Graph Based Crossover – A Case Study with the Busy Beaver Problem

Francisco B. Pereira
ISEC
Qta. da Nora, 3030
Coimbra, Portugal
xico@dei.uc.pt

Penousal Machado
ISEC
machado@dei.uc.pt

Ernesto Costa
CISUC
Dep. de Eng. Informática,
Univ. of Coimbra, Polo II,
3030 Coimbra, Portugal
ernesto@dei.uc.pt

Amílcar Cardoso
CISUC
amilcar@dei.uc.pt

Abstract

The success of the application of evolutionary approaches depends, to a large extent, on problem representation and on the used genetic operators. In this paper we introduce a new graph based crossover operator and compare it with classical two-point crossover. The study was carried out using a theoretical hard problem known as Busy Beaver. This problem involves the search for the Turing Machine that produces the maximum number of ones when started on a blank tape. Experimental results show that, in this domain, the new graph-based operator provides a clear advantage over two-point crossover.

1 INTRODUCTION

Genetic operators play a particular important role in Evolutionary Computation. The task of recombination operators is to promote the exchange of genetic material between individuals. Typically, the operators used depend on the problem being solved and on the chosen representation. When using linear representations the most common operators are single-point, two-point and uniform crossover [Mitchell, 1996]. If a structured representation (e.g. trees) is adopted, it is advantageous to define recombination operators suitable for this structure [Angeline, 1994]. An example of such an operator is the standard GP crossover that exchanges sub-trees between individuals [Koza, 1992].

For some problems the natural representation is a graph. The main goal of this paper is to introduce a graph-based crossover operator and study its application to this type of problems. The application domain is the Busy Beaver (BB) problem [Rado, 1962] which involves the representation of Turing Machines (TMs). Tibor Rado introduced BB in the context of the existence of non-computable functions. It can be described as follows: Suppose a TM with a two way infinite tape and 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 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. $\Sigma(N)$ happens to be non-computable.

Some values for $\Sigma(N)$, and the corresponding TM's are known today for small values of N (e.g. $\Sigma(1)=1$, $\Sigma(2)=4$, $\Sigma(3)=6$, $\Sigma(4)=13$). As the number of states increases the problem becomes harder, and, for N>=5, there are 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. To prove that a machine is the N-state BB, we must perform an exhaustive search over the space of all N-state TMs and prove that no other machine produces a higher number of ones. This becomes extremely complex due to the halting problem.

The BB is one of the most interesting theoretical problems. It has attracted the attention of many researchers and several contests were organized trying to produce the best candidates. The used techniques perform a partial search on the solution space, looking for TMs that produce the best lower bound for the value of $\Sigma(N)$. Some of the best contenders were obtained by [Marxen and Buntrock, 1990] (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 decide non-termination.

In the original setting, the problem was defined for 5-tuple TMs. This type of machines, given a current state and symbol, write a new symbol, enter a new state and move the read/write head left or right. One of the main variants consists in considering 4-tuple TMs. These machines, during the transition to a new state, either write a new symbol to the tape or move the head (the actions are not allowed simultaneously).

In [Pereira et al, 1999] we used an evolutionary approach to the BB problem, and set new best lower bounds for

4-tuple BB(7) and BB(8), showing that $\sum(7) >= 102$ and that $\sum(8) >= 384$. If we compare these results with previous best candidates ($\sum(7)>=37$ and $\sum(8) >= 84$), it is clear that the used approach brings significant improvements. In previous studies we used two-point crossover. Here we will study the application of a graph-crossover operator to BB. An extensive set of experiments was performed and the results show that it clearly outperforms classical two-point crossover. Additionally, the use of this new operator allowed the discover of new best candidates for BB(6) and BB(7), setting the new records in 25 and 164 (The previous best BB(6) candidate was found by Chris Nielsen and writes 21 **1**s).

The paper is organized as follows: section 2 comprises a formal definition of 5 and 4-tuple TMs and the rules for the BB problem. In section 3, we present our graph-based crossover operator. In section 4, we discuss several ways of interpreting TMs. Section 5 relates to the simulation and evaluation of TMs. In section 6, we present and discuss the results of comparative experiments between classical crossover and graph-crossover. Section 7 concerns ongoing research work. Finally, in section 8, we draw some conclusions.

2 PROBLEM DEFINITION

A deterministic TM can be specified by a sextuple (Q,Π,Γ,δ,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 the head 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.

