Size Control with Maximum Homologous Crossover

Michael Defoin Platel, Manuel Clergue, Philippe Collard

Size control with Maximum Homologous Crossover

Michael Defoin Platel, Manuel Clergue and Philippe Collard

Laboratoire I3S, CNRS-Université de Nice Sophia Antipolis

Abstract. Most of the Evolutionary Algorithms handling variable-sized structures, like Genetic Programming, tend to produce too long solutions and the recombination operator used is often considered to be partly responsible of this phenomenon, called *bloat*. The Maximum Homologous Crossover (MHC) preserves similar structures from parents by aligning them according to their homology. This operator has already demonstrated interesting abilities in bloat reduction but also some weaknesses in the exploration of the size of programs during evolution. In this paper, we show that MHC do not induce any specific biases in the distribution of sizes, allowing size control during evolution. Two different methods for size control based on MHC are presented and tested on a symbolic regression problem. Results show that an accurate control of the size is possible while improving performances of MHC.

1 Introduction

One of the major research areas in Genetic Programming (GP) is the management of the size of programs. Indeed, the "natural" trend of GP systems is to quickly increase the size of individuals until they reach the maximum allowed size, a phenomenon commonly known as *bloat*.

1.1 Bloat

This uncontrolled growth of programs is one of the weaknesses of GP as a problem-solver: resources needed by the system to address a problem are not adapted to the difficulty, the system consumes all the resources provided, leading generally to a waste of computing time and memory. Moreover, this behavior may dramatically influence the efficiency of the system in terms of solution quality and it works against the assumption [14] that between two equally fit programs, we should retain the smaller, which is often more robust and more evolvable.

Many authors have proposed explanations for 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 *protection hypothesis*. On the other hand, in [8], 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 of finding better solutions is low, the

process becomes neutral and only equally fit solutions can be retained. But the search space contains many more long genotypes than short ones with the same fitness. This is the *drift hypothesis*. We note that recent work on Exact Schemata Theorems [14] tends to confirm this hypothesis, while giving a theoretical explanation for bloat. In [16], authors give another explanation for bloat by pointing out the asymmetric effects on the fitness of subtrees deletions and insertions. Indeed, they show that when a subtree is removed, the effects on the fitness depend on its size (strong effects for large subtrees) but not in case of a subtree insertion. This is the *removal bias hypothesis*. Another important aspect of the bloat problem is the presence in programs of inviable code, called the introns. Most of bloat theories suggest that the phenomenon is due to the propagation of introns. However, some interesting work [9][10] tends to contradict the *introns hypothesis*.

Various methods have been investigated to solve the size problem. Maybe the widespread idea to control the size, is to modify the fitness of programs and so the selection process. For examples, we can quote : the variable fitness [17], the parsimony pressure [16], the multi-objective evaluation [5] and the Tarpeian method [12]. Another way to tackle the size control problem is based on specific genetic operators, in particular more homologous crossover operators, [13] and [4].

1.2 Maximum Homologous Crossover

The Maximum Homologous Crossover (MHC) [7] is a recombination mechanism mimicking natural crossover that maximally preserves homology between parents. The MHC ensures that the genetic material exchanged during crossover is chosen, according to an *edit distance*[1] , in the most dissimilar regions of parents, and so leaves unchanged their nearly identical parts, *ie* the homologous regions. Thus, offspring can not be very different from their parents.

MHC was originally designed for Linear GP (LGP), where programs are sequences of instructions of an imperative language (C, machine code, ...). Our study is based on a stack-based GP implementation [11] and [3], where a sequence of instruction is evaluated using an operand stack. Figure 1 gives an example of MHC recombination between two programs P_x and P_y in stack-based representation. We see that during Step 1, an alignment $(\overline{P}_x, \overline{P}_y)$ of the two parents is computed, see [7] for details, to identify homologous regions. We note that aligned programs may contain some gaps (ε) and that they always have the same size. Thus, a crossover site can be chosen in $(\overline{P}_x, \overline{P}_y)$, here at position 5, and the classical 1-point crossover used in GA can be used, see Step 2. Finally, in Step 3, the inserted gaps are removed, producing offspring P'_x and P'_y. In a previous study [6], authors have shown, on the Even-N Parity Problem , that MHC is a less destructive operator than the Standard Crossover (SC) used in

[1] The *edit distance* corresponds to the minimal number of elementary operations (deletion, insertion or substitution) required to change one program into the other.

