ADAPTATION OF REPRESENTATION IN GP

CEZARY Z. JANIKOW
University of Missouri – St. Louis
Department of Mathematics and
Computer Science
St Louis, Missouri

RAHUL A DESHPANDE
University of Missouri – St. Louis
Department of Mathematics and
Computer Science
St Louis, Missouri

ABSTRACT
This paper discusses our initial work on automatically adapting Genetic Programming (GP) representation. We present here two independent techniques: AMS and ACE. Both techniques are based on Constrained GP (CGP), which uses mutation set methodology to prune the representation space according to some context-specific constraints. The ASM technique monitors the performance of local context heuristics when used in mutation/crossover, during GP evolution, and dynamically modifies the heuristics. The ACE technique iterates complete CGP runs and then uses the distribution information from the best solutions to adjust the heuristics for the next iteration. As the results indicate, GP is able to gain substantial performance improvements as well as learn qualitative heuristics.

INTRODUCTION

Genetic Programming (GP), proposed by Koza (1992), follows the evolutionary algorithms approach to problem solving by utilizing a population of solutions evolving under limited resources. The solutions, called chromosomes, are evaluated by a user-defined objective fitness evaluation. They compete for survival based on this fitness, and they undergo evolution by means of simulated crossover and mutation operators.

GP differs from other evolutionary computation methods by using trees as the solution representation. Trees provide a rich representation, one that is sufficient to represent computer programs, analytical functions, variable length structures, even computer hardware. The user defines the representation space by defining labels for the trees: functions of specific arities to label the internal nodes, and terminals (functions of no arguments and ephemeral constants) to label the leaves. The arities define the syntactic constraints of the trees. The fitness of a tree is evaluated by recursively applying function/terminal semantics, as defined by the user.

Two important principles need to be satisfied for GP to be successful (Koza, 1992). First, the principle of *sufficency* requires that the set of functions and terminals, along with their semantics, be rich enough to allow building solution trees of desired quality. Unfortunately, the user is often unaware of the minimal set and thus provides an enlarged set to be on the safe side. This enlarges the search space and thus often reduces search efficiency. Second, the principle of *closure* requires that no context constraints be imposed on labeling a tree except for labeling with proper arity. This was required in the absence of methodologies to enforce any such constraints.

Because of the requirements of both sufficiency and closure, the actual representation space that is explored during a GP simulated evolution is much larger than the desired or the actual solution space for the problem in hand. In the late nineties, two similar generic methods for automatic processing of additional constraints were proposed: Strongly Typed GP (STGP) and Constrained GP (CGP) (Janikow, 1996). Both follow the idea of enclosing the search in the feasible (or desired) space. That is, the initial population is generated so that each chromosome satisfies all constraints. Subsequently, every operator (mutation and crossover) is guaranteed to produce constraint-valid offspring if constraint-valid parents are used. Similar ideas have been later incorporated in Context Free Grammar-based GP (CF-GP), where the constraints are in the form of BNF production rules for program construction.

These methodologies allow explicitly expressing a set of local context rules to prune the actual trees that are evolved – only those satisfying the rules will be evolved. For example, an *if* function may now require that the test subtree produce a valid condition.

However, in either of the above methodologies, the user is still responsible for presenting the constraints to the system. Unfortunately, in many cases the user may not be aware of the best heuristics to solve a particular problem.

This paper presents two different techniques, based on CGP, which allow GP to monitor its performance and adjust the heuristics. The heuristics are in the form of *weak* constraints, that is to say they do not prohibit a given local structure to evolve in GP, but rather they allow different local structures to have different probabilities of labeling the tree nodes. However, they can be used as *hard* constraints as well, *e.g.*, when a given heuristic falls below certain probability threshold. The first technique, called AMS, is very simple and uses very limited information. It is based on comparing the fitness of parents to that of offpring in determing the utility of a given local heuristic when used in mutation/crossover. This is expected to work better for crossover, where one offspring is generated by one heuristic. The actual results validate this observation. The other technique, called automatic constraint extraction (ACE), is more complex in terms of run complexity. It iterates CGP runs, with fixed heuristics for each iteration. At the end of each complete CGP run (iteration), the best solutions are evaluated for distribution of local labeling contexts, and these contexts are then applied to modify the heuristics for the next iteration.

In the following, we describe both techniques and then present some experimental results using the 11-multiplexer problem.

