Adaptable Representation in GP

Cezary Z Janikow
Department of Math and CS
UMSL
St Louis, MO 63121
janikow@umsl.edu

ABSTRACT
Genetic Programming uses trees to represent chromosomes. The user defines the representation space by defining the set of functions and terminals to label the nodes in the trees. The sufficiency principle requires that the set be sufficient to label the desired solution trees, often forcing the user to enlarge the set, thus also enlarging the search space. Structure-preserving crossover, STGP, CGP, and CFG-based GP give the user the power to reduce the space by specifying rules for valid tree construction, based on types, syntax, and heuristics. These rules in effect change the representation. However, in general the user may not be aware of the best representation, including heuristics, to solve a particular problem. Last year, ACGP methodology was introduced for extracting local problem-specific heuristics, that is for learning a local model of the problem domain. ACGP discovers representation, in the space of probabilistic representations, one that improves the search itself and that provides the user with heuristics about the domain. This paper discusses and illustrates the probabilistic representation.

Categories and Subject Descriptors
I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

I.2.6 [**Artificial Intelligence**]: Learning

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms
Design, Experimentation.

Keywords
Genetic Programming, Heuristics, Representation

1. INTRODUCTION
Evolutionary computation techniques solve a problem by utilizing a population of solutions evolving under limited resources. The solution chromosomes are evaluated by a problem-specific user-defined evaluation method. They compete for survival based on this fitness, and they undergo simulated evolution by means of crossover and mutation operators.

Genetic Programming (GP), introduced by Koza [12] differs from other evolutionary methods by mainly using trees to represent potential solutions. Trees provide rich representation that is sufficient to represent computer programs, analytical functions, variable length structures, even computer hardware [1][12]. The user defines the representation space by defining the set of functions and terminals labeling the nodes of the trees. One of the foremost principles is that of *sufficiency* [12], which states that the function and terminal sets must be sufficient to express a solution to a problem. The reason is obvious: every solution will be in the form of a tree, labeled only with the user-defined elements. Sufficiency usually forces the user to enlarge the sets of functions and terminals, to ensure the inclusion of the necessary atomic labels. This unfortunately bloats the representation and increases the search space.

GP representation does not place any preferences on any specific tree that can be instantiated. Selection forces the search to diverge into better payoff regions. Then, GP searches the space in genotype neighborhoods of these regions. However, some regions are better than others. McPhee with Hopper [14], and Burke [3] analyzed the effect of the root node selection on GP, which in fact amounts to selection of specific subspace. Hall and Soule [5] have studied the phenomenon more extensively and have concluded that the choice of the root node has a very significant impact on the solutions generated, and that fixing the root node properly amounts to limiting the search space needed to be searched. Daida has shown that later GP generations introduce little variation into the structure of the generated trees [4], indicating that these later generations search a smaller subspace of the search space. Langdon has shown that GP typically searches only a well defined region of the potential search space [13]. Thus, GP itself selects better genotype regions, or can be forced to do so. However, very little research has been devoted to such design issues beyond the root node. Heuristics can be used for the purpose. Global heuristics, such as the choice of the root, provide for coarse level subspace preference, while local heuristics provide to finer preferences. There are two kinds of heuristics that can be available in a problem domain: *weak* and *strong*. Strong heuristics are in fact constraints, that is they impose specific rules. Weak constraints are, on the other hand, stochastic preferences. Using such heuristics amounts to changing the otherwise uniform GP representation.

Processing arbitrary heuristics is a very complex issue. Some methods have been developed over the years. *Structure-preserving* crossover was introduced as the first attempt [12], with

GECCO'05, June 25-29, 2005, Washington, DC, USA.

the primary initial intention to preserve structural constraints imposed by automatic modules *ADF*s. It was capable of processing very limited forms of strong constraints. In the nineties, three independent more generic methodologies have been developed to allow problem-independent constraints on the tree construction. *Strongly Typed* GP (STGP) [15] processes strong constraints based on data types, along with overloaded functions. *Context-free* based GP (CFG-GP) [17] allows processing strong context-free constraints, that is syntax rules. Some more recent research allows processing and even generating weak constraints [2].

