Evolutionary Self-Adaptive Semantics

Rafael Inhasz, Julio M. Stern

Institute of Mathematics and Statistics, São Paulo University
rafael.inhasz@yahoo.com.br jstern@ime.usp.br

Abstract.
We present SASC, Self-Adaptive Semantic Crossover, a new class of crossover operators for genetic programming. SASC operators are designed to induce the emergence and then preserve good building-blocks, using meta-control techniques based on semantic compatibility measures. SASC performance is tested in a case study concerning the replication of investment funds.

Keywords: Genetic programming, crossover, building blocks, emergent semantics, meta-control.

1. INTRODUCTION

Genetic Programming (GP) are evolutionary algorithms that work on populations, whose individuals represent possible (viable) solutions to the optimization problem, see Banzhaf et al.(1998) and Koza el al.(1992,94). The solution functions, code or programs defining an individual are its *genotype*, while the image, graph or output of these functions are the individual's *phenotype*. An *adaptation*, *cost* or *fitness* function, computed from an individual's phenotype, is the optimization problem's objective function.

GP are meta-heuristics based on some key functions and operators inspired on evolution theories for biological species. *Reproduction* operators generate new individuals, the *children*, from existing ones, their *parent(s)*, hence expanding the population. *Mutation* operators act on single individuals, for asexual reproduction, while *crossover* operators act on pairs of individuals, for sexual reproduction. A mutation operation generates a random change in the parent's code. This change is usually small, but may have important consequences for the individual fitness, often bad, but sometimes good. A crossover operation generates new children by swapping portions of their parents' codes at randomly selected *recombination points*.

Reproduction operators are random operators. However, they only introduce a limited amount of entropy (noise or disorder) in the process, making it possible for children to *inherit* many characteristics coded by their parents' genotype. GP starts from an initial population that may be randomly generated. The population then evolves according to the random reproduction and selection stochastic processes. The entropy introduced at reproduction allows for creative innovation, while the selection processes induce learning constraints. Under appropriate conditions, after many generations (near) optimal individuals are likely to emerge in the population.

The *schemata theorem*, arguably the most characteristic result of GP theory, shows that, under appropriate conditions, the emerging optimal solutions naturally exhibit a hierarchical modular organization. Such modules are known as *genes*, *schemata* or *building blocks*, see Holland (1975), Langdon and Poli (2002), Reeves (1993), Simon

(1996) and Stern (2008b). In light of the Schemata theorem, it is easy to understand that efficient crossover operators must be compatible with, preserve, favor, or even induce the emerging modular structure. More efficient operators are less likely to break down existing building blocks during reproduction, an unfortunate event known in the literature as *destructive crossover*.

This paper presents a new crossover operator, named SASC or *Self-Adaptive Semantic Crossover*. SASC is based on *meta-control* techniques designed to guide the random selection of recombination points by a measure of *semantic compatibility* between the portions of code being swapped. It is important to realize that SASC's meta-control system is not hard-wired or pre-defined. On the contrary, it is an emerging feature, co-evolving with the population. The meta-control system is based on the history of each individual in the population. However, the required historical information, accumulated during the individual's evolutionary line, is very limited. Hence, its implementation only generates a minor computational overhead.

2. GENETIC PROGRAMMING IN FUNCTIONAL TREES

In this and the following sections, we deal with GP in the context of functional trees. In this setting, the objective is to find the correct specification, the best functional form, or just a good emulation of a complex *target* function. The only information available about the target function is an input-output data-bank. An individual in the population is represented as a tree, with atoms at the leaves representing constants or input variables, and primitive operators at internal nodes. The root node output, at the top of the tree, expresses the individual's phenotype. Atoms and primitive operators are taken from finite sets, $A = \{a_1, a_2, \ldots\}$ and $OP = \{op_1, op_2, \ldots op_p\}$. Each operator, op_k, takes a specific number of arguments, $r(k)$, known as the arity of op_k.