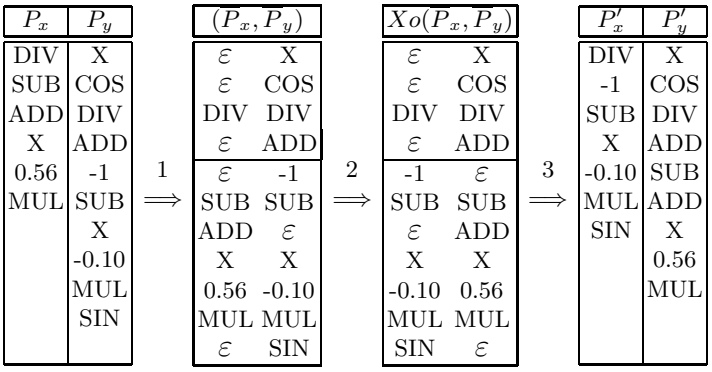

Fig. 1. MHC of programs P_x and P_y in stack-based representation : Step 1, alignment and Xover site selection (here 5); Step 2, swapping sequences ; Step 3, deletion of gaps.

LGP[2]. Moreover the performances of the two crossover operators were very similar but MHC has demonstrated a significant tendency in bloat reduction. The fact that using less desctructive operators allows a kind of reduction in bloating behaviours tends to confirm the *protection hypothesis*. However, an unexpected consequence of this size growth limitation was the need to accurately tune initial sizes in the population. The hypothesis was that MHC is unable to properly manage the size of individuals. In this context, the size may be viewed as a new dimension of the search space that needs some specific operators to be explored. Some experiments, on a flat landscape and on the Even-N Parity Problem, have demonstrated the possibility of controlling the size of programs using MHC.

In this paper, we investigate further, with two methods based on MHC, how to control the size of program during evolution but also how to improve the performances of MHC, as a fully functional recombination mechanism.

2 Size Control with MHC

In [14][15], authors have shown the biases introduced by SC in the exploration of the size of programs. They concluded that, without selective pressure, the distribution of the size converges toward a gamma distribution, *ie* SC does not modify the average length of individuals but leads to an oversampling of shorter programs of the search space and also to the creation of very long programs compared to the average size. To compare the effects of SC and MHC on size distribution, we have performed, for both crossover operators and without mutatin, 200 experiments on a flat landscape with a population of 1000 individuals during 1000 generations. The initial size of programs was randomly chosen be-

[2] In LGP, the SC operator randomly exchanges prefixes (or suffixes) between linear sequences.

tween 1 and 50 instructions and the instruction set was defined with 10 different symbols.

We can see, in Figures 2 and 3, the expected gamma distribution obtained with SC, while with MHC, the distribution seems to converge much more slowly toward a gamma distribution. More precisely, at the last generation, when SC is used, the most numerous programs, around 7% of the population, have only 2 instructions, while for MHC, they represent 4% of the population and have 23 instructions. So MHC is less biased than SC what explains its ability to reduce bloat and at the same time its difficulty to explore efficiently the size dimension.

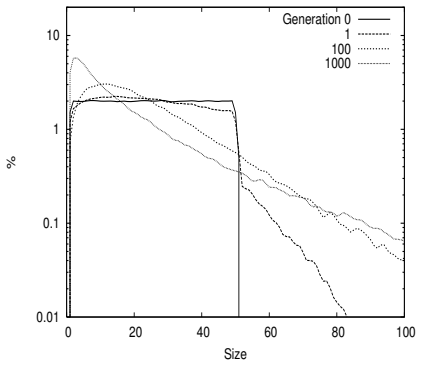

Fig. 2. Distribution of programs size using SC and without selective pressure.

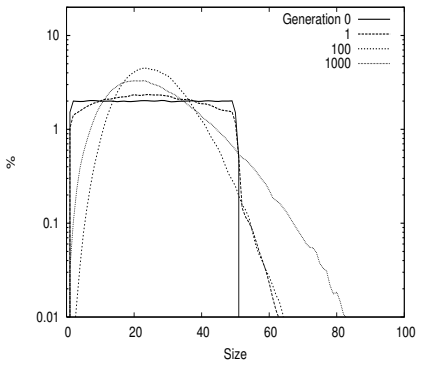

Fig. 3. Distribution of programs size using MHC and without selective pressure.

