Hierarchical Self-Organization in Genetic Programming

Justinian P. Rosca
Computer Science Department
University of Rochester
Rochester, NY 14627
rosca@cs.rochester.edu

Dana H. Ballard
Computer Science Department
University of Rochester
Rochester, NY 14627
dana@cs.rochester.edu

Abstract

This paper presents an approach to automatic discovery of functions in Genetic Programming. The approach is based on discovery of useful building blocks by analyzing the evolution trace, generalizing blocks to define new functions, and finally adapting the problem representation on-the-fly. Adaptating the representation determines a hierarchical organization of the extended function set which enables a restructuring of the search space so that solutions can be found more easily. Measures of complexity of solution trees are defined for an adaptive representation framework. The minimum description length principle is applied to justify the feasibility of approaches based on a hierarchy of discovered functions and to suggest alternative ways of defining a problem's fitness function. Preliminary empirical results are presented.

1 INTRODUCTION

Automatic discovery of problem representation primitives is a challenging goal that has been tackled in recent research in machine learning in general and Genetic Programming (GP) in particular. Its importance comes from the fact that a representation with higher level primitives can transform a hard problem into an easy one. This paper shows that in Genetic Programming efficient structure synthesis can be obtained by discovering useful genetic material and using it globally to adapt the problem representation. The discovery process is based on an analysis and tracking of substructures called *building blocks* over generations of evolution. We describe how building blocks result from several heuristic criteria for extending the problem representation. These are used to extend the GP paradigm in what we call Adaptive Representation (AR) GP.

In the following sections we define the concepts of GP blocks and building blocks and show that the interpretations given provide the basis of a theory for adapting problem representation and making GP search more efficient. Next we introduce different notions of complexity useful in controlling the algorithm. We view the problem of determining a GP program that satisfies the criteria imposed by the fitness function as one of hypotheses formation. This enables us to apply the minimum description length principle to analyze the power of approaches based on automatic discovery of functions. We conclude by analyzing an example, mentioning related work and considering directions for future work.

2 BUILDING BLOCKS IN GP

In Genetic Algorithms (GA), the schemata theorem [Holland, 1992], [Goldberg, 1989] summarizes the effect of fitness-proportionate reproduction, crossover and mutation. Schemata are template strings representing sets of individuals in the search space. Schemata are defined by strings over the (usually binary) alphabet extended with a *don't care* symbol. The importance of schemata lies in the characterization of how GAs work, both at the representation and processing levels. The schemata theorem proves that individuals that match short defining length (distance between the first and the last specific string positions) schemata with fitness values above population average will be selected in a geometrically increasing number of samples in the next generations. Thus at the representation level, schemata outline the role of *building blocks*. A building block is a schema of short defining length whose defined positions tend to improve the behavior of individuals that match it and which describes fragments of a final solution. Individuals in the population provide a partial evaluation of schemata they match. GAs work best when the internal representation encourages the emergence of useful building blocks that can subsequently be combined with each other to improve performance.

In GP no precise definition of a schema exists, and no corresponding schemata theorem has been proven. One more difficulty is outlined in [Angeline, 1994]: Since GP individuals represent code, as opposed to GA strings which are collections of data, there exists a complicated many-to-many mapping between genotypic and phenotypic features.

