Abstract- The Maximum Homologous Crossover attempts to preserve similar structures from parents by aligning them according to their homology. In this paper, it is successfully tested on the classical Even-N Parity Problem where it demonstrates interesting abilities in bloat reduction. Then, we show that this operator gives an accurate control of the size of programs during the evolution and thus, allows the development of new strategies for the search space exploration.

1 Introduction

1.1 Bloat

Since the early work on Genetic Programming (GP), it has been known that the average size of individuals tends to increase quickly during the evolutionary process, e.g. [8][12]. This phenomenon, called bloat, is one of the most addressed problems of the GP literature. It has been remarked that the main part of the programs of an evolved GP population is constituted of introns, that is code which does not contribute to the fitness. An obvious drawback of bloat is its enormous consumption of resources (memory and execution time). Moreover, when the system is bloating, the recombination of individuals may have no effects since introns will be usually exchanged and may lead to stagnation. However, introns can be useful, for example, by storing the past in dynamic environments. It exists many theories, that are still disputed, to explain the bloat. To name a few, Altenberg [1] notes that bloat arises during evolution as the population attempts to protect useful subtrees from the crossover effects. This is the ‘defense against crossover’. On the other hand, in [10], Langdon and Poli argue that ‘fitness causes bloat’. The idea is that the search starts from short genotypes with a given fitness. Then after a while, since the chance to find better solutions is little, the process becomes neutral and only equally fitted solutions can be retained. But the search space contains many more long genotypes than short ones with the same fitness. Various methods have been proposed to solve the size problem, for example, the variable fitness [20], the parsimony pressure [8] or some specific crossover operators (see below).

1.2 Linear GP

For GP, as for Evolutionary Computation in general, the way individuals are represented is crucial. The representation induces choices about operators and may strongly influence the performance of the algorithm. The emergence of GP in the scientific community arose with the use, inter alia, of a tree-based representation, in particular with the use of Lisp language in the work of Koza [8]. However, GP systems manipulating linear structures exist [3][15], which have shown experimental performances equivalent to Tree GP (TGP). In contrast to TGP, Linear GP (LGP) programs are sequences of instructions of an imperative language (C, machine code, . . . ). In this paper, we focus on LGP mainly because it allows direct access to instructions and so it provides an easier way to perform recombination. Moreover, in LGP all possible sequences of instructions are valid programs, so there are no syntactical constraints on sequences swapped during recombination, which is not true for TGP.

1.3 Crossover and Homology

The role played by crossover in the GP evolutionary process is a much debated question. In the context of TGP, crossover consists in swapping subtrees. According to Koza [8], crossover is the central operator in the GP search process, where useful subtrees tend to spread as they are swapped. However, Banzhaf et al.[20] argue that crossover behaves more like a macro mutation that like an effective recombination operator. In this case GP should be viewed as a population based hill-climber. Similarly, some authors [2] have obtained worse results for crossover compared to mutation based system. Moreover, standard crossover exchanges subtrees without taking context into account; this is a brutal operation that may prevent the emergence of structured solutions [6]. Finally, Poli and Langdon [16] point out the fact that standard GP crossover is a local and biased operator, which can not explore search the space properly. In the LGP context, the standard crossover blindly exchanges parts of the parental sequences. The previous remarks for TGP still stand here. Some new operators have been designed to overcome the drawbacks of the standard GP crossover.
The main idea behind all those recombination mechanisms is homology. In biology, homology indicates genetic relationship, i.e. the structural relatedness of genomes due to descent from common form. Indeed, reproduction in nature is a smooth process which ensures that offspring will not be so different from ancestors, allowing the structural stability that defines species. Natural crossover does not exchange genes randomly. Indeed, during the second stage of the prophase of meiosis (called zygote), the homologous chromosomes are first aligned according to their similarity before crossover takes place. This implies that genes are swapped with others that represent similar features. We note several previous attempts to improve the effectiveness of crossover which, either implicitly or explicitly, try to better preserve homology, see, for example, [6], [9], [13] and [16].

In Section 2, we summarize the work on the Maximum Homologous Crossover. Then, we present its results obtained on the Even-N Parity Problem. Finally, we investigate how the size of programs can be controlled when homology is taken into account and how this allows the design of original courses through the search space.

2 Maximum Homologous Crossover

In this section, we describe our recombination mechanism mimicking natural crossover. We call it Maximum Homologous Crossover (MHC) because it maximally preserves the homology defined according to an edit distance.