Constrained GP (CGP) allows the same, that is processing both strong and weak constraints, on arbitrary trees rather than syntax trees, along with overloaded functions [6][11]. In 2003, a new methodology, *Adaptable* CGP (ACGP) was introduced, which allows automatic extraction of problem-specific heuristics. At present ACGP works only with local-level heuristics, so called *first-order* [8][9]. However, even these limited heuristics have been shown to improve GP search properties quite substantially [8][9][10].

In section 2 we discuss the GP representation, and in section 3 we discuss how weak and strong constraints affect it. In section 4 we present some illustrations of ACGP heuristics and their effect on GP problem solving.

2. GP REPRESENTATION AND SEARCH SPACE

The search space of GP is a 2-dimensional space, theoretically unbounded but practically bounded in both dimensions. First, there is the space of unlabeled (uninstantiated) tree structures. Placing the structures into equivalence classes by size, and ordering them by the number of nodes, gives the first unbounded dimension. However, GP usually imposes some restriction on the number of nodes (directly, or indirectly by depth limits). This creates a bound. Second, a particular unlabeled tree structure can be instantiated in a number of ways. This is again unbounded if terminals include ephemeral constants. However, given computer representation, even this dimension is bounded. This is illustrated in figure 1. The bounded area is the search space, spanned by the given representation (functions and terminals).

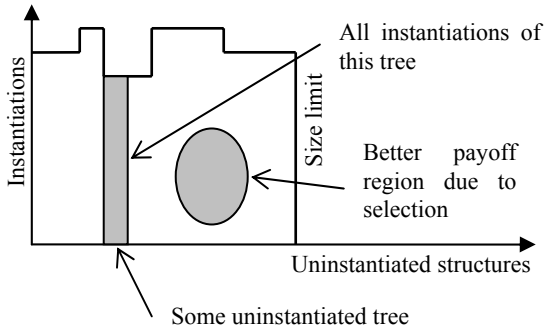

Figure 1. GP 2-D search space spanned by given functions and terminals.

In GAs, where the choice of genetic operators is much richer, the question of representation cannot be discussed without reference to operators, because two representations, given some specific properties, can produce equivalent search given some relationship between the operators [7]. However, most GP systems use the same operators of reproduction, crossover, and mutation/uniform mutation. Therefore, the question of representation becomes much more important.

The representation space spanned as in figure 1 is uniform. That is, a given node can be labeled with any arity-consistent function or terminal. This is not always desirable, beneficial, or even valid. During evolution GP "learns" to adjust the uniform space, by searching more profitable regions due to selection. However, mutation always labels its subtree using the same label set as used in the initial random population, thus disregards this information completely. Crossover does better as it only selects subtrees from the more profitable regions, but because it takes the subtrees out of context it often leads to subspaces already disregarded by selection. Thus, both operators can take a tree from the better payoff region and produce a tree outside the region.

Figure 2 illustrates the behavior of GP – the mutation and crossover do not use nor learn any information (crossover only partially utilizes selection information).

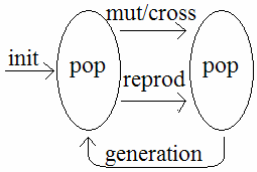

Figure 2. GP in action.

3. RESTRICTING THE REPRESENTATION WITH STRONG AND WEAK CONSTRAINTS

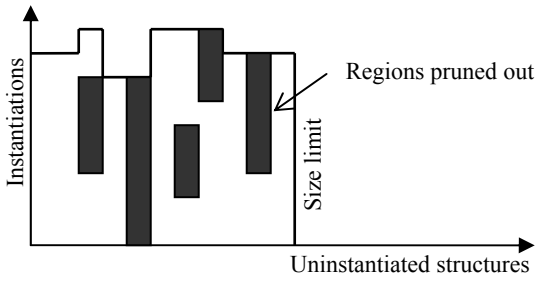

Figure 3. GP 2-D search space pruned by some strong constraints.