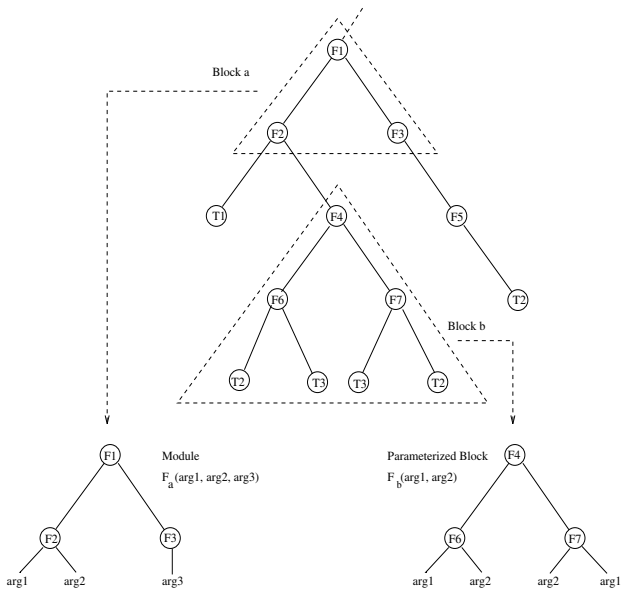

Figure 1: Bottom-up approach to block selection vs. the module approach. The bottom-up approach (right) generalizes the block b to a function $F_b(arg1, arg2)$; The modular approach (left) generalizes an arbitrary portion of the tree.

An individual in GP is a piece of "code" in a base language (LISP in our experiments) and can be represented by its parse tree. We define blocks as entire subtrees of a given maximum height from population individuals. This represents the bottom-up approach in figure 1. (In contrast the *module approach* [Angeline and Pollack, 1994] starts with subtrees of any depth and chops off all branches at a given depth.) We will show that this *bottom-up approach* enables the definition of blocks whose usefulness can be evaluated. A key idea is that although one might like to keep track of blocks of arbitrary size, only monitoring the merit of small blocks is feasible. However useful blocks tend to be small and the process can be applied recursively to discover more and more complex useful blocks.

A block can be generalized by substituting each occurrence of a terminal of the block by a variable (descriptive generalization in [Michalski, 1983]). This process transforms a block into a parameterized function whose parameters are given by the block variables. Figure 1 presents an example of a function F_b generated in this approach.

In GP, a good building block corresponds to a parameterized piece of code that once used as a problem primitive has noticeable good effect on the behavior of the individuals. Thus, a building block has the potential to successfully extend the set of function primitives of the problem. For a comparison with genetic algorithms (GA), note that if we place in the initial population individuals that match useful schemata, the convergence of the GA improves dramatically. We explain this by the reduction in dimensionality of the search space caused by the "biased" initial population. In GP such an improvement could be explained by the more powerful set of primitives. Thus our goal is the discovery of new functions based on an analysis of blocks. This analysis leads to candidate building blocks and the dynamic extension of the function set with new functions created on-the-fly from promising candidates. We call such a representation an *adaptive representation*, as it is automatically changed while the GP program searches the space of admissible solutions.

3 ANALYSIS OF CANDIDATE BUILDING BLOCKS

There exist two obvious choices for candidate building blocks: fit blocks and frequent blocks. A block whose merit (fitness or frequency) exceeds a threshold becomes a candidate.

Frequent blocks are determined by keeping track of how often a block appears in the entire population. Surprisingly, frequent blocks are not necessarily useful building blocks. By analogy to the GA schemata theorem, a good building block spreads rapidly in the population, and this determines a high frequency count for the block. For example, in the even parity problem (see section 6), if we augment the function set with the exclusive OR function (XOR), then XOR (a building block) will soon become dominant in fit program trees. The problem is that the *converse* of the GA schemata theorem is not true. Poor blocks, i.e. blocks that are identities or add no functionality, may be frequent and should not be considered as candidates, although they may have a role of preserving recessive features (introns in [Angeline, 1994]).

Some conclusions obtained by monitoring frequent blocks indicate their unsuitability in estimating block usefulness [Rosca and Ballard, 1994]. Blocks that appear in a final solution may be discovered very late in the search process. They are not necessarily responsible for the evolution process, usually having a low frequency count. Frequent blocks in early generations may become rare in late generations. The fact that simpler and simpler blocks (and highly probable to be less useful) become more frequent as the number of generations increases complicate the process even more.

A much better choice for discovering building blocks is to consider fit blocks. We can incrementally check for new fit blocks instead of relying on expensive statistics in the population.