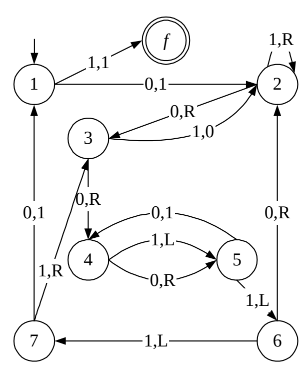
The original definition [Rado, 1962] considered deterministic 5-tuple TMs with N+1 states (N states and an anonymous halt state). The tape alphabet has two symbols, Γ={blank, **1**}, and the input alphabet has one, Π={**1**}. The productivity of a TM is defined as the number of **1**s present, on the initially blank tape, when the machine halts. Machines that do not halt have productivity zero. Σ(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 (Boolos, 1995). Thus, 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 A GRAPH CROSSOVER OPERATOR

In GP individuals are usually represented as trees; the genetic operators consider this representation, i.e., they "see" the individuals as trees and preserve the inherent syntactic restrictions [Angeline, 1994]. In our approach the individuals are N-State TMs and their natural representation is a directed graph with N+1 nodes. The main idea of our crossover operator is the exchange of sub-graphs between individuals.

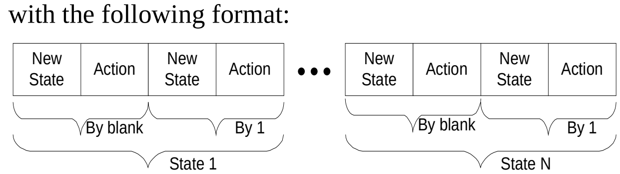
As stated before, a 4-tuple TM can be defined by a sextuple (Q,Π,Γ,δ,s,f). Without loss of generality, we can consider Q={1,2,…,N,N+1}, set 1 as the initial state and N+1 as the final one. Since Π={**1**} and Γ={blank, **1**}, we only need to represent the transition function, δ:Q×{blank,**1**}→Q×{L,R,blank,**1**}. Figure 1 shows a TM and the corresponding transition table.



δ	By blank		By one	
Q	New State	Action	New State	Action
1	2	1	*f*	1
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

Figure 1: A 4-tuple TM and its transition table. The blank symbol is represented by a **0**. This machine writes 102 **1**s in 4955 transitions [Pereira et al, 1999].

Therefore, each individual is coded by an integer string with the following format:



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. 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. These sub-machines are composed by nodes (and corresponding arcs) that are functionally dependent. In TMs functional dependency is usually related to the distance between nodes (considering minimum path length). Thus, the probability of two nodes being functionally dependent is higher if they are directly connected. Therefore, our sub-graphs will be composed by a subset of closely linked nodes and by the linking transitions, i.e. the transitions among them. We will call this type of transitions internal. An illustrated description of the crossover algorithm follows.

To perform crossover between individuals *A* and *B*, we start by randomly selecting the crossover points (states) P_A and P_B, and a crossover size *X* (the number of states of each sub-graph). Next, we will determine the sub-graphs, S_A and S_B. Each sub-graph will be composed by the set of the *X* closest states from the corresponding crossover point and by their internal transitions. Distance between two states is defined as the length of minimum path connecting them in the directed graph. If there is no path connecting the two states the distance is infinite. Thus, to construct each sub-graph we start by determining the set of states to be included, which is achieved through breadth first search. The exclusion of states at the same distance is done randomly. This process yields two lists of states, L_A and L_B, ordered according to the distance from the crossover point and with the order of states at the same distance chosen randomly. Since we are evolving TMs of fixed size we need to ensure that the lists contain the same number of nodes[1]. When the lists have different sizes the largest is truncated.

The internal transitions of S_A will be replaced by the internal transitions of S_B and vice-versa. This implies establishing a correspondence between states. We chose to define correspondence based on the position of the states in the lists L_A and L_B (i.e. nodes at the same position are correspondent). In Figure 2 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 2, the transition $2\times\mathbf{0}\rightarrow4\times$R of individual *A* is not replaced by $6\times\mathbf{0}\rightarrow1\times$R since this transition is not internal.

[1] *X* is the maximum crossover size. There are situations where it is impossible to reach *X* states when starting from the crossover point.