As mentioned in section 1, some techniques have been developed to impose strong constraints on GP, including STGP, CFG-based GP, and CGP. They all use different means for imposing the constraints, but the end result is the same – they impose local or global constraint on possible instantiations of GP trees (indirectly they may also limit the number of uninstantiated tree structures). Therefore, the representation changes. For example, assume that functions and terminals are F={*f1*,*f2*} (both unary) and T={*t1*,*t2*}. Suppose we want to prevent *f1* from using *t1* as its child, a strong constraint. We may ensure that no initial trees invalidate this constraint. Or selection can discover this constraint, if indeed beneficial, by removing trees which use *f1*-

>*t1* subtrees. However, as stated before, mutation can easily produce a tree with the *f1->t1* subtree. Moreover even in the absence of mutation, and the absence of trees with *f1->t1* subtrees, crossover can move a *t1* leaf to become a child of a node labeled *f1*, again generating the undesired *f1->t1* subtree. The above mentioned methodologies prevent this from happening, thus effectively removing chunks of the search space of figure 1. The resulting GP search space as illustrated in figure 3.

CGP provides the same potentials with strong constraints, by types or by explicit listings. However, it also allows the weak local constraints (plus weak root constraints). CGP can change the genotype representation space by stating that the root node can be labeled with *f1* or *f2* with probability of 80% and 20%, respectively. Or it can also state that *f2* cannot use *f1* (strong constraint) and it can use *t1* or *t2* with probability of 70% and 30%. This probabilistic representation changes the white search space, of figure 3, to non-uniform density space (some trees more likely than others). These weak heuristics, in the *first-order* form (on the root and between any parent-child), are used to initialize the population, and to drive mutation and crossover.

However, both these operators still disregard what GP "learns" from selection, but instead they use user preferences. The weak heuristics cause probabilistic differentiation between GP representations spanned over its function and terminal set. This is illustrated in figure 4.

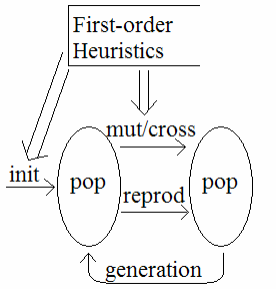

Figure 4. CGP in action.

CGP allows the representations to take probabilistic nature [11]. Using such heuristics to change the effective representation has been shown effective [6][10]. CGP can also use data types as the basis for its heuristics, and it supports type-overloaded functions [9]. Its processing power is illustrated in figure 5, which compares its solving capabilities against those of GP, using the multiplexer function [10][12]. The figure shows the average (out of 10 independent runs) learning curve for GP, and for two cases of CGP when fed with two kinds of heuristics. **CGP1** uses the simple heuristic that the *if* function should only test addresses, straight or negated. **CGP2** extends this heuristic by dropping all functions except *not* and *if*, and by allowing only data or recursive *if* in the action parts of *if*. For more examples of useful heuristics for the multiplexer, see [6][7][8].

One of may ask about the complexity of processing the heuristics. It turns out that due to minimal overhead [6], smaller trees, and the reduction in the search space, **CGP1** and **CGP2** actually complete the 50 generations (no stopping on termination) much faster, as illustrated in figure 4 (**Total** time for the 10 runs). When we measure only the time needed for the best of the 10 runs to find the solution, when executed concurrently (**Until solved**), the difference is much more pronounced (none of the 10 GP runs solved the problem). For illustration of the effects of strong constraints on GP while solving the Santa Fe trail problem, see [10].

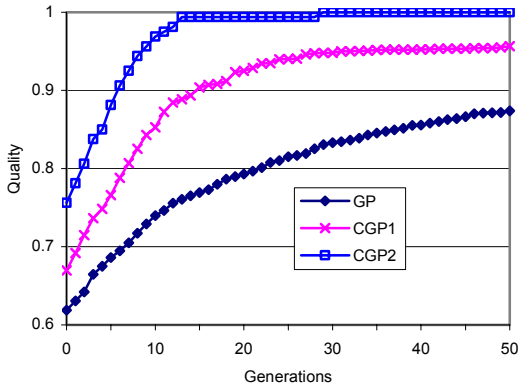

Figure 5. GP and CGP on the multiplexer: quality.