Several methods can be used to evaluate the fitness of a block. First, one can use the program fitness function to evaluate blocks too. This has the advantage of requiring no more domain knowledge than the knowledge built into the fitness function, but is not a general method. Second, one can use a slightly modified version of the fitness function, corresponding to a lower dimensionality problem of the same type. For example the block fitness may be measured only on a reduced set of fitness cases, dependent on the variables used in the block. This method actually scales

the fitness function down to cope with a smaller size problem (see the example from section 6). Third, the user can provide one or more *block fitness functions* based on supplementary domain knowledge which is often available and can be successfully used for complex problems. Each of the block fitness functions exerts "environmental pressure" for the selection of viable blocks. In a co-evolutionary framework such pressure could come from co-evolving species [Hillis, 1990].

As expected fit blocks are very useful in dynamically extending the problem representation by means of definition of new global functions. A highly fit block generated in the initial population spreads in the population very easily. Once the population evolves on a right path, good blocks appear among the frequent blocks.

4 ADAPTIVE REPRESENTATION

Based on an analysis of candidate building blocks we hypothesize on-the-fly new functions which dynamically extend the function set. The initial members of the function/terminal set are the initial problem building blocks.

The evolution process is naturally split into *epochs*. An epoch is a sequence of consecutive generations in which no candidate building blocks are discovered (have merit above the threshold). Once at least one such block is discovered in a generation, the population undergoes substantial changes. The most fit individuals are retained, while the less fit ones are replaced by new individuals randomly generated using the extended function set. This is the beginning of a new epoch. The proportion of individuals replaced is a parameter of the algorithm.

In order to identify building blocks of height less than a given parameter g_{max} we take advantage of the known population dynamics. All new blocks can be discovered efficiently ($O(M)$ time, where M is the population size) by marking the program-tree paths actually affected by a GP operator and by examining only those paths while searching for new blocks [Rosca and Ballard, February 1994].

The skeleton of the GP algorithm that incorporates the AR component is presented in figure 2. It also outlines the steps involved in adapting the problem function set $\mathcal{F}_i$ at generation i.

GP skeleton

Define problem: terminal set $\mathcal{T}$, function set $\mathcal{F}_0$, fitness/cost function $\mathcal{C}$, set of fitness cases $\mathcal{E}$, block fitness functions $\mathcal{C}_{BB}$. Denote the population at a given generation i by $\mathcal{P}_i$.

1. Initial Generation: $i = 0$
2. Randomly initialize population $\mathcal{P}_0(\mathcal{T}, \mathcal{F}_0)$
3. Evaluate population($\mathcal{P}_0, \mathcal{E}, \mathcal{C}$)
4. Repeat until termination criterion is met (generation i)
   (a) Generate a new population $\mathcal{P}_{i+1}$ by reproduction, crossover, mutation of individuals($\mathcal{P}_i$)
      i. Select genetic operation O (O_r, O_c, O_m)
      ii. Select winning individuals $\mathcal{W}$ from current population ($O, \mathcal{P}_i$)
      iii. Generate offspring(O,$\mathcal{W}$,$\mathcal{P}_{i+1}$)
   (b) Adapt representation($\mathcal{P}_i, \mathcal{F}_i, \mathcal{F}_{i+1}$)
   (c) Enrich population $\mathcal{P}_{i+1}$ ($\mathcal{T}, \mathcal{F}_{i+1}$, epoch-replacement-fraction)
   (d) Evaluate population($\mathcal{P}_{i+1}, \mathcal{E}, \mathcal{C}$)
   (e) Next generation: $i = i + 1$

Adapt_representation($\mathcal{P}_i, \mathcal{F}_i, \mathcal{F}_{i+1}$)

1. Discover candidate building blocks $\mathcal{BB}_i$ by evaluating each block's merit($\mathcal{P}_i$);
2. Prune the set of candidates($\mathcal{BB}_i$);
3. For each block in the candidate set, $b \in \mathcal{BB}_i$, repeat:
   (a) Determine the terminal subset T_b used in the block(b);
   (b) Create a new function f having as parameters the subset of terminals and as body the block(b, T_b);
   (c) Extend the function set $\mathcal{F}_{i+1}$ with the new function($\mathcal{F}_i, f$).