We have already mentioned that two main strategies have been investigated to fight the bloat phenomenon. With the first one, the idea is to work on fitness to modify the search space in order to eliminate too long programs. For example, with the Tarpeian Method (TM) (see [12] for pseudo-code), some "holes" are dynamically introduced in the fitness landscapes by assigning, with a given probability, a very low fitness to programs whose size is higher than average in the population. With the second strategy, the approach consist in designing unbiased operators that prevent the creation of too long programs. For example, the size-fair operators (cf. [4]) ensure that the amount of genetic material exchanged during recombination is comparable between parents and so they modify little the size of programs. In this case, the goal is to control the distribution of size of programs that undergo the selection process. We focus on two different ways to control the distribution of size with MHC.

Firstly, we propose to use the mutation operator to modify the average size of programs during evolution. In our stack-based system, mutation consists either of an insertion, a deletion or a substitution of one instruction, each operator

having its own application rate. We define an operator MHC+INS$_r$ to be MHC combined with an insertion rate of r higher than deletion and substitution rates. This unbalanced setting of the mutation rates must enable the system to increase the average size of programs and so to increase the chances of visiting areas of high performances. We note that, in [2], a similar setting was used in the context of LGP with homologous recombination to improve performances. We have plotted, in Figure 4, the size distribution obtained with MHC+INS$_{1.0}$, *ie* insertion rate equal to 1.0 and deletion and substitution rates fixed to 0.0. We see that MHC+INS$_r$ allows a translation of the size distribution reported when using MHC alone.

Secondly, we propose to use SC to modify the average size of programs during evolution. An operator MHC+SC$_p$ is defined where MHC is used to perform recombination but with p being the probability that SC will be used instead. Thus, MHC+SC$_{0.5}$ corresponds to an equally use of both MHC and SC. We note that in [4], authors have already speculated that judicious mixing of size-fair and standard operators could be the best way to encourage robust problem solving performances. We have plotted, in Figure 5, the size distribution obtained with MHC+SC$_{0.2}$. We see that MHC+SC$_p$ allows a transformation of the size distribution reported when using MHC alone.

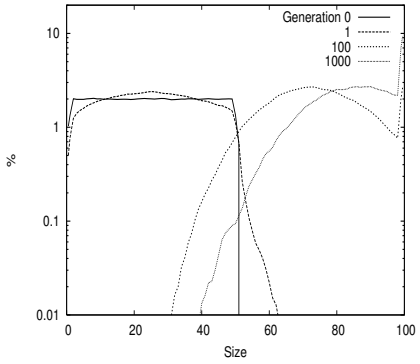

Fig. 4. Distribution of programs size using MHC+INS$_{1.0}$ and without selective pressure.

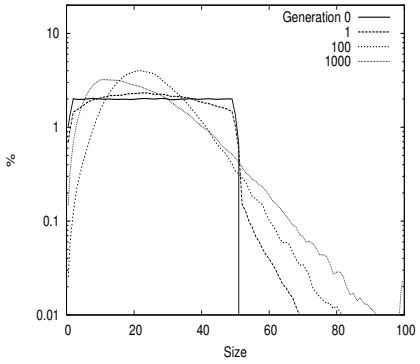

Fig. 5. Distribution of programs size using MHC+SC$_{0.2}$ and without selective pressure.

3 Experimental Results

3.1 Problem and Parameters settings

In this section, we aim to verify the ability to control the size of programs on a Symbolic Regression Problem. We choose the Poly-10 problem [12], where the

target function is the 10-variate cubic polynomial $x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$, because it was introduced as a benchmark for the study of the TM. In this study, the fitness is the classical Root Mean-Square Error. The dataset contains 50 test points and is generated by randomly assigning values to the variables x_i in the range $[-1, 1]$.

We want to compare the performance between the operators SC with TM, MHC and the two alternatives MHC+INS and MHC+SC. For the TM, we call n the parameter giving the probability that programs whose size is higher than average will receive a very bad fitness. We test different value for n varying from 0.05 to 0.9. The GP system has very distinct behavior according to the operator used, this is why to perform a fair comparison, the evolutionary parameters tuning must be extensively investigated. For each operator and for each tuning of the size control parameters (n, p and r), we perform 50 independent runs with various mutation and crossover rates. 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 500 individuals are randomly created according to a maximum creation size of 50. The instruction set contains: the four arithmetic instructions ADD, SUB, MUL, DIV, the ten variables $X_1 \ldots X_{10}$ and one stack-based GP specific instruction DUP which duplicates the top of the operand stack. The evolution, with elitism, maximum program size of 500, 16-tournament selection, and steady-state replacement, takes place over 100 generations [3]. We use a statistical unpaired, two-tailed t-test with 95% confidence to determine if results are significantly different.