CGP – CONSTRAINED GENETIC PROGRAMMING

CGP extends GP to allow mutation and crossover to generate only those offspring that are valid with respect to user provided constraints. The contsraints are in the form of local contexts (parent-child node relationship), and are precompiled into lookup tables - Mutations Sets (MS) (Janikow, 1996). CGP2.1 uses Typed MS structures to accommodate different data types and hence overloaded function instances (Janikow, 1999). For the experiments reported here, we used untyped MS. CGP also allows local heuristics, which are in the form of probabilities of labeling specific context structures. CGP allows the user to specify individual weights for different elements of every mutation set (Janikow, 1996). A weight twice as high as another weight indicates that the corresponding function or terminal should be twice as likely to be used in a given context. A weight set to zero in fact results in removing the corresponding function or terminal from consideration during evolution (only in the specific context, and not globally). For example, the user may specify that a given argument of a particular function be quite unlikely to use recursion on itself, without prohibiting recursion completely. It is this capability that is used in our techniques – both use some runtime information to adjust these probabilities.

ASM - ADAPTATION OF MUTATION SETS BY JUDGING OFFSPRING

The AMS algorithm, presented below, compares the fitness of a parent with its offspring. It then either increases or decreases the probability of the local heuristic that was used to generate the offpring.

1. Run CGP to solve a particular problem, while starting with the user provided constraints and heuristics (none if not given).
2. If crossover is selected to create an offspring (we consider both offspring individually), perform the following:
    a. Select a subtree to be inserted according to the current heuristics for the local context where the subtree is being inserted. This heuristic is denoted with h_i.
    b. Compute Δf_i, the difference in fitness values of the offspring with that of the receiving parent. Positive difference implies that a better offspring is generated.

   c. Modify the specific heuristic h_i, expressed as weight (before normalized as probability), according to: w_i += Δw_c. We use a reward technique such that if Δf_i >0 then Δw_c is a positive constant, otherwise Δw_c is a negative constant (we have experimented with Δw_c proportional to Δf_i but the results were similar).

3. If mutation is selected to create an offspring, perform the following:
   a. Select a subtree to be removed.
   b. Grow the new subtree according to the current heuristics for the local contexts. Note each applied heuristic is denoted at h_i (multiple heuristics are required and a given heuristic can be used multiple times c_i).
   c. Compute Δf_i, the difference in fitness values of the offspring with that of the receiving parent. A positive difference implies that a better offspring is generated.
   d. Modify each used heuristic $h_{i,}$ expressed as weight (before normalized as probability) according to: w_i += $c_i\Delta w_m$. Again, if Δf_i >0 then Δw_m is a positive constant, otherwise Δw_m is a negative constant (again we experimented with Δw_m proportional to Δf_i but the results were similar).
   e. Observe that we used Δw_m much smaller than Δw_c simply because single mutation involved using many heuristics at once and the reward is shared among all heuristics.

ACE – AUTOMATIC CONSTRAINT EXTRACTION

This technique runs on the top of CGP. In other words, it utilizes complete runs of CGP. To avoid statistical anomalies, ACE in fact can utilize a number of independent CGP runs simultanbeously . The algorithm has already been extended, but the results and descriptions will be presented separately.

ACE takes input parameters which determine the number of indepedent runs, the number of CGP generations per run, and the number of iterations that ACE is to run. The iterations are also stopped if no further significant distribution change is detected. Other parameters determine the number of best quality trees to be investigated from each CGP run, the total number of trees selected from all the independent runs, as according to user-specific objectives, and the objectives guiding the selection. At present, the selection can be based on quality or size in any combination. The selected trees are investigated for the distribution of functions and terminals, and the discovered distribution is fed back into CGP to adjust the constraint weights. Weights falling below specific threshold can also be dropped to 0 (thus becoming *hard* constraints).

For example, if ACE runs 10 independent runs, and it extracts the 15 best quality trees from each run, it ends up with 150 trees at the end of the iteration. If 20% of the trees are to be selected for investigation, and the objective is to minimize size, the 30 smallest trees from the 150 quality trees are selected. Distribution statistics from the 30 trees is computed, and it is fed back into the next ACE iteration in the form of CGP weights (or just adjustments of the weights).

ACE can start with no constraints or it can start with user-defined constraints and initial heuristics.

EXPERIMENTS AND RESULTS

To illustrate both techniques, we selected the well known and studied 11-multiplexer problem [1][3]. The 11-multiplexer problem is to discover the boolean function that passes the correct data bit (out of eight bits *d0…d7*) when fed three addresses (*a0… a2*). There are 2048 (2^{11}) possible combinations. Koza [3] and others have used a set of four atomic functions to solve the problem: *if/else*, *and*, *or*, and *not*, in addition to the data and address bits. This set is not only sufficient but is also redundant. In (Janikow, 1996), we have shown that operating with only *if* (sufficient by itself) dramatically improves the performance by simply reducing the search space. Moreover, we have shown that the performance is further enhanced when we restrict the *if*'s condition argument to test addresses only, while restricting the two action arguments to select only data bits or recursive *if*.