Figure 1 shows four individuals in the population of a GP trying to emulate the target function, $f(w,y,z) = y^2 + w^{z/y}$, from the primitive set of expanded arithmetic operators, $OP = \{+, -, \times, /, \wedge\}$. Inputs at the leaves are represented in a square, and operators at internal nodes or at the root are represented in a circle. Figure 1 also shows a crossover, having the first two individuals as parents and the last two as children. The recombination points in the parent trees are highlighted. Notice that the first parent contains the component, partial solution or building block for the first term in the target function, y^2, while the second parent contains the building block for the second term, $w^{z/y}$. Since none of these interesting building blocks are preserved in the children, we call this a destructive crossover. A child inherits its root node, and hence usually most of its code, from the parent we call its *mother*, while from its *father* the child receives a, usually smaller, sub-tree. Hence, in this example, parent 1 and 2 are, respectively, mother and father of child 1, and father and mother of child 2.

Angeline (1996) proposed the SSAC - *Selective Self-Adaptive Crossover* - in order to make destructive crossovers less likely. Standard crossover selects recombination points in a parent tree with uniform distribution. In SSAC like crossovers, each node, $n(i)$, stores a meta-control variable, ρ_i, a real number bounded to the normalization constraint: $0 \leq \rho_{min} \leq \rho_i \leq \rho_{max}$. The probability of selecting node $n(i)$ for recombination is proportional to ρ_i. That is, the probability of choosing node $n(i)$ as the recombination

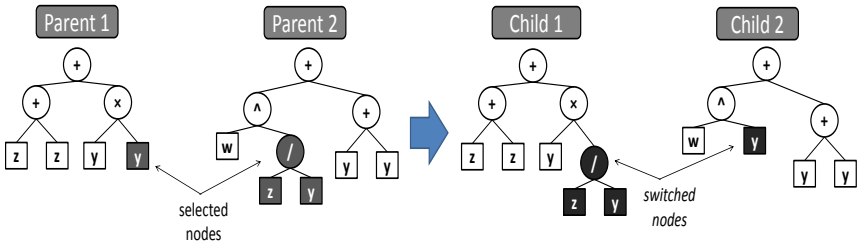

FIGURE 1. Figure 1: Example of destructive crossover

point in that tree is $p_i = \rho_i / \sum_j \rho_j$.

After a crossover, nodes at the children carry along the meta-control variables they had at the parents, and afterwards suffer the effect of random noise. For example, the meta-control variable in node $n(i)$ can be updated as $\rho'_i = (1 + \mu_i + \sigma_i \varepsilon)\rho_i$, where ε is the standard Normal random variable, μ_i is a zero or positive drift, and σ_i is a positive scale factor. All ρ_i are initialized at the minimum value, ρ_{min}, and allowed to move inside the normalization bounds. For details on Angeline's original implementation, see Angelinr (1996).

The intuition behind SSAC is that survivors in the GP competition process are well adapted individuals, containing good building blocks. Moreover, successful breeders must be able to give these building blocks intact to their children. At these breeders, large meta-control variables should mark plausible building blocks, indicating good recombination points to be used (again) in the future. Genotype codes and meta-control variables should both co-evolve, facilitating the emergence, marking, and preservation of good building blocks.

Before ending this section we make some additional comments about the schemata theorem. As already mentioned in the introduction, it is in the light of the schemata theorem that we can understand why efficient crossover operators must be compatible with, preserve, favor, or even induce the emerging modular structure. However, Holland's original theorem was stated for a very particular case, namely, genetic algorithms using string coded programs. *Schemata theories* extend this fundamental result to genetic programming using functional trees, see Langdon and Poli (2002). Hence, we must rely on Rosca, Poli and Langdon's results to keep our work on well founded theoretical ground.

3. THE SELF-ADAPTIVE SEMANTIC CROSSOVER

SASC descends from Angeline's SSAC and SAMC operators, but it also incorporates information concerning the sub-trees rooted at the nodes in possible recombination points. The first information used for this purpose is captured through the notion of similarity. (Sub)Trees A and B are phenotypically similar if their output, computed at the records available on the data bank, agree within a specified tolerance.