3.2 Best results

In what follows, SC stands for SC without TM (n=0), MHC+INS stands for MHC+$INS_{2.0}$ and MHC+SC stands for MHC+$SC_{0.1}$. In Table 1, the best results, in terms of average fitness of the best program found, among all the parameters settings tested, are reported (crossover rate varying from 0 to 1.0 and

Table 1. Best Results.

Xover Type	Average Fitness	Average Size	Average Size without introns
SC	$0.13_{(\sigma=0.03)}$	$457.42_{(\sigma=79.07)}$	$457.14_{(\sigma=79.25)}$
MHC	$0.25_{(\sigma=0.05)}$	$92.28_{(\sigma=31.39)}$	$91.74_{(\sigma=31.45)}$
MHC+INS	$0.14_{(\sigma=0.03)}$	$247.18_{(\sigma=90.36)}$	$245.00_{(\sigma=90.75)}$
MHC+SC	$0.11_{(\sigma=0.02)}$	$419.12_{(\sigma=96.90)}$	$418.80_{(\sigma=96.82)}$

[3] In a steady state system, the generation concept is somewhat artificial and is used only for comparison with generational systems. Here, a generation corresponds to a number of replacement equal to the number of individual in the population, *ie* 500.

mutation rate from 0 to 2.0). As expected, using MHC, the system has found less fit but smaller programs than using other operators. This is unsurprising since the optimization of the "maximum initial size" parameter, needed by MHC (see Section 1.2), has not been performed. Statistical analysis of the results of SC, MHC+INS and MHC+SC shows that their average fitness does not differ significantly. Conversely, the average size of the best solution found varies greatly. The operator MHC+INS seems to give a good trade-off between fitness and size since, in this case, the average size is almost 2 times smaller than with SC. We note that an increase of the n parameter has always led to fitness degradation for the SC operator.

3.3 Application rates

In what follows, SC stands for SC without TM (n=0), MHC+INS stands for MHC+$\text{INS}_{2.0}$ and MHC+SC stands for MHC+$\text{SC}_{0.1}$. We have gone to great effort to determine the appropriate setting for each operator studied. Figure 6 depicts the average fitness of the best program found as a function of the mutation rate for the best crossover rate found. In other words, each point of the plot corresponds, for a given mutation rate, to the best result found among all crossover rates. All operators demonstrate a similar behavior according to the mutation rate, except for MHC+INS, which has obtained better performances without mutation. However, we know that it performs, by construction, at least 2.0 insertions on average per individual. The use of the mutation operator is critical but with low rates (the optimal is less than 0.4). Let us recall that a rate of 0.4 corresponds to, on average, 0.4 mutations of each type (insertion, deletion and substitution) per individual, so to a little more than one change per individual.

In Figure 7, we have plotted the best results found according to the crossover rate for a mutation rate set to 0.2. We see that SC obtains its best result with a small crossover rate but that its performances tend to worsen when too many recombinations are performed. On the other hand, the performances of MHC, MHC+INS and MHC+SC operators do not vary so much according to the crossover rate, but with a small tendency to increase for high rates. Figure 8 represents the average size of the best program found as a function of the crossover rate for a mutation rate of 0.2. We see that the size of the programs found using SC, MHC and MHC+INS does not depend on the crossover rate. More precisely, for SC, the size is limited by the "maximum allowed size" parameter, here 500 instructions. Whereas for MHC, the "maximum creation size", here 50 instructions, is the major parameter influencing size (see also Figure 9). Finally for MHC+INS, we see that the insertion of instructions, here 2.0 on average, in each individual of the population leads to an increase of more than 100 instructions compared to MHC. It is obvious that in the case of MHC+INS, when no recombination is performed, size control does not work (around 400 instructions) since there is nothing to compensate the unbalanced mutation setting. Conversely, the MHC+SC operator finds programs of different sizes according to the crossover rate. This means that the size of programs does not depend

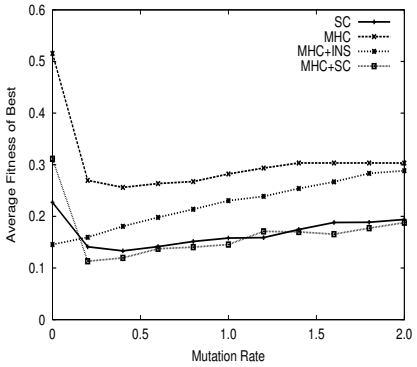

Fig. 6. Average fitness of best as a function of the mutation rate on Poly-10.