Figure 2: Skeleton of a GP algorithm and the Adaptive Representation extension

5 DESCRIPTIONAL COMPLEXITY IN GENETIC PROGRAMMING

In many GP applications the fitness function is extended, for reasons of efficiency or convergence, with *size pressure* or *execution bound* components. This has been done mostly in an empirical way. For example, individuals with too big a size, number of state-changing operations (as in the LAWN MOWER or ARTIFICIAL ANT problems described in [Koza, 1994]) or "running time" are penalized. It is very important how we measure size or running time and how we include such factors in the fitness calculation. A change in the ranking of individuals given by a pure performance fitness function determines a reshaping of the fitness landscape. Within the context of the given representation and set of genetic operators, search may become easier or harder after the change. This section proposes a theoretical basis for analyzing such additional measures.

5.1 COMPLEXITY MEASURES FOR PROGRAM TREES

Suppose an individual is represented by a program which calls discovered functions which may call older discovered functions. Nonetheless the call graph based on the caller-callee relation has no cycles. Let $Size(F)$ be the number of nodes in the tree representing a program F. Let F_0 be the program tree representing an individual T_0 which contains direct or indirect calls to $F_1, F_2, ..., F_m$. We define *structural complexity* $SC(F_0)$ and *evaluational complexity* $EC(F_i)$ in the following way:

$$SC(F_0) = \sum_{0 \le j \le m} Size(F_j) \quad (1)$$

$$EC(F_i) = Size(F_i) + \sum_{j \in J_i} EC(F_j) \cdot |Calls(F_i, F_j)| \quad (2)$$

where $J_i = \{j | i < j \le m$ and $F_i \ calls \ F_j\}$ and $|Calls(F_i, F_j)|$ is the number of times F_i calls F_j. In standard GP, where no functions other than the primitive ones are used, the structural and evaluational complexities are equal to the program size $Size(F)$. Assuming that functions from the initial function set are executed in unit time, the evaluational complexity shows how many time units it takes to execute an individual program. Note that $EC(F)$ is different from the structural complexity of individuals after the "inline" expansion of all the called functions down to the primitive functions, which we call "expanded structural complexity" and represents a true measure of the size of an individual if we had to built it from primitive functions. It corresponds to the notion of circuit size complexity from complexity theory. The following inequalities between the complexity measures introduced hold:

$$Size(F) \le SC(F) \le EC(F) \le Expanded\text{-}SC(F)$$

5.2 MINIMUM DESCRIPTIONAL COMPLEXITY

We can view the problem of determining a program that explains the set of examples or optimizes a fitness function as one of hypotheses formation: we look for the best program that explains a set of constraints (data) captured by the fitness function. Rissanen's *minimum description length* (MDL) principle offers an answer to approaching the problem. It states that the best theory to explain a set of data is one which minimizes the length of the data description together with the hypothesis description. In general, problems such as the inference of a decision tree that best explains a set of examples [Quinlan and Rivest, 1989], the construction of a finite automaton or the inference of a boolean function that satisfies a set of constraints are all problems that match the described pattern and can be solved using the MDL principle [Li and Vitanyi, 1993]. MDL is also called *stochastic complexity*.

The MDL principle advocates a hierarchical representation of evolved programs (see appendix A). A more complex behavior can be obtained based on a more complex, hierarchical organization. For such an organization, the size of individuals and discovered functions is kept within reasonable bounds while the structural complexity of individuals is much bigger. The power of GP using automatically discovered functions to solve problems is significantly increased too ([Koza, 1992], [Rosca and Ballard, 1994], [Koza, 1994], [Kinnear, 1994]).

Moreover, the descriptional complexity can be used as a measure for the fitness of an individual T that would drive GP towards discovering solutions with a smaller descriptional complexity. It balances the requirement of getting a simpler description of a solution tree T and the requirement of minimizing the number of misses. The mechanism for adapting the representation while searching for solutions represents a natural way to generate a hierarchy of more and more complex functions (a hierarchical representation) and to make possible the discovery of a solution of small (or even minimal) description length.