We assume that two parents, father A and mother B, have been selected for crossover according to the mating distributions used at the GP. SASC starts by using a first heuristic procedure to define new meta-control variables, δ_i, at the nodes, $n(i)$, of the

father, A. Let $A(i)$ be the sub-tree of A rooted at $n(i)$. For each sub-tree, $A(i)$, the procedure searches the mother, B, for sub-trees, $B(j)$, that are similar to and also either the same size or shorter than $A(i)$. If such a short similar sub-tree is found, $\delta_i = \rho_{min}$. Otherwise, $\delta_i = \rho_i$. Finally, the recombination point at the father is randomly selected with probabilities $p_i = \delta_i / \sum_j \delta_j$. The intuition behind the first heuristic procedure is to stimulate innovation, that is, to only chose recombination points at the father that, by the crossover operation, are able to contribute with an innovative component, $A(i)$, that is not already present in the mother or, at least, to contribute with a similar component that is more efficiently coded.

After the recombination point at the father, $n(i)$ - root of sub-tree $A(i)$, has been chosen, a second heuristic procedure selects the recombination point at the mother, $m(j)$ - root of sub-tree $B(j)$. Again, new meta-control variables, λ_j are defined for the nodes $m(j)$, followed by a random selection with probabilities $p_j = \lambda_i / \sum_j \lambda_j$. The idea behind this second heuristic procedure is to stimulate the crossover to exchange sub-trees, $A(i)$ and $B(j)$, with analogous meanings, compatible semantics, similar interpretations, etc. This heuristic procedure draws inspiration from biology, where analogy is defined as compatibility in function but not necessarily in structure or evolutionary origin.

The formal expression used to evaluate the meta-control variables at the second heuristic procedure is: $\lambda_j = w_0 + \sum_{d=1}^{D} w_d C_k(A(i), B(j))$. The index d spans D semantic dimensions or factors. The positive weights, w_d, add to one, and the semantic compatibility measures, C_k, are normalized in the interval $[0,1]$.

The functional form of the compatibility measures, $C_k(\ )$, are completely dependent on insights and interpretations for the actual problem being solved. In the case of the arithmetic functional tree presented at this section, the analogy between two sub-trees could be established, for example, simply by the fraction of input variables they share in common. In this case, blocks coding y^2 e $2y$ would have compatibility measure equal to 1, while the blocks coding y^2 and $w^{z/y}$ would have compatibility measure equal to $1/3$.

After a SASC crossover, the children's nodes carry along the meta-control variables, ρ_i, they had at the parents, and are afterwards updated by a random perturbation. We used a standard Normal multiplicative noise with drift μ_i and scale factor σ_i, that is, $\rho'_i = (1 + \mu_i + \sigma_i \varepsilon)\rho_i$. At practical implementations we always used a positive drift at the recombination points, and a null drifts elsewhere. Sometimes we also used scale factors, σ_i, that decrease with the height of node $n(i)$. For instance, take σ_i inversely proportional to the depth of sub-tree $A(i)$. Using larger scale factors at lower nodes can help to induce the emergence of smaller building-blocks, that are more efficiently coded, and less prone to destructive crossover.

4. IMPLEMENTATION AND CASE STUDY

Our implementation of SASC methods is based on ECJ, an open-source evolutionary computing system written in Java. ECJ is developed at George Mason University's ECLab Evolutionary Computation Laboratory. ECJ maintains a well organized object-oriented design. Its powerful classes and methods proved to be very flexible, and could be easily extended to our purposes. The SASC package, developed by the first author, extends some ECJ classes in order to easily implement the methods under discussion. Most

of the new code is concentrated at the class *SASCNode*, used to represent functional trees evolving by SASC GP. This class also includes abstract methods that facilitate the implementation of semantic compatibility measures, specified at sub-classes implemented for each specific problem.