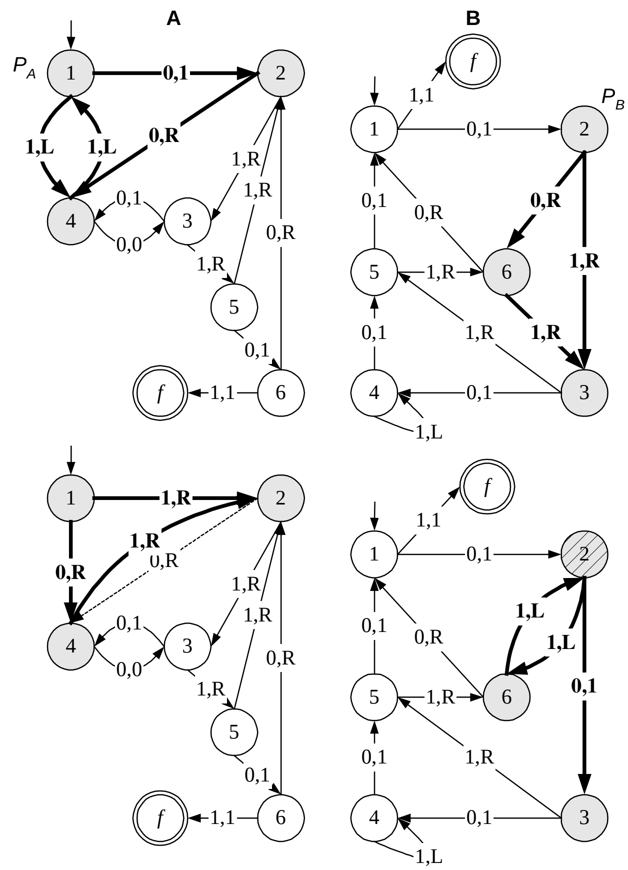


Figure 2: Example of Crossover. Nodes belonging to the sub-graphs are depicted in gray and transitions in bold. A dashed line represents internal transitions that weren't replaced.

4 CHROMOSOME INTERPRETATION

It is clear that there are several TMs that exhibit the same behavior; these machines can be considered equivalent. If we construct sets of machines with equivalent behavior we only need to run one of the machines of the set. The most important equivalent class is 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.

It would be nice to represent the machines in the TNF. Unfortunately, to convert a machine to its TNF we have to run it, number the states in the order that they are visited, and delete unused states and transitions. Therefore, we can consider the following alternatives:

- 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 [Machado et al, 1999].

A further possibility is to code back the changes induced by the TNF interpretation, 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*).

The importance of the selected representation was subject of a previous study [Machado et al, 1999]. The central conclusion was that *TNF* and *BackCoding* clearly outperform *Standard* interpretation. It was also visible that *TNF* performs better than *BackCoding*, although in a lesser scale. Another interesting result is the need for a fairly high mutation rate, which may indicate that the fitness landscape is highly irregular and hard to sample. These results were obtained with two-point crossover.

5 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 (MaxT). Machines that don't halt before this limit are considered non-halting TMs.

To assign fitness we consider the following factors [Pereira et al, 1999], in decreasing order of importance:

- Halting before reaching the predefined limit for the number of transitions.
- Accordance to the rules [Boolos and Jeffrey, 1995].
- Productivity.
- Number of used transitions.
- Number of steps made before halting.

This approach yields better results than using the productivity, alone, as fitness function [Pereira et al, 1999]. 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 through all the states, even if they have the same productivity.

6 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 transitions we set MaxT to 250. The experiments were performed using GALLOPS 3.2 [Goodman, 1996].

The parameters of the Evolutionary Algorithm (EA) were the following: TM representation = {Standard, TNF, BackCoding}; Number of Evaluations = {40 000 000}; Population Size= {100, 500}; Generation Gap=1; Crossover Operator = {Two-point, Graph}; Crossover rate = 70%; Single point mutation; Mutation rate = {1%, 5%, 10%}; Elitist strategy; Tournament selection; Tournament size = {2,5}. Two-point crossover was restricted to gene boundaries. Maximum graph crossover size = 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 1 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, {BackCoding, 5%, 500, 5}, achieves 18.6 average productivity.

It is also important to consider the evolution of productivity of the best individual over time. In Figure 4 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

Table 1: 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.