We suggest here that evolution minimizes the stochastic (descriptional) complexity, but increases the structural complexity of individuals. Individuals tend to take a hierarchical organization because this is a way of working towards achieving a minimum descriptional complexity, for a given "power" in explaining the input fitness cases. This is true to certain degree even with simple fitness measures incorporating rewards for hits and penalties for misses as well as a size pressure component, that have been empirically used so far in GP applications.

6 EXPERIMENTAL RESULTS

The results discussed herein are based on runs with the AR-GP kernel [Rosca and Ballard, 1994]. The main parameters were population size $M = 4000$, number of generations $G = 50$. Selection is done by running tournaments of size 7. The epoch-replacement-fraction parameter was $\frac{1}{2}$. Other GP parameters have typical values as reported in [Koza, 1992]. We show that we can improve the performance of GP even for very sparse solution spaces.

6.1 LEARNING THE EVEN-N-PARITY FUNCTION

The EVEN-N-PARITY problem is to find a logical composition of primitive boolean functions that computes the sum of input bits over the field of integers modulo 2. It is solvable by problem decomposition into simpler subproblems and thus represents a good test bench for the discovery of more and more complex building blocks. This function appears to be difficult to learn in a GP implementation especially for values of n greater than 5.

The initial function set contains primitive boolean functions of two variables:

$$\mathcal{F}_0 = \{AND, OR, NAND, NOR\}$$

The terminal set is defined by a set of boolean variables:

$$\mathcal{T} = \{D_0, D_1, D_2, ..., D_{n-1}\}$$

Any boolean function of n variables is defined on the set of 2^n combinations of input values. Given a program implementing a boolean function, we compare its performance on all possible combinations of boolean values for the input variables against a table of fitness cases $\mathcal{E}$ defining the EVEN-N-PARITY function. Each time the program and the EVEN-N-PARITY table give the same result, the program records a *hit*. GP has to search for a program that has the

Table 1: Important steps in the evolutionary trace for a run of EVEN-8-PARITY

Generation 0. New functions [F681]: (LAMBDA (D3) (NOR D3 (AND D3 D3))); [F682]: (LAMBDA (D4 D3) (NAND (OR D3 D4) (NAND D3 D4)))

Generation 1. New function [F683]: (LAMBDA (D4 D5 D7) (F682 (F681 D4) (F682 D7 D5)))

Generation 3. New functions [F684]: (LAMBDA (D4 D5 D0 D1 D6) (F683 (F683 D0 D6 D1) (F681 D4) (OR D5 D5))); [F685]: (LAMBDA (D1 D7 D6 D5) (F683 (F681 D1) (AND D7 D7) (F682 D5 D6)))

Generation 7. The solution found is: (OR (F682 (F682 (F683 D4 D2 D6) (NAND (NAND (AND D6 D1) (F681 D5)) D1)) (F682 (F683 D5 D0 D3) (NOR D7 D2))) D5)

maximum number (2^n) of hits. The cost or standardized fitness of a program i having $Hits(i)$ hits, and a size $Size(i)$ is:

$$\mathcal{C}_{standardized}(i) = [2^n - Hits(i)] \cdot C1 + Size(i) \cdot C2$$

where $C1, C2$ are constants. According to the general rule (5, see Appendix) that estimates program descriptional complexity, this intuitive formula accounts well for the errors (misses) if we take $C1 = n$, but it does not account correspondingly for program complexity. Nonetheless it suggests good choices for the two constants. All the experimental results reported here are based on this decision. We have also tested a formula derived directly from the MDL principle with poorer results. One possible explanation is the greater pressure on eliminating dead regions of code which may prove to be a reservoir for diversity later on. The block fitness function was the same as the fitness function, with $C2 = 0$ and one slight change. Hits are evaluated on a subset of the set of fitness cases, determined by fixing the values of variables not used in the block to arbitrary values. This is equivalent to taking simpler (shorter) parity examples. Only blocks with a maximum number of hits are considered as candidates.