Finally, we should mention that ECJ supports distributed computing, specifying the desired number of parallel threads as a parameter to be set according to the available resources offered by the hardware and operating system. This feature was especially useful for multi-population scenarios, to be described in the next section, where SASC GP had an excellent performance.

SASC operator was compared to standard crossover operators at a test case problem concerning the replication of an hypothetical investment fund. Although hypothetical, this problem has strong similarities with real problems regarding the construction of synthetic portfolios faced by the first author in his professional activities. Portfolios of this kind are typical of correlation trade, since its return statistics are sensitive to the correlation matrix for the returns of various components in a basket. Such portfolios can be easily synthesized using readily available exotic derivatives like rainbow options, that is, calls or puts on the best or worst of several underlying assets.

Lemon, the hypothetic fund, is based on stocks negotiated at *BM&F-Bovespa - São Paulo Securities, Commodities and Futures Exchange*. Lemon's daily log-return, r_t, is given by the log-return average of four components, r_t^k, corresponding to key economic sectors. These are, using standard *BM&F-Bovespa* equity codes: $r^1 = \min$(BBDC4, PETR4, BBAS3), $r^2 = \min$(LAME4, LREN3, NETC4), $r^3 = \max$(TNLP4, TCLS4, VIVO4) and $r^4 = \max$(CYRE3, ALLL11, GFSA4). These components represent four key economic sectors: Telecommunications, construction and transports, finance and cyclic consumption.

An asset manager wants to synthesize a second fund, Lime, with the objective of tracking fund Lemon. However, only the daily share values of fund Lemon are available, not its operational rules. Of course, GP was the method chosen to find the best specification of the synthetic portfolio Lime. The atoms for this problem are daily log-returns, from 04-Nov-2008 to 01-Apr-2009, of the 63 most liquid stocks negotiated at *BM&F-Bovespa*. These include all the stocks used to specify fund Lemon. The primitive operators are $\{\max, \min, \text{mean}\}$, the maximum, minimum and mean value of two real numbers.

The fitness function for this problem is the mean squared error between the synthetic and the target log-returns, plus a regularization term adding, for each node, $n(i)$, a penalty $\pi(i)$. For the application at hand, we used $\pi(i) = c_{h(i)} 2^{h(i)-1}$, where $h(i)$ is the height of node $n(i)$. For the example at hand, we used $c_{h(i)} = 1$ at the root node and zero otherwise. The purpose of regularization term is to avoid needless complexity and over-fitting in the final model, see Cherkaasky and Mulier (1998).

In the GP experiments, we used two distinct population scenarios. Scenario 1: One population of 300 individuals evolving over 700 generations, Scenario 2: 8 populations of 300 individuals each, that first evolve in isolation over 400 generations and are then allowed to merge and evolve for 100 generations more. In both scenarios the GP is allowed to warm-up using the standard crossover, 200 generations for scenario 1 and 100 for scenario 2, and then switch to (or not) to SASC crossover. SASC's semantic

compatibility function is the Boolean indicator of having at least one atom in common.

The actual GP implementation uses a dual tree representation for each individual in the population, as suggested in Angeline (1996) original paper. The first tree only stores the genotype used to code the function expressed by the individual's phenotype. Meanwhile, the second tree only stores meta-control variables.

GP meta-parameters were set as follows: mutation rate was set at 5%, using a 3-round tournament selection process. Crossover rate was set at 95%, using a 7-round high pressure / 3-round low pressure combination of father / mother selection, see Stern (1998b). $\rho_{min} = 0.001$, $\rho_{max} = 0.999$, $w_0 = 0.01$, $w_1 = 0.99$, $\sigma_i = 0.4$ for $h(i) = 2$ and approximately inversely proportional to the node height for $h(i) > 2$. Further details about the algorithm fine tuning can be seen at the source code documentation, available from the first author.

Figure 2 compares the GP results using standard and SASC crossover operators. The use of Angeline's original SSAC instead of the standard crossover operator had only a minor impact in GP performance, and is not shown in the figure. This figure displays 95% confidence intervals for the mean square error of the best solution found over 50 independent GP runs.

