representation. It is important to notice that the solution space is also decreased. When using a standard representation there are several isomorphic solutions while in *TNF* these became only one. Therefore, this reduction of the problem space doesn't explain the difference between the performance of *TNF* and *Standard*. There is, however, another type of problem space reduction in *TNF*. A careful observation of the interpretation algorithm presented in Fig. 2, will reveal that in *TNF* the machines are only allowed to enter the final state after visiting all other states. Thus, in *TNF*, machines that halt always visit all the states, allowing the development of the complex behaviour, required to find good candidates. In *Standard* representation, a machine can halt after visiting a small number of states. Although these machines may have a simple behaviour, they will still have a fitness score higher than most of the individuals of their generation, and will tend to dominate the populations hindering the formation of building blocks. The chart in Fig.4 shows the average number of visited states for *Standard*, *TNF* and *BackCoding*. Using *TNF* yields an average of 5.5 while *Standard* representation only achieves a 4.6 average. Another interesting characteristic of *TNF* representation is that it induces an ordering of the states. States that are directly connected have a higher probability of being close in the chromosome[2]. Thus, there is a higher similarity between genotype neighbourhood and phenotype neighbourhood.

[2]E.g. In *Standard* representation we can jump directly from state 1 to state 6. In *TNF* this is only possible if states 2 through 5 have already been visited.

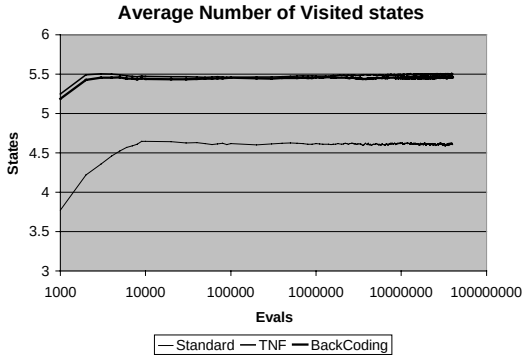


Fig. 4. Average number of visited states

The difference in the performance of *TNF* and *BackCoding* is less visible, and only becomes clear when considering the results in Table 1. With *BackCoding*, changes in the TM produced by *TNF* simulation are coded back to the genotype and directly passed to the descendants. This reduces the diversity of the population, which may explain the difference in the results.

Mutation plays an important role in the evolutionary process. The charts show that BB requires a high mutation rate. With 1% mutation rate the best candidate was never found. It is perceptible both on the charts and on Table 1 that 5% mutation rate is slightly better than 10%. The need for high mutation rates is an interesting result by itself, and may indicate that the fitness landscape is highly irregular and hard to sample.

An interesting and "hidden" difference between *TNF* and *BackCoding* is that *TNF* allows the accumulation of neutral mutations. Consider that we have a chromosome whose gene value for *New State* of the transition of state two by blank is 3; admit also that, when this transition is first used, state three was not yet visited. The alteration of this value to 6 won't produce any changes in the phenotype, since the *TNF* interpretation would still consider it as a transition to state 3. These neutral mutations may become effective, sometime later, due to other alterations of the chromosome. With *BackCoding* the transfer of the phenotype to the genotype eliminates neutral mutations.

7 Conclusions and Further Work

In this paper we explored the importance of representation and interpretation, in the search for good candidates for BB(6) using an evolutionary approach. We proposed an alternative interpretation of the standard TM representation, *TNF*. We also considered the possibility of back coding this interpretation to the genotype. In order to assess the performance of the different representations we conducted a comprehensive set of experiments. The results show that *TNF* clearly outperforms the standard representation, enabling the discovery of the best candidate for BB(6). The

addition of back coding to TNF does not improve the results. Preliminary results indicate that this results are extensible to BB(7) and BB(8). The combination of TNF with a fitness function that explores the differences between bad machines makes evolutionary computation an excellent approach to find good candidates for the BB problem.

As future work we intend to include and test several learning models. The addition of non-standard high level genetic operators, designed to take advantage of TNF representation may be an interesting research direction. We are also considering a distributed GA implementation, with several populations and migration mechanisms.

To attack BB(9) and higher, we need to speed up the simulation of the TMs. Among the possibilities are: using macro-transitions, avoiding evaluation of equivalent machines and early detection of non-halting machines [6].

8 Acknowledgments

This work was partially funded by the Portuguese Ministery of Science and Technology, under Program PRAXIS XXI.

9 References

1. Boolos, G., and Jeffrey, R. (1995). *Computability and Logic*, Cambridge University Press.
2. Goodman, E. (1996). GALOPPS (Release 3.2 – July, 1996), The Genetic Algorithm Optimized for Portability and Parallelism System, Technical Report #96-07-01, Michigan State University.
3. Holland, J. (1975) *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press.
4. Jones, T., Rawlins, G. (1993) Reverse HillClimbing, Genetic Algorithms and the Busy Beaver Problem, In Forrest, S. (Ed.), *Genetic Algorithms: Proceedings of the Fifth International Conference (ICGA-93)*. San Mateo, CA: Morgan Kaufmann, pp 70-75.
5. Lally, A., Reineke, J., and Weader, J. (1997). An Abstract Representation of Busy Beaver Candidate Turing Machines.
6. Marxen, H. Buntrock, J. (1990). Attacking Busy Beaver 5, *Bulletin of the European Association for Theorethical Computer Science*, Vol 40.
7. Pereira, F. B., Machado, P., Costa, E. and Cardoso, A. (1999). Busy Beaver: An Evolutionary Approach. To be published in the *Proceedings of the 2nd Symposium on Artificial Intelligence (CIMAF-99)*, Havana, Cuba.
8. Rado, T. (1962) On non-computable functions, *The Bell System Technical Journal*, vol. 41, no. 3, pp.877-884.
9. Wood, D. (1987). *Theory of Computation*, Harper & Row, Publishers.