We solved all PARITY problems up to order 11 on a SUN SPARCstation 10 by adapting the representation based on fit blocks.

It is important to point out the hierarchical structure of the functions dynamically created and their semantics. Table 1 presents the main steps of the trace in a run of EVEN-8-PARITY and figure 3 presents the final call-graph induced in that run. Different levels in the call graph correspond to higher epochs of the evolutionary process. Functions on the same level do not call one another. Only functions in different epochs may take advantage of the genetic material discovered previously. The foundation of the hierarchy is made of the primitive functions included in $\mathcal{F}_0$. We have argued from a theoretical point of view that hierarchical structures are more powerful than structures based on the

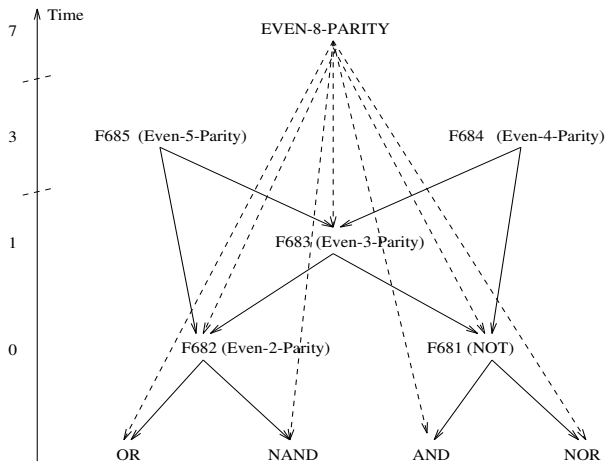

Figure 3: Call graph for the extended function set in the EVEN-8-PARITY example

Table 2: Comparison of results (rounded figures): AR-GP vs. results reported by Koza in [8], marked ($^{\#}$). SC is the structural complexity, G is the number of generations

Method	EVEN-3		EVEN-4		EVEN-5		EVEN-8	
	G	SC	G	SC	G	SC	G	SC
GP$^{\#}$	5	45	23	113	50	300		
ADF-GP$^{\#}$	3	48	10	60	28	157	24	186
AR-GP	2	17	3	15	5	32	10	41

initial function set. Table 2 brings some experimental evidence. With ADF or AR the scalability of the even parity problem improves significantly. Row 1 from table 2 shows the number of generations to find a solution with 99% probability in one run and the average structural complexity of solution trees obtained for ten runs of AR-GP on EVEN-PARITY problems with population size $M = 4000$. Rows 2 and 3 present some comparative results taken from [Koza, 1994] for sample runs of GP with similar parameter values, but $M = 16000$.

Figure 4 presents the evaluational complexity and the size of both the best of generation individual and on average over the entire population. The structural complexity values are bounded from below and above by $Size$ and EC respectively. The best of generation individual becomes simpler and simpler due to size pressure and the variety of useful and more powerful blocks appeared in the population. Starting with generation 3 discovered functions begin to dominate the structure of the best of generation individual as they gradually replace the primitive functions. A statistical analysis of the frequent blocks after the function set is extended on the basis of fit blocks, outlines the importance of the new functions created, and thus of the hypothesized building blocks. The new functions rapidly become dominant in the population, if they are useful. We can thus evaluate extensions of the function set.

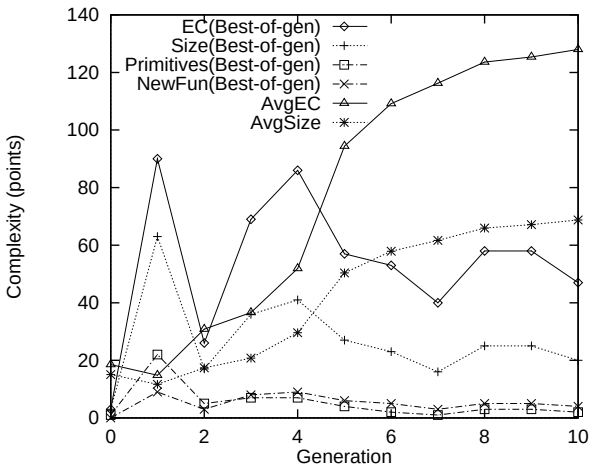

Figure 4: Complexity of best of generation individual and average values over the entire population in the EVEN-8-PARITY example