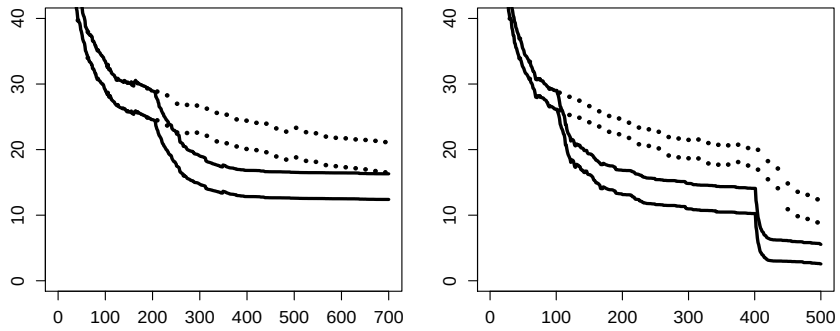

FIGURE 2. Figure 2: Confidence interval for best solution MSE by generation.

Figure 3 shows the best empirical solution found by SASC GP. The figure also highlights the building blocks encapsulated by meta-control variables larger than a critical threshold. This solution replicates very well the target fund. Notice that each of the highlighted building blocks corresponds to one of the key economic sectors used to define the operation rules of fund Lemon.

Each best solution found at a batch of 50 SASC GP experiments under scenarios 1 and 2 was categorized according to the number of key economic sectors represented by a constituent building block. Table 1 displays the average mean square error of each category. This table shows that better adjusted functional trees have more of the four key economic sectors present as a building block. This conclusion may be obvious to someone knowing the operating rules of Lemon, the original target fund. However, it is remarkable that the best solutions offered by SASC GP for the replication fund Lime, synthesized only from input-output data, are able to capture so well the logic and semantics of fund Lemon.

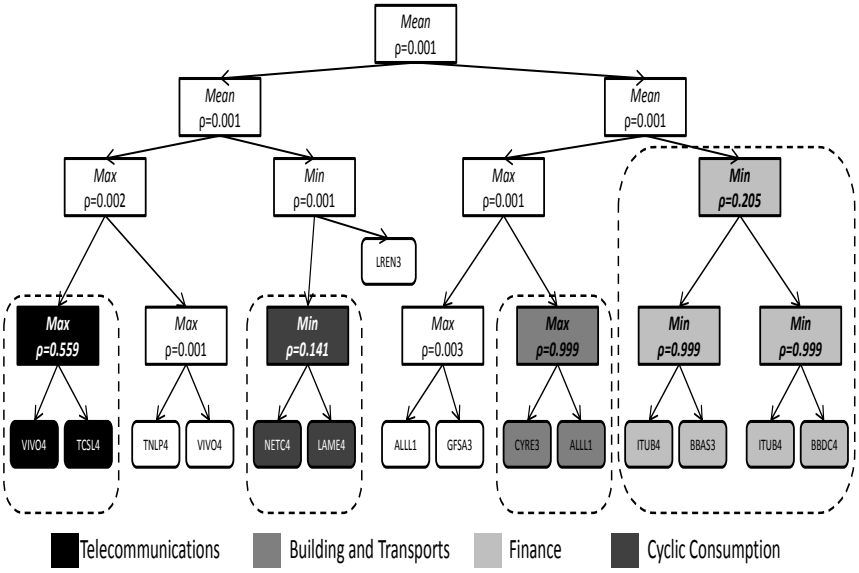

FIGURE 3. Figure 3: Emerging building-blocks in near-optimal solution

5. CONCLUSIONS AND FINAL REMARKS

From Figure 2, we can conclude that, at least for the test case at hand, GP has a much better performance when using SASC than the standard crossover operator. At scenario 2 the best empirical solution, shown at Figure 3, is found repeatedly. At scenario 1, SASC not only achieves better results, but also seems to greatly accelerate the finding of good solutions. These effects are even stronger at scenario 2, where a second acceleration effect is clear just after the populations merge. At this final stage, one can observe that the best solution are formed purging spurious building blocks and combining good building blocks that had emerged at the previously isolated populations. It is as if SASC were able to isolate, identify, and collect good building blocks.