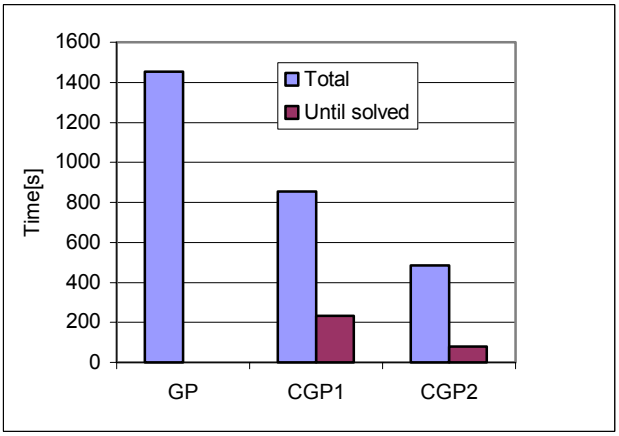

Figure 6. GP and CGP on the multiplexer: timing.

4. LEARNING THE REPRESENTATION THROUGH HEURISTICS

In CGP, the user must specify the representation to solve a problem, either uniform or probabilistic. However, what is the user doesn't know? ACGP was developed to extract first-order heuristics. ACGP works as CGP for a number of generations, after which it analyzes the distribution of the first-order heuristics in the population, uses this information to update the heuristics, reinitializes the population if needed, and starts all over. Thus, the information learned by selection is fed back into initialization, mutation, and crossover [8][9]. This is illustrated in figure 7.

We have demonstrated the capacity of the system using the multiplexer problem in [6][8][9], and we have illustrated it on the Santa Fe trail [10]. Figure 8 compares the average learning curve for GP (base) and ACGP runs. The dips in ACGP's performance correspond to reinitialization the population from scratch after learning new problem representation.

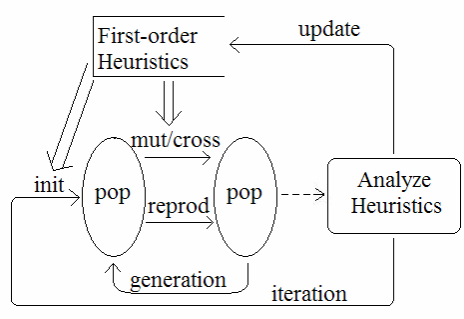

Figure 7. Off-line ACGP in action.

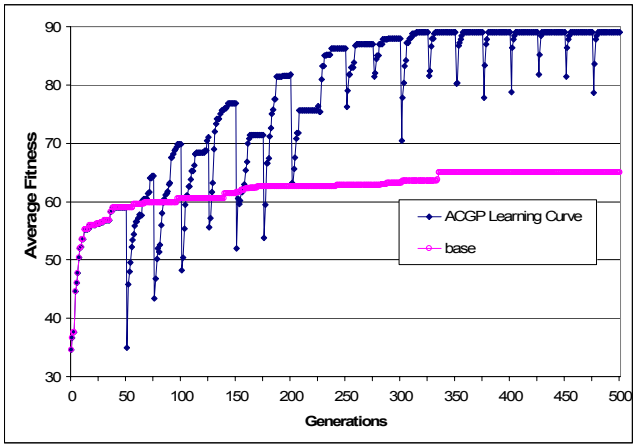

Figure 8. Learning curves on the Santa Fe trail.

5. CONCLUSIONS

We have discussed the effect of strong and weak heuristics on GP representation, and we illustrated the effect of such modified representation on problem solving capability and efficiency. Strong constraints reduce the search space, while additional weak constraints make the remaining search space non-uniform. We have also demonstrated the ACGP system, which learns the probabilistic representation most suitable for solving a problem, by analyzing the genotype subspace identified by selection, and feeding this information into CGP. All discussed constraints are of local and global first-order form, that is only on local parent-child relationship and globally on the root node. ACGP is presently being extended to deal with richer local heuristics as well as more extensive global heuristics.