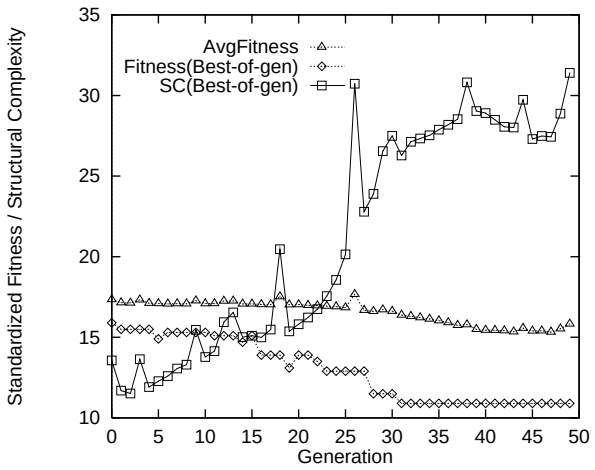

Figure 5: With AR inhibited, an EVEN-5-PARITY run shows permanent increase in structural complexity but a plateau in fitness

Figure 5 outlines that the complexity of individuals increases considerably for a simpler problem (EVEN-5-PARITY) when standard GP is applied, without bringing noticeable improvements in the standardized fitness. A similar problem would appear if the new functions created by AR did not correspond to good building blocks.

7 RELATED WORK

The *Automatic Definition of Functions* (ADF) approach [Koza, 1992] has opened this line of research in GP. In ADF-based GP each individual has a fixed number of components: functions to be automatically evolved (having a fixed number of parameters) and result producing branches. Each component is a piece of LISP code built out of specific primitive terminal and function sets, and is subject to genetic operations. The primitive sets are defined such that the components form a hierarchy similar to our call-graph. Although the hierarchy is fixed, the definition of its elements is not known initially. Genetic operations are constrained depending on the components on which they operate. For example crossover can only be performed between components of the same type. Although the hierarchy of components is fixed, it boosts the power of solution discovery especially for problems with regular solutions or decomposable into smaller subproblems.

The *module acquisition* approach [Angeline, 1994] is based on the creation and administration of a library of modules which extend the problem representation and on the use of two new genetic operators, compress and expand, that control the process of modifying population individuals. A module is a function with a unique name defined by selecting and chopping off branches of a subtree selected randomly from an individual. The function's parameters are determined by the places where the tree is trimmed (see figure 1). The ideas of modifying the expressiveness of the base language and manipulating genetic material, in particular by "freezing" portions of it as an effect of the compress operator, are very powerful although their practical application has proved to be less efficient than ADF.

A performance comparison of ADF and module acquisition is presented in a recent paper [Kinnear, 1994]. The author analyzes the performance of the two methods and many other variations of the methods and attributes the better performance of ADF to the repeated use of calls to automatically defined functions and to the multiple use of parameters. An attempt to explain the course of evolution in GP based on an understanding of what building blocks are appears in [Tackett, 1993]. The idea that frequent subtrees in one individual correspond to synthesized features suggests the conclusion that those subtrees comprise "building blocks".

The minimum description length principle has been recently used in an application of Genetic Programming to pattern recognition [Iba *et al.*, 1994]. The standard GP representation of individuals is transformed into a decision tree representation. The fitness function used relies on the formulas derived in [Quinlan and Rivest, 1989]. A minimum description length fitness function drives search towards minimum true error rate decision trees for the underlying classification procedure.

8 CONCLUSIONS AND FUTURE WORK

In building a system it is desirable to distinguish the elementary components of the system from more complex components used in defining subsystems. The functionality of a complex system is based on elementary parts which are

its *stable components*. The discovery of stable components when solving a problem could speed the search process, as the time needed for the complex system to evolve based on its subsystems is much shorter than if the system evolves from its elementary parts [Simon, 1973].