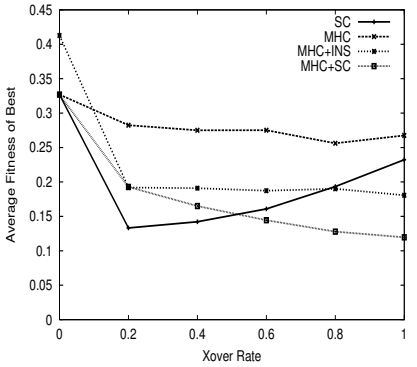

Fig. 7. Average fitness of best as a function of the crossover rate on Poly-10.

only on the proportion of MHC and SC (the parameter p) but rather on the number of SC recombinations performed per generation, which increases with the crossover rate.

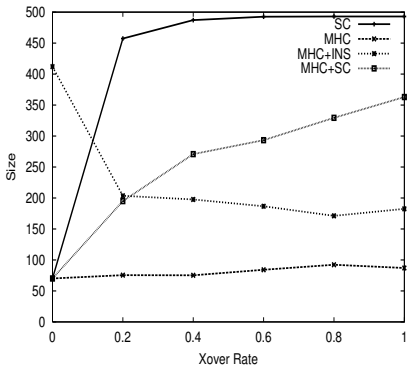

Fig. 8. Average size of best as a function of the crossover rate on Poly-10.

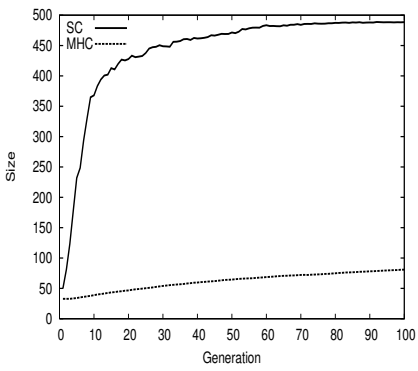

Fig. 9. Evolution of the average size of best on Poly-10.

3.4 Fitness *vs* Size Trade-off

In order to visualize the fitness *vs* size trade-off, we have plotted, in Figure 10 a scatter plot of the average fitness and the average size of the best solutions found. Each point corresponds to one of the setting of the parameters (of both mutation

and crossover rates) tested in this study. Lines connecting points depict the Pareto frontiers. We can see that the trade-off between size and fitness differs for the four operators. We see that when SC with n=0 or MHC are used, variations in the size dimension are very small. On the other hand, frontiers for SC with TM and for both alternatives of MHC cover larger ranges in the fitness vs size space. However, excepted for SC with n=0.9 that gives the shorter programs, the size control methods using MHC report better trade-off than SC with TM. Let us recall that results presented here do not correspond to a multi-objective approach since our goal was not to minimize, in words of Pareto optimality, both size and fitness criteria. We next investigate further the influence of the parameters r and