2.1 Edit distance

The MHC preserves homology by computing an alignment that minimizes a metric of dissimilarity between parents. As a metric, we use string edit distance, like Levenshtein distance [11] which has been already used in GP to compute or control diversity [4], or to study the influence of genetic operators [14]. By definition, the edit distance between two programs corresponds to the minimal number of elementary operations (deletion, insertion or substitution of one instruction) required to change one program into the other.

More formally, let us consider $P_x \in P_{\Sigma}$ a program of size $m$ such that $P_x = x_0x_1 \ldots x_{m-1}$, with $x_i \in \Sigma \forall i \in [0, m - 1]$, where $\Sigma$ is a finite set of available instructions. Let $P_x$ and $P_y$ be two programs of size $m$ and $n$ respectively and $e$ be an empty instruction. An alignment $(\overline{P}_x, \overline{P}_y)$ of size $p$ with $\overline{P}_x$ and $\overline{P}_y \in P_{\Sigma \cup \{e\}}$ is:

$$(\overline{P}_x, \overline{P}_y) = \left( \begin{array}{c} P_{x_0} \ P_{x_1} \ldots \ P_{x_{p-1}} \\ P_{y_0} \ P_{y_1} \ldots \ P_{y_{p-1}} \end{array} \right)$$

where:

- $p \in [\max(n, m), n + m]$
- $\mathbf{p}_i = x_j$ or $\mathbf{p}_i = e$ for $i \in [0, p - 1]$ and $j \in [0, m - 1]$
- $\mathbf{y}_i = y_j$ or $\mathbf{y}_i = e$ for $i \in [0, p - 1]$ and $j \in [0, n - 1]$
- $\forall i \in [0, p - 1]$ such that $\mathbf{p}_i = \mathbf{y}_i = e$

An aligned pair of instructions $(\mathbf{p}_i, \mathbf{y}_i)$ indicates either a substitution of $x_j$ by $y_j$, or a deletion of $x_j$ (if $\mathbf{y}_i = e$), or an insertion of $y_j$ (if $\mathbf{p}_i = e$). So, an alignment $(\overline{P}_x, \overline{P}_y)$ may also be viewed as a sequence of operations that transforms $P_x$ into $P_y$. We define the cost $\chi$ of an alignment such as $\chi(\overline{P}_x, \overline{P}_y) = \sum_{i=0}^{p-1} cost(\mathbf{p}_i, \mathbf{y}_i)$ with:

$$cost(\mathbf{p}_i, \mathbf{y}_i) = \begin{cases} C_1 & \text{if } \mathbf{p}_i = e \text{ or } \mathbf{y}_i = e \\ C_2 & \text{else if } \mathbf{p}_i \neq \mathbf{y}_i \\ 0 & \text{else} \end{cases}$$

and $A(P_x, P_y)$ the set of all alignments of $P_x$ and $P_y$, then the distance between $P_x$ and $P_y$ is:

$$D(P_x, P_y) = \min \{ \chi(\overline{P}_x, \overline{P}_y) | (\overline{P}_x, \overline{P}_y) \in A(P_x, P_y) \}$$

As an example, in the first step in Figure 1, the distance between $P_x$ and $P_y$ is 7, i.e. seven operations are required to transform $P_x$ into $P_y$ (5 insertions, 1 deletion and 1 substitution). Each column of the alignment $(\overline{P}_x, \overline{P}_y)$ refers to a program and stores a sequence of instructions with gaps inserted (corresponding to $e$).

2.2 Best Alignment and Recombination

A best alignment $(\overline{P}_x, \overline{P}_y)$ between $P_x$ and $P_y$ is that for which $\chi(\overline{P}_x, \overline{P}_y) = D(P_x, P_y)$ holds. We denote $A^*(P_x, P_y)$ the set of all best alignments between $P_x$ and $P_y$. Computation of $A^*(P_x, P_y)$ can be reasonably performed, using dynamic programming, in $O(nm)$ time complexity. Such an algorithm [7] has also been used to align DNA strings in Bio-Informatics.

In order to perform MHC between $P_x$ and $P_y$, only one alignment $(\overline{P}_x, \overline{P}_y)$ is randomly chosen in $A^*$. Recombination between $\overline{P}_x$ and $\overline{P}_y$ can then take place. Since $\overline{P}_x$ and $\overline{P}_y$ have the same length, classical crossovers existing in Genetic Algorithms (GA) can be performed (1-point, 2-point, uniform, ...). Finally, to get valid children, the $e$ symbols are removed.

By choosing the costs of operations, $C_1$ (insertion or deletion) and $C_2$ (substitution), different sets of best alignments could be defined. We focus on two among them:

- with $C_2=C_1=1$, where a substitution is always preferred to a pair of insertion and deletion. This setting is used to compute the so-called Levenshtein distance.