Name	Experiment	Crossover/Reproduction/Mutation
E1	No AMS w/all operators	**85/10/5%**
E2	AMS w/crossover+rep	**90/10/0 %**
E3	AMS w/mutation+rep	**0/10/90%**
E4	AMS all operators	**85/10/5%**

Table 1: AMS experiment setup.

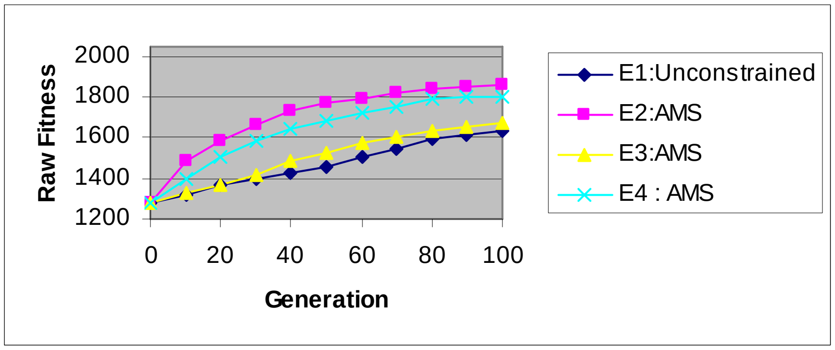
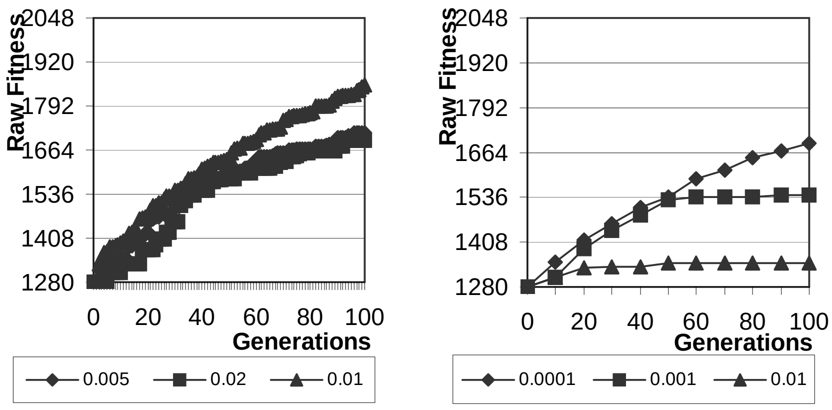


Figure 1: AMS experiments in practice.

For AMS, our objective was to determine if CGP can improve its run by adjusting the heuristic weights. In addition, another objective was to evaluate the utility of mutation and crossover for the technique. We used $\Delta w_c = 0.01$ and $\Delta w_m = 0.0005$ for $\Delta f_i > 0$, and $\Delta w_c = -0.01$ and $\Delta w_m = -0.0005$ otherwise. We performed two sets of different experiments. The first set, as defined in Table 1, started with no constraints or heuristics and different operators while running with and without AMS. The results are illustrated in Fig. 1. They indicate that AMS is able to adapt the heuristics while running CGP so that the overalll performance improves very significantly. The only case when AMS does not show significant improvement is when it is run with mutation only (reproduction accesses only global fitness as opposed to adapting local heuristics). This effect was expected – a single mutation requires many heuristics yet there is only one offspring. In crossover, on the other hand, one offspring is generated by one heuristic.



(a) (b)

Figure 2: Effects of selecting different Δw_c (a) and Δw_m (b) in AMS.

The second set of experiments was designed to determine the influence of some of the AMS parameters, namely Δw_m and $\Delta w_{c.}$ The results are presented in Fig. 2. As seen, the magnitude of crossover adjustment has little impact on the resulting improvement in evolution speed. However, the magnitude of mutation adjustment proves to be very important. For some values, the adjustments produce no improvements over multiple generations, while for others, the effect is always positive (0.0001 in this case). This verifies our previous intuition that in this case of very limited information, when only the parent-child relationship is observed, it is much more difficult to derive any heuristic information from mutation due to its necessity to apply multiple heuristics before any feedback is received. Some more sophisticated bucket brigade algorithms may prove more beneficial.

To test the ACE technique, we conducted three different experiments with three different extraction objectives: quality, size, and a mix of the two. All experiments were conducted with 10 independent CGP runs, 10 iterations, and all remaining parameters being the dafult CGP parameters (Janikow 1996).