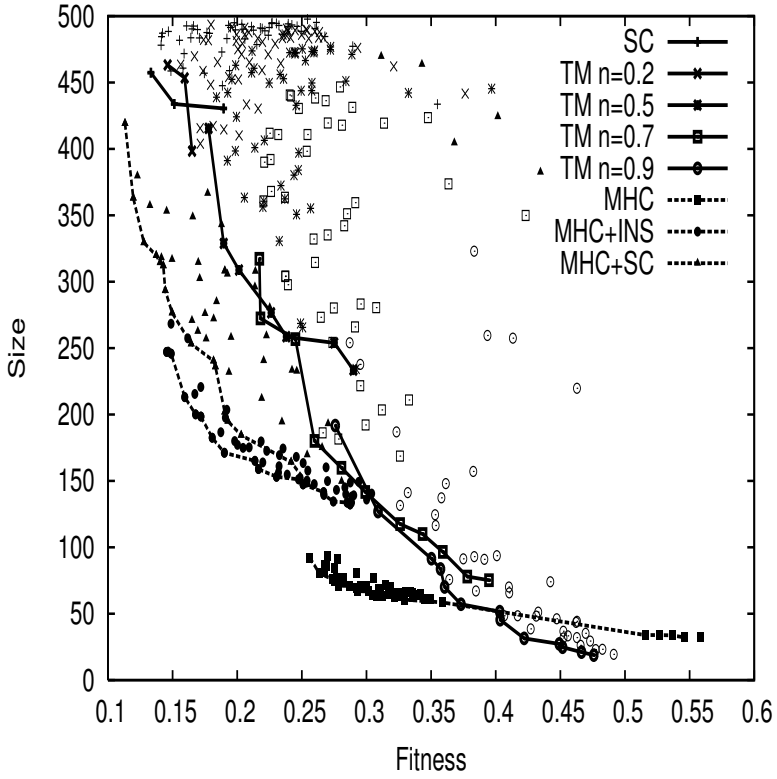

Fig. 10. Fitness vs Size Trade-off on Poly-10. Lines connecting points correspond to Pareto frontiers.

p on fitness and size for both MHC+INS$_r$ and MHC+SC$_p$ operators. We have performed some specific experiments with a mutation rate of 0.2 and a crossover of 0.80. Figures 11 and 12 show the variations of, respectively, the average fitness

and size of the best program found as a function of r for MHC+INS$_r$. We see that the insertion of instructions, controlled by r, always leads to an improvement in fitness but that for r greater than 2.0, no gains can be obtained. The average size is strongly correlated to the parameter r and all the allowed sizes in the search space can be reached. Compared to the performances of MHC, using MHC+INS$_{2.0}$, we obtain programs around two times more fitter but also two times bigger (see Table 1 above).

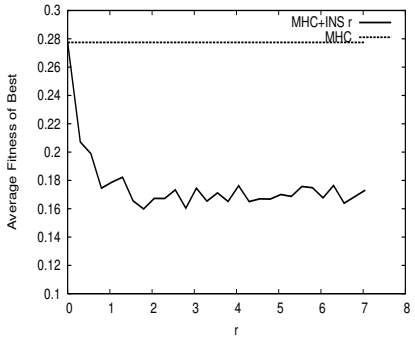

Fig. 11. Average fitness of best as a function of r on Poly-10 with MHC+INS$_r$.

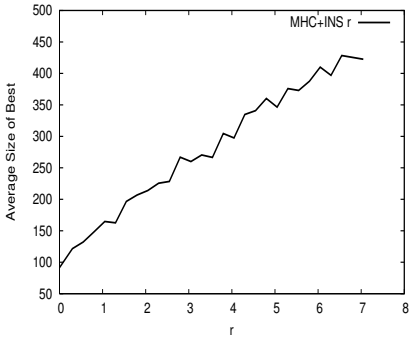

Fig. 12. Average size of best as a function of r on Poly-10 with MHC+INS$_r$.

Figures 13 and 14 show the variations of, respectively, the average fitness and size of the best program found as a function of p for MHC+SC$_p$. We see an improvement of the fitness, compared to MHC and SC, for all the values of p. This means that the combination of both MHC and SC performed better than when MHC and SC are used separately. However the size of the programs increases quickly with parameter p and seems to reach the maximum size when p is greater than 0.5. Moreover, we note a minimum of the fitness curve, when p is equal to 0.2. This implies that the p parameter must be carefully fixed. In the size control context, we can define, for the Poly-10 problem, a "region of interest" of the MHC+SC$_p$ operator for p in the range [0,0.2].

4 Conclusion and Perspectives

Contrary to SC, MHC do not induce any specific biases in the distribution of sizes and so an accurate control of the distribution during evolution is possible and have to be investigated.

In this paper, two methods for controlling the distribution with MHC are introduced and tested. The first one, called MHC+INS$_r$, where MHC works in

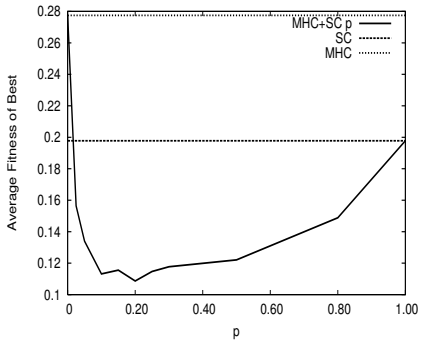

Fig. 13. Average fitness of best as a function of p on Poly-10 with MHC+SC$_p$.