- with $C_2=2$ and $C_1=1$, where a substitution is chosen as often as a pair of insertion and deletion.
We denote MHC\textsubscript{1} and MHC\textsubscript{2}, the corresponding variants of MHC. Figure 1 gives a 1-point MHC\textsubscript{1} with the recombination site at position 5 between P\textsubscript{x} and P\textsubscript{y} in stack-based representation, producing offspring P\textsubscript{x}' and P\textsubscript{y}'.

### 2.3 Features of Maximum Homologous Crossover

We note that offspring produced with MHC could also be obtained using standard LGP crossover. Indeed, MHC only restricts the choice of possible crossover sites in both parents: for example in Figure 1, the instructions 'IF' from P\textsubscript{x} and P\textsubscript{y} correspond to the same crossover locus in alignment (P\textsubscript{x}, P\textsubscript{y}), so they could not appear together in children. Moreover, MHC modifies the probability of a site selection according to the local homology of parents. In the previous example, the alignment (P\textsubscript{x}, P\textsubscript{y}) gives a probability of 3/11 that the sequence 'AND, OR' in P\textsubscript{x} be broken. Without alignment, this probability would only have been 1/6. This particular behavior increases the disruption rate of the less homologous regions (like 'AND, OR'), where many gaps are present, since they are more involved in site selection. On the other hand, the most homologous regions are less often disrupted.

An interesting property of MHC, is that the more similar the parents are to each other, the more similar the offspring are to their parents; in other words, the distance between parents and offspring is always smaller than the distance between parents. We have found experimentally, by performing MHC between randomly generated programs that $D(P_x, P_x) + D(P_y, P_y) = D(P_x, P_y)$, with $P_x, P_y \in P_2$ and $P_x \in \{MHC(P_x, P_y)\}$. Thus, we can assert that MHC performs more like GA crossover than standard LGP crossover, since it is a global search operator at the beginning of the evolutionary process, and becomes more local as the population diversity falls (decrease of distance).

### 2.4 Previous Work on Maximum Homologous Crossover

In our previous work [5], we defined the Homology Driven Fitness problem (HDF), where the fitness of a candidate program is given by its distance $D$ to a randomly chosen optimum of a given size. Thanks to HDF, we have shown that MHC gives better results than standard LGP crossover when the fitness and the distance to optimum are highly correlated. We also defined the Royal Roads (RR) Landscapes for LGP that are especially designed to highlight the ability of recombination operators to preserve existing structures during evolution. Thus, we have shown that MHC is a less destructive operator than standard LGP crossover on RR.

### 3 Even-N Parity Problem

In this section, we study the ability of MHC to address a classical GP benchmark. We focus on the Even-N Parity problem mainly because recent work [18] on boolean problems describes an efficient implementation of fitness evaluation that highly speeds up the system (in our case 32 times faster).

#### 3.1 Setup

We want to compare the efficiency of two different recombination operators, the standard LGP 1-point crossover (SC) and the 1-point MHC which have very distinct behaviour. This is why the evolutionary parameters’ tuning must be extensively investigated. We have implemented a stack-based GP system (see Perkis[15] for details). We have performed 35 independent runs with various mutation and crossover rates, and 3 different maximum creation sizes. Let us notice that a mutation rate of 1.0 means that each program involved in reproduction will undergo, on average, one insertion, one deletion and one substitution. Populations of 1000 individuals were randomly created according to a maximum creation size of 25, 50 and 75. The instruction set contains: the four problem specific instructions AND, OR, NOT, IF and one stack-GP specific instruction DUP which duplicates the head of the execution stack. We note that no terminals are used. Our preliminary experiments have shown better results when the execution stack is initialized with the $N$ inputs of the parity problem. The evolution, with elitism, maximum program size of 100, 10-tournament selection, and steady-state replacement, took place during 100 generations. We used a T-test with 95% confidence to determine if results were significantly different.
3.2 Results

In Figures 2 and 3, the success rate is reported as a function of respectively, the mutation and crossover probability for Even-10 Parity problem with maximum creation size of 50 (similar settings will be used for the Even-8 parity problem and the other maximum creation sizes investigated). For experiments performed with SC, we see that the success rate depends only slightly on the mutation rate. On the other hand, the use of MHC requires more tuning of the mutation rate. This is not surprising since, as we have already mentioned, MHC behaves more like crossover in GA than does SC, so methods working against premature convergence (like mutation) are necessary. The performances improve linearly with the MHC probability whereas good results are obtained even with a small rate for SC (> 0.4).

Table 1: Best results found on Even-8 Parity.