The explanatory power of the emergent building blocks, that is, on one hand, how well they capture the semantics of the system under study and, on the other hand, how much they contribute to its prediction accuracy, is made even clearer by Table 1. Accordingly, Figure 3 suggests that SASC GP can also provide an implicit method of semantic analysis. That is, at least in our case study, the internal operational logic and the semantics of the target system is adequately represented by the building blocks of

TABLE 1. Number of key sectors represented by building blocks

Category	Scenario 1	MSE	Scenario 2	MSE
One key sector	14%	12.3	10%	8.9
Two key sectors	16%	8.1	30%	1.9
Three key sectors	8%	9.3	38%	1.4
Four key sectors	0%	-	4%	0.1
Other (spurious) blocks	62%	21.7	18%	10.2

the best solutions synthesized by SASC GP. Nevetheless, it is important to keep in mind that these logical and semantic relations were not externally imposed or driven, but are truly emergent properties co-evolving with the GP solutions.

Future Research: In future research we plan to investigate techniques of self-adaptive meta-control using abstract type node labels as auxiliary control variables. Transformation rules for label mutation and label compatibility rules for permissible recombination points should be able to induce building block formation and encapsulation, and also be able to foster emergent semantic interpretations, even in problems lacking natural heuristics for explicit semantic compatibility measures.

Acknowledgements: The authors are grateful for the support of *IME-USP,* The Institute of Mathematics and Statistics of the University of São Paulo, *FAPESP,* Fundo de Amparo à Pesquisa do Estado de São Paulo, *CNPq,* The Brazilian National Research Council, and *BM&F-Bovespa,* The São Paulo Securities, Commodities and Futures Exchange. The authors are also grateful for the helpful comments of Marcelo Lauretto and an anonymous referee.

REFERENCES

P.Angeline (1996). Two Self-Adaptive Crossover Operators for Genetic Programming. p.89-109 in: P.J.Angeline, K.E.Kinnear. *Advances in Genetic Programming. Vol.2.*. MIT.

W.Banzhaf, E.D.Francone, R.E.Keller, P.Nordin, (1998). *Genetic Programming, an Introduction.* San Francisco:Morgan Kaufmann.

V.Cherkaasky, F.Mulier (1998). *Learning from Data.* NY: Wiley.

J.H.Holland (1975). *Adaptation in Natural and Artificial Systems.* Univ.of Michigan Press.

H.Iba, T.Sato (1992). Meta-Level Strategy for Genetic Algorithms Based on Structured Representation. p.548-554 in *Proc.of the Second Pacific Rim Intern.Conf.on Artificial Intelligence.*

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

J.R.Koza (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge: MIT.

W.B.Langdon, R.Poli (2002). Foundations of Genetic Programming. Springer.

M.Lauretto, F.Nakano, C.A.B.Pereira, J.M.Stern (2009). Hierarchical Forecasting with Polynomial Nets. p.305-315 in Nakamatsu et al. (2009).

K.Nakamatsu, G.Phillips-Wren, L.C.Jain, R.J.Howlett (2009). *New Advances in Intelligent Decision Technologies.* Heidelberg: Springer.

C.R.Reeves (1993). *Modern Heuristics for Combinatorial Problems.* Blackwell Scientific.

H.A.Simon (1996). *The Sciences of the Artificial*. MIT Press.

J.M.Stern (2008a) *Cognitive Constructivism and the Epistemic Significance of Sharp Statistical Hypotheses.* Tutorial book for the 28th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering, Boracéia, São Paulo, Brazil

J.M.Stern (2008b). Decoupling, Sparsity, Randomization, and Objective Bayesian Inference. *Cybernetics and Human Knowing,* 15, 2, 49-68.

J.M.Stern, E.C.Colla (2009). Factorization of Bayesian Networks. p.275-294 in Nakamatsu et al. (2009).