		Two-Point												**Graph**											
		Standard				**TNF**				**BackCoding**				**Standard**				**TNF**				**BackCoding**			
	Pop.	100		500		100		500		100		500		100		500		100		500		100		500	
	T. Size	2	5	2	5	2	5	2	5	2	5	2	5	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	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%	9.8	14.3	8.2	12.6	14.4	15.9	14.0	14.1	13.0	15.8	12.3	13.0	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%	9.3	10.6	8.1	9.0	15.5	14.5	13.0	12.2	11.6	14.2	11.2	12.7	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	9.2	10.8	8.3	9.8	13.9	13.6	12.1	12.0	12.0	13.4	11.4	12.1	12.2	12.9	10.9	12.3	14.7	15.4	13.4	14.2	13.5	14.5	13.2	14.9
		9.98		9.07		13.77		12.03		12.73		11.77		12.56		11.61		15.03		13.76		14.04		14.05	
		9.53				12.90				12.25				12.08				14.39				14.04			
		11.56												13.50											

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.

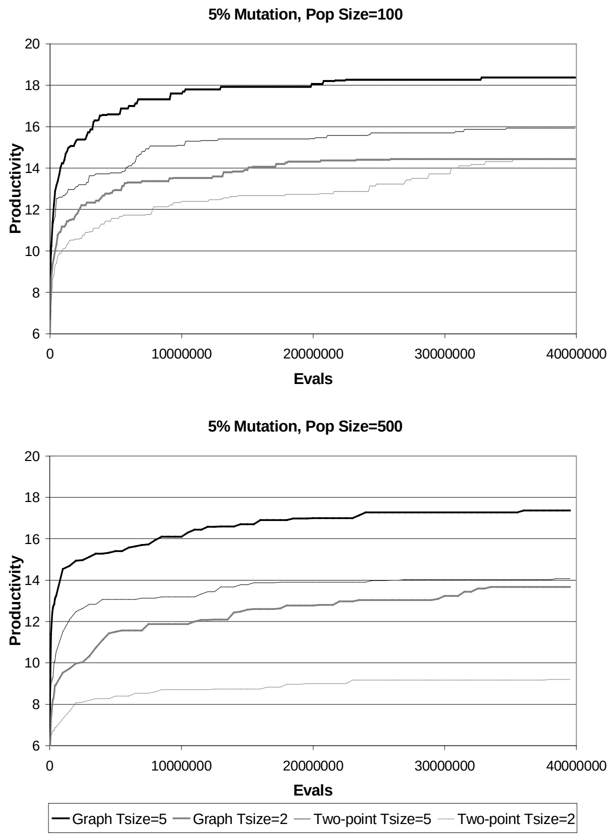


Figure. 4: The charts show the number of ones written by the best individual using a *TNF* interpretation. The results are averages of series of 30 runs.

The chosen mutation rates (1%, 5%, 10%) may seem a little odd. In a previous study [Machado et al, 1999] we showed that, when using two-point crossover, high mutation rates were required in order to find good candidates. This fact was explained by a tendency of the EA to converge to local maxima. The advantages of using graph crossover diminish as the mutation rate increases. This is shown clearly in Table 2, 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 3 we indicate the number of times that the best 4-tuple BB(6) candidate (21 **1**s in 125 transitions) 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 (confidence level 99%).

The idea behind the proposed crossover operator is to take advantage of the inherent relations in the sub-structures

Table 2: 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 (confidence level 99%).

		Standard				**Totals**	**TNF**				**Totals**	**BackCoding**				**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														

Table 3: Number of runs in which the maximum was reached. Blank Cells indicate that none of the 30 runs reached the maximum.

		Two-Point												Graph											
		Standard				TNF				BackCoding				Standard				TNF				BackCoding			
	Pop. Size	100		500		100		500		100		500		100		500		100		500		100		500	
	T. Size	2	5	2	5	2	5	2	5	2	5	2	5	2	5	2	5	2	5	2	5	2	5	2	5
Mutation	1%														1	1		2		2	1		1	1	
	5%					1	6	2			4	1		1	1		2		11		5		3		8
	10%		1			4	3	1				2						3	1	1		1	1		
Totals			1			5	9	3			4	3		1	2	1	2	5	12	3	6	1	5	1	8
		1				14		3		4		3		3		3		17		9		6		9	
		1				17				7				6				26				15			
		25												47											

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 [Machado et al, 1999] we showed that using a TNF interpretation of the TMs (with or without BackCoding) 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.