<table>
<thead>
<tr>
<th>Xover Type</th>
<th>Success Rate Found at Gen.</th>
<th>Size</th>
<th>Effective Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Creation Size = 25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC 100</td>
<td>38(_{\sigma=14})</td>
<td>87(_{\sigma=15})</td>
<td>57(_{\sigma=10})</td>
</tr>
<tr>
<td>MHC(_1) 100</td>
<td>45(_{\sigma=9})</td>
<td>27(_{\sigma=3})</td>
<td>24(_{\sigma=3})</td>
</tr>
<tr>
<td>MHC(_2) 100</td>
<td>52(_{\sigma=12})</td>
<td>35(_{\sigma=5})</td>
<td>30(_{\sigma=4})</td>
</tr>
<tr>
<td>Maximum Creation Size = 50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC 100</td>
<td>40(_{\sigma=12})</td>
<td>84(_{\sigma=17})</td>
<td>59(_{\sigma=12})</td>
</tr>
<tr>
<td>MHC(_1) 100</td>
<td>45(_{\sigma=7})</td>
<td>34(_{\sigma=4})</td>
<td>28(_{\sigma=3})</td>
</tr>
<tr>
<td>MHC(_2) 100</td>
<td>51(_{\sigma=12})</td>
<td>45(_{\sigma=7})</td>
<td>34(_{\sigma=5})</td>
</tr>
<tr>
<td>Maximum Creation Size = 75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC 100</td>
<td>43(_{\sigma=14})</td>
<td>92(_{\sigma=11})</td>
<td>62(_{\sigma=10})</td>
</tr>
<tr>
<td>MHC(_1) 100</td>
<td>46(_{\sigma=17})</td>
<td>42(_{\sigma=7})</td>
<td>33(_{\sigma=5})</td>
</tr>
<tr>
<td>MHC(_2) 97</td>
<td>51(_{\sigma=18})</td>
<td>40(_{\sigma=9})</td>
<td>37(_{\sigma=6})</td>
</tr>
</tbody>
</table>

Table 2 gives results on Even-10 Parity problem for SC, MHC\(_1\) and MHC\(_2\) using the best tuning of parameters found. Using SC, the problem is still easy to solve. When MHC is used, we obtain success rates statistically equivalent only with a maximum creation size of 50. But again, we note a very significant diminution of size and effective size with MHC.

Figure 4 shows the average size of the best program as a function of the crossover probability for Even-10 Parity with a maximum creation size of 75. When SC is used, the average size of the best program is correlated to the operator rate. In contrast, using MHC only the maximum creation size influences the average size of the best. The evolution of average size of the best individual for Even-8 Parity with
Table 2: Best results found on Even-10 Parity.

<table>
<thead>
<tr>
<th>Xover Type</th>
<th>Success Rate</th>
<th>Found at Gen.</th>
<th>Size</th>
<th>Effective Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Creation Size = 25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>97</td>
<td>54(σ=14)</td>
<td>92(σ=12)</td>
<td>68(σ=11)</td>
</tr>
<tr>
<td>MHC_1</td>
<td>94</td>
<td>70(σ=11)</td>
<td>32(σ=3)</td>
<td>30(σ=3)</td>
</tr>
<tr>
<td>MHC_2</td>
<td>88</td>
<td>79(σ=10)</td>
<td>44(σ=5)</td>
<td>38(σ=4)</td>
</tr>
<tr>
<td>Maximum Creation Size = 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>100</td>
<td>58(σ=17)</td>
<td>94(σ=11)</td>
<td>69(σ=13)</td>
</tr>
<tr>
<td>MHC_1</td>
<td>97</td>
<td>76(σ=9)</td>
<td>40(σ=6)</td>
<td>35(σ=3)</td>
</tr>
<tr>
<td>MHC_2</td>
<td>97</td>
<td>70(σ=11)</td>
<td>49(σ=8)</td>
<td>41(σ=6)</td>
</tr>
<tr>
<td>Maximum Creation Size = 75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>97</td>
<td>57(σ=17)</td>
<td>91(σ=12)</td>
<td>66(σ=12)</td>
</tr>
<tr>
<td>MHC_1</td>
<td>80</td>
<td>55(σ=15)</td>
<td>44(σ=6)</td>
<td>33(σ=5)</td>
</tr>
<tr>
<td>MHC_2</td>
<td>68</td>
<td>59(σ=13)</td>
<td>57(σ=10)</td>
<td>46(σ=7)</td>
</tr>
</tbody>
</table>

Figure 4: Average Size of best as a function of the Crossover Probability on Even-10 Parity.