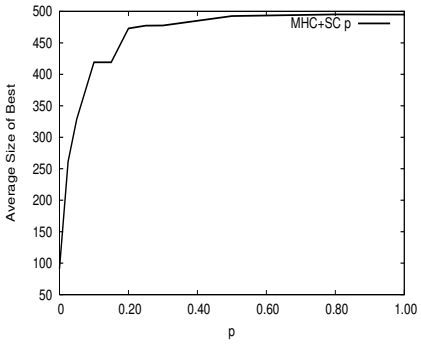

Fig. 14. Average size of best as a function of p on Poly-10 with MHC+SC$_p$.

conjunction with the mutation operator, directly modifies the number of genes in the population, *ie* the total amount of available instructions. In the second one, called MHC+SC$_p$, the MHC works in conjunction with SC to allow the creation of much bigger programs than the average size in the population. As expected, we note a significant increase in the average size and in the average fitness of the solution found. This reinforces our first assumption: to be efficient with MHC, the size of programs has to be explored as a new dimension of the search space. Nevertheless, the two methods presented here are static and so require a specific tuning that may depend on the problem addressed. Hopefully, the first steps in the study of size control methods, and more generally of MHC behavior, allow us to believe that dynamic control of the size is possible, according to some exogenous or endogenous properties.

MHC understanding, thanks to experimental results, is improved. For various benchmarks, the performance of this operator is equivalent but with an accurate management of the size. Future work should consist in a study of much more complex problems and then to real-world applications where an uncontrolled growth of the size of programs is a strong limitation for GP. For this purpose, the use and design of new dynamic methods for size control with MHC, taking into account some exogenous or endogenous additional informations, will undoubtedly be required.

References

1. L. Altenberg. The evolution of evolvability in genetic programming. In *Advances in Genetic Programming*. MIT Press, 1994.
2. M. Brameier and W. Bhanzhaf. Explicit control of diversity and effective variation distance in linear genetic programming. In *Genetic Programming, Proceedings of*

the 5th European Conference, EuroGP 2002, volume 2278 of *LNCS*, pages 37–49, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.
3. W. S. Bruce. The lawnmower problem revisited: Stack-based genetic programming and automatically defined functions. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 52–57, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.
4. R. Crawford-Marks and L. Spector. Size control via size fair genetic operators in the PushGP genetic programming system. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 733–739, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
5. E. D. de Jong, R. A. Watson, and J. B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 11–18, San Francisco, 2001. Morgan Kaufmann.
6. M. Defoin Platel, M. Clergue, and P. Collard. Homolgy gives size control in genetic porgramming. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 281–288. IEEE Press, 2003.
7. M. Defoin Platel, M. Clergue, and P. Collard. Maximum homologous crossover for linear genetic programming. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 194–203, Essex, 14-16 April 2003. Springer-Verlag.
8. W. B. Langdon and R. Poli. Fitness causes bloat. In *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag London, 23-27 1997.
9. S. Luke. Code growth is not caused by introns. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 228–235, Las Vegas, Nevada, USA, 8 2000.
10. S. Luke. Modification point depth and genome growth in genetic programming. *Evol. Comput.*, 11(1):67–106, 2003.
11. T. Perkis. Stack-based genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, Florida, USA, 27-29 1994. IEEE Press.
12. R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 204–214, Essex, 14-16 April 2003. Springer-Verlag.
13. R. Poli and W. B. Langdon. Genetic programming with one-point crossover. In *Soft Computing in Engineering Design and Manufacturing*, pages 180–189. Springer-Verlag London, 23-27 June 1997.
14. R. Poli and N. F. McPhee. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038, pages 126–142. Springer-Verlag, 18-20 2001.
15. J. E. Rowe and N. F. McPhee. The effects of crossover and mutation operators on variable length linear structures. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 535–542, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.
16. T. Soule and J.A. Foster. An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 3(1):283–309, 2002.
17. R.E. Keller W. Banzhaf, P. Nordin and F.D. Francone. *Genetic Programming - An Introduction.* Morgan Kaufmann, 1998.