7 ONGOING RESEARCH

Our study leaves some open issues that are subject of current research efforts:

- The influence of crossover size.
- The number of exchanged transitions depends on the similarity of the sub-graph's topology (higher similarity usually involves a higher number of exchanged transitions).
- There are alternative ways to construct the lists of nodes of the sub-graphs. They could, for instance, be constructed in way that would maximize the number of exchanged transitions.
- Comparing graph-crossover with other types of recombination operators.

Concerning the BB problem itself, a straightforward modification to the TM simulation enables us to attack, simultaneously, BB(N) and BB(X) for X<N. Thus, when attacking BB(7) we also try to solve BB(6) to BB(1). The presented experimental results concern BB(6). Currently, we are trying to set new lower bounds for BB(7) and BB(8). A new BB(7) candidate was already found. It writes 164 **1**s in 23198 transitions. To our surprise we also discovered a new BB(6) contender. This result was quite unexpected since the previous record, 21 **1**s in 125 transitions, stood by several years. The new machine writes 25 **1**s in 256 transitions (Figure 5). We didn't discover it before because the limit for the number of transitions was set to 250. This new record was only found because we were searching for BB<=7 and thus using an higher limit of transitions.

Our future efforts will be aimed at breaking current BB(8) and BB(9) records in the 4-tuple variant. To achieve this we need to speed up the simulation of the TMs, in order to allow a higher limit for the number of transitions. Among the possibilities are early detection of non-halting machines [Marxen and Buntrock, 1990] and using macro-transitions.

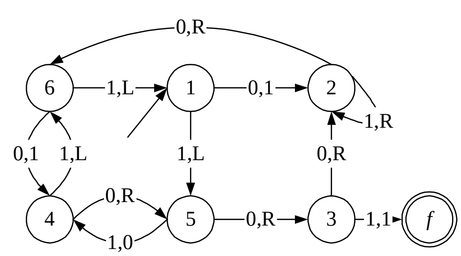


Figure. 5: The new BB(6) best candidate. This machine writes 25 **1**s in 256 transitions.

8 CONCLUSIONS

In this paper we proposed a new recombination operator designed to work with individuals with a graph structure. The application domain was the BB problem, more specifically the search for BB(6) candidates. In order to assess the performance of the operator, we conducted a comprehensive set of experiments comparing it with classical two-point crossover.

The experimental results show that graph crossover clearly outperforms standard two-point crossover. The increase of performance is visible for nearly all considered experimental settings, showing that manipulating the individual in a way that is consistent with its representation improves the results. Preliminary testing indicates that the results are extensible to BB(7) and BB(8). Additionally, new lower bounds for BB(6) and BB(7) were established.

We believe that the presented operator can be applied to other problems for which the natural representation is a graph. Due to the nature of our application domain we only allowed the exchange of sub-graphs with the same number of nodes. However, this constraint can be removed, enabling the application to a wider variety of problems.

Acknowledgments

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

References

Angeline, P. (1994). Genetic programming: A current snapshot. In D. B. Fogel and W. Atmar, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*. Evolutionary Programming Society, 1994.

Boolos, G., and Jeffrey, R. (1995). *Computability and Logic*, Cambridge University Press.

Goodman, E. (1996). GALOPPS, The Genetic Algorithm Optimized for Portability and Parallelism System, *Technical Report #96-07-01*, Michigan State University.

Koza, J. (1992). *Genetic Programming: On the programming of computers by the means of natural selection*. MIT Press.

Machado, P., Pereira, F. B, Cardoso, A., Costa, E. (1999). Busy Beaver – The Influence of Representation, *Proceedings of the Second European Workshop in Genetic Programming,* Göteborg, Sweden.

Marxen, H. Buntrock, J. (1990). Attacking Busy Beaver 5, *Bulletin of the European Association for Theorethical Computer Science,* Vol 40.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Pereira, F. B., Machado, P., Costa, E. and Cardoso, A. (1999). Busy Beaver: An Evolutionary Approach. *Proceedings of the 2nd Symposium on Artificial Intelligence*, Havana, Cuba.

Rado, T. (1962) On non-computable functions, *The Bell System Technical Journal*, vol. 41, no. 3, pp.877-884.

Wood, D. (1987). *Theory of Computation*, Harper & Row, Publishers.