Figure 5: Evolution of size of best on Even-8 Parity with the best Crossover and Mutation Rate found.

Figure 6: Success Rate as a function of the Mutation Rate on Even-10 Parity with the best Crossover Probability found.

4 Size Control

The results reported before have shown that when MHC is used the performance of LGP depends on the tuning of the maximum creation size of programs. Moreover, we have
seen that, when the run starts with an optimal setting of maximum size, performances equivalent to those obtained with SC could be expected and the system does not suffer from bloat. However, we think that such a setting is difficult to predict a priori for a given problem. In the following section, we show that MHC can not explore the space of size of programs but that working with mutation, an accurate control of size is possible.

### 4.1 Size Control without Selective Pressure

In [17][19], authors have shown the biases introduced by SC in the exploration of the size of programs. They concluded that SC do not modify the average length of individuals but leads to an oversampling of shorter programs of the search space. In Figure 7, we have plotted the average size of programs, mean of 35 runs on a flat landscape, with insertion and deletion rates fixed to 1.0, with maximum creation sizes of 50, 250 and 450 and a maximum allowed size of 500. It is clear that when SC is used, the average size of programs in the population is modified, since the three curves converge after 500 generations toward an average size of 100. We think that this unexpected behavior of SC (in conjunction with mutation) comes from the use of two hard thresholds for the minimum (here 0) and maximum program size (here 500). Indeed, when SC produces offspring greater than the size limit, the tails of those programs are simply removed and then the average size of the population decreases. In contrast, when SC produces very short offspring, the number of instructions may be not sufficient so that all the deletion operations could not be applied, and then the average size of the population increases. This is not the case for MHC (see Fig. 8) which only explores the space of program size around the average size of the population, so that the two thresholds are rarely reached.

Now, we use the mutation operator to modify the average size of programs during evolution. In the Figure 9, the mutation rate is set to 1.0 for insertion but neither substitution nor deletion are used. We see that the three average sizes quickly converge to around 150 whereas with MHC, sizes are increased, controlled by the mutation rate, until reaching the maximum allowed (see Fig. 10). We note that the size control works also with when deletion is used alone.

### 4.2 Size Control on Even-N Parity

Here, we want to verify that the size control is also possible in conjunction with the selection operator. As an example, the Figure 11 shows the evolution of average size of individuals for the Even-10 Parity problem with a mutation rate of 2.0 for deletion, 1.0 for insertion and 0 for substitution. This original setting allows a new kind of exploration of the search space. To our knowledge, this is the first time that a GP system starts search form large solutions. We do not pre-
tend that this could be an optimal strategy, but we note that all runs converge toward an optimum. On the other hand, using SC, the average size of programs can not be reduced (see Fig. 12).

Figure 9: Evolution of the average size of the population on a flat landscape using Standard Crossover and Insertion.

Figure 10: Evolution of the average size of the population on Even-10 Parity using MHC₁ and Insertion.

Figure 11: Evolution of the average size of the population on Even-10 Parity using MHC₁ and Deletion.

Figure 12: Evolution of the average size of the population on Even-10 Parity using Standard Crossover and Deletion.

5 Conclusion and Perspectives

Our previous work has pointed out the ability of MHC to preserve building blocks on ad hoc problems, that we called Royal Roads Landscapes for Linear Genetic Programming. In this paper, we test our approach on somewhat more real problems, the Even-N Parity. MHC exhibits almost the same performances than SC, when only the success rate is
taken into account. However, the compactness of the programs found gives a real advantage to MHC. While with SC, the size of individuals exhibits exponential growth, MHC keeps it at reasonable values. This allows us to claim that homology prevents GP from bloat. It is worth noticing that, contrary to some previous attempts to limit bloat, this property is not based on modification of the selection scheme (e.g. fitness penalty or Pareto optimization), but rather on an intrinsic property of the dynamical behavior of the operator. Moreover, the introduction of homology offers a control lever on programs’ size. Indeed, it is not clear how harmful is the bloat. There is an obvious need for an exploration of the size dimension, but too important sizes prevent from converging toward interesting programs. Using MHC, we are able to choose when the size should increase or decrease. This may allow the design of size strategies either exogenous, e.g. predefined cycles of decrease and increase, or endogenous, e.g. size control depending on the state of the population or on its history.

Future work should address at least three issues. First, the validation of our approach needs a confrontation to more complex and more real-like problems. Symbolic regression problems seems to be good candidates for this. Second, the dynamical behavior of MHC needs to be more understood. We still lack explanation on why it prevents from bloating. And last, from the two previous points, we should be able to design clever size strategies, to enhance the performances of GP.

Bibliography