Fig. 3 (left) illustrates the best quality tree from among the trees selected for computing the distribution statistics at the end of every ACE iteration. As can be seen, we gain perfect performance just after two iterations. When optimizing for size or mix (size+quality), we may see a dip after iteration 6 – size optimization may comprimize quality. Except for the dip, there seem to be no apparent improvement after the saturation. However, this is not so. As illustrated in Fig. 3 (right), the number of required generations for an iteration to be successful keeps decreasing after that saturation (ieration 2).

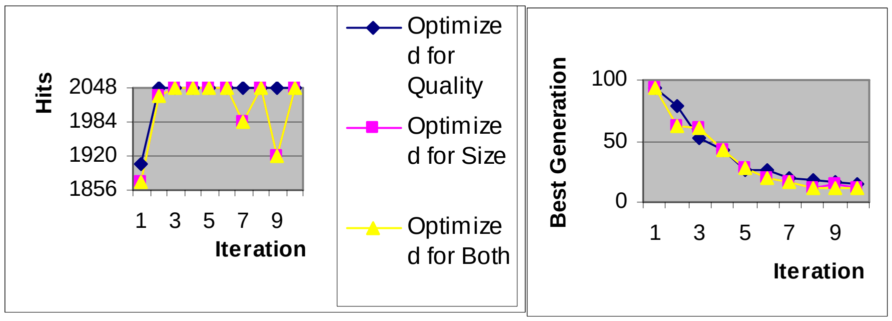
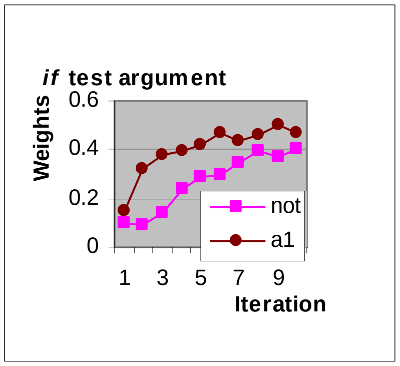


Figure 3: Improvements due to ACE: best of iteration (left) and the number of generations needed to solev the problem, per iteration (right).

The next question is that of the interpretation of the evolve heuristics. Fig. 4 illustrates the evolving distribution of the condition part of *if* . As stated before, the best prior results were obtained while testing only addresses. Fig. 4 left indicates that the evolved heuristics was to use address *a1* and the function *not* in the condition part. However, the same figure on the right indicates that the remaining two addresses, *a0* and *a2*, were discovered to be used indirectly through the argument of the *not* function. Thus, all three addresses are tested by *if* either directly or indirectly. The remaining functions are terminals have been automatically degraded to zero total weight and thus removed from consideration.

CONCLUSION AND FUTURE RESEARCH

We have presented a new methodology for adapting the representation of solutions in GP. The methodology is based on CGP, which allows heuristics to be used for biasing the representation space. The heuristics are local, that is they differentiate function/terminals to be used in specific parent-child contexts. In the AMS technique, these heuristics are adapted by observing the utility of specific heuristics when used to create offspring in crossover and mutation and when judged by the relative fitness of the offspring as related to that of the parent. This particular technique is very simple yet still able to generate significant improvements on the speed of evolution of solutions. In the ACE technique, the heuristics are adapted by evaluating the distribution of labels in the best trees selected from a pool of independent CGP runs. This technique uses more reliable information, and it seems to produce much better results yet at a higher computational cost. Further improvements and results will be presented separately.



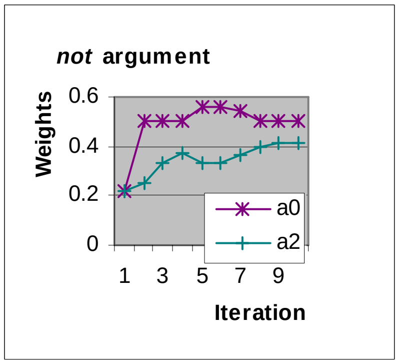


Figure 4: Direct (left) and indirect distribution on the condition part of *if*.

REFERENCES

Janikow, C.Z., 1996. "A Methodology for Processing Problem Constraints in Genetic Programming", Computers and Mathematics with Applications, Vol. 32, No. 8, pp. 97-113, 1996.

Janikow, C.Z., 1999. "CGP 2.1 User's Manual".

Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Natural Selection. The MIT Press, Cambridge, MA.

Bonzaf, et al. 1998. Genetic Programming: an Introduction. Morgan Kaufmann.