6. REFERENCES

[1] Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. *Genetic Programming*. Morgan Kaufmann 1998.

[2] Bosman, P.A.N. and E.D. de Jong (2004). Learning Probabilistic Tree Grammars for Genetic Programming.

[3] Burke, Edmund, Gustafson, Steven, and Kendall, Graham. A survey and analysis of diversity measures in genetic programming. In Langdon, W., Cantu-Paz, E. Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M., Schultz, A., Miller, J., Burke, E. and Jonoska, N., editors. *GECCO2002: Proceedings of the Genetic and Evolutionary Computation Conference*, 716-723, New York. Morgan Kaufmann.

[4] Daida, Jason, Hills, Adam, Ward, David, and Long, Stephen. Visualizing tree structures in genetic programming. In Cantu-Paz, E., Foster, J., Deb, K., Davis, D., Roy, R., O'Reilly, U., Beyer, H., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M., Schultz, A., Dowsland, K., Jonoska, N., and Miller, J., editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of LNCS, 1652-1664, Chicago. Springer Verlag.

[5] Hall, John M. and Soule, Terence. Does Genetic Programming Inherently Adopt Structured Design Techniques? In O'Reilly, Una-May, Yu, Tina, and Riolo, Rick L., editors. *Genetic Programming Theory and Practice (II)*. Springer, New York, NY, 2005, 159-174.

[6] Janikow, Cezary Z. A Methodology for Processing Problem Constraints in Genetic Programming. *Computers and Mathematics with Applications*, 32(8):97-113, 1996.

[7] Dasgupta, Dipankar, Janikow, Cezary Z and Chakraborty, Uday. Representations and operators in genetic algorithms". *Proc. 4th International Conf. on Pattern Recognition and Digital Techniques*, Calcutta, India, 1999.

[8] Janikow, Cezary Z. Adapting Representation in Genetic Programming. In K. Deb et al. editors. *Proceedings of Genetic and Evolutionary Computation Conference: GECCO 2004*, 507-518.

[9] Janikow, Cezary Z. ACGP: Adaptable Constrained Genetic Programming. In O'Reilly, Una-May, Yu, Tina, and Riolo, Rick L., editors. *Genetic Programming Theory and Practice (II)*. Springer, New York, NY, 2005, 191-206.

[10] Janikow, Cezary and Mann, Christopher. CGP visits the Santa Fe Trail: the Effects of Heuristics on GP. *Proceedings of Genetic and Evolutionary Computation Conference: GECCO 2005.* To appear.

[11] Janikow, Cezary Z. CGP and ACGP User's Manuals. http://www.cs.umsl.edu/CGP.

[12] Koza, John R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[13] Langdon, William. Quadratic bloat in genetic programming. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., and Beyer, H-G., editors, *Proceedings of the Genetic and Evolutionary Conference GECCO 2000*, 451-458, Las Vegas. Morgan Kaufmann.

[14] McPhee, Nicholas F. and Hopper, Nicholas J. Analysis of genetic diversity through population history. In Banzhaf, W., Daida, J., Eiben, A. Garzon, M. Honavar, V., Jakiela, M. and Smith, R., editors *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1112-1120, Orlando, Florida, USA. Morgan Kaufmann.

[15] Montana, David J. Strongly Typed Genetic Programming. *Evolutionary Computation*, 3(2):199-230, 1995.

[16] Pelikan, Martin and Goldberg, David. Boa: The Bayesian Optimization Algorithm. In Banzhaf, Wolfgang, Daida,

Jason, Eiben, Agoston E., Garzon, Max H., Honavar, Vasant, Jakiela, Mark, and Smith, Robert E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 525-532, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[17] Whigham, P. A. Grammatically-based Genetic Programming. In Rosca, Justinian P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33-41, Tahoe City, California, 9 July 1995.