We claim that in GP building blocks emerge that correspond to such stable components. They can be discovered in the population using a process of generalization of substructures (blocks) already incorporated in the individuals. Automatic adaptation of the representation is implemented by extending the function set with *fit building blocks* discovered. Evaluation of blocks can be easily performed for the class of problems with scalable fitness functions, property which characterizes a rich set of real-world problems. Moreover such an evaluation corresponds to a natural way in which learning can be driven, starting from simple training examples and working towards more complex ones. The effect of our approach is a dynamical reshaping of the search space and exemplifies the effect learning, in the form of acquisition of new functions, has in the evolutionary process [Hinton and Nowlan, 1987]. Extraordinary improvements of the effort needed to find a solution have been observed for problems with regularity in their solutions. Developing robust criteria for discovering useful building blocks in general could help in the design of GP applications for complex problems.

On the theoretical level, issues such as problem complexity and its relation to problem representation, or defining robust/adaptive measures for implementing the fitness function would represent interesting lines of research. An analysis based on the minimum description length principle proves the power of a hierarchical approach to problem solving in GP. The structural, evaluational and descriptional measures of complexity for a hierarchical problem representation help in understanding how complex the problem is and what kind of fitness pressure could create an efficient selective pressure on the GP population. Evolution generates individuals of increased structural complexity but not necessarily of increased descriptional complexity.

This extended GP paradigm also suggests a general way in which more domain knowledge can be used to aid in the discovery of useful building blocks and to adapt the problem representation dynamically.

APPENDIX: Minimum Description Length Principle for Hierarchical Organizations of the Function Set

We can apply the minimum description length principle to interpret how good a hierarchical representation is. We take the simpler case of binary trees, although similar conclusions can be obtained for trees with bounded degree nodes. A MDL encoding of a GP individual is given by a minimum encoding of its program tree T and its misses $Misses(T)$. A tree T of size n can be encoded reasonably as an array of length n. Each element of the array encodes the label of the nodes and the indices of its two children. Labels are taken from a set of size A, thus the array can be encoded using $n \cdot (\log A + 2 \cdot \log n)$ bits. Furthermore, for a fixed total number of fitness cases k, the number of misses of a program tree T can be codified with approximately $Misses(T) \cdot \log k$ bits. A measure of this codification is the *descriptional complexity* of T, $DC(T)$:

$$DC(T) = n \cdot (\log A + 2 \cdot \log n) \; + \; Misses(T) \cdot \log k \quad (3)$$

We want to search for trees with no misses, and a minimal descriptional complexity. In the general case, we use the additive property of DC. Let a fixed individual program be represented by a tree F_0 with a size $n_0 > 0$ which calls (directly or indirectly) automatically discovered functions $F_1, F_2, ..., F_m$ having sizes $n_1, n_2, ..., n_m$ with $n_i \geq 0$ for all i. A depends on the function and terminal set sizes and on m:

$$A = |\mathcal{F}| + |\mathcal{T}| + m \quad (4)$$

The structural complexity n of T is computed by formula (1). We also consider that the number of misses for the functions is 0. Thus the encoding of T is:

$$DC(T) = \sum_{0 \leq j \leq m} (2 \cdot n_j \cdot \log n_j + n_j \cdot \log A) + \quad (5)$$

$$+ \, Misses(T) \cdot \log k$$

For a fixed total structural complexity

$$SC(T) = n = \sum_{0 \leq j \leq m} n_j$$

and a fixed number of misses, the sum in $DC(T)$ has a maximum if $n_0 = n$ and all $n_j = 0$ and a minimum if all trees have equal size. We can prove this by applying Jensen's inequality [Beckenbach and Bellman, 1965] to the convex function $(\log x + \log A)$.

The case $n_0 = n$ and $n_i = 0$ for all $1 \leq i \leq m$ represents the situation when no (automatically discovered) subfunctions are used. $DC(T)$ is maximum in this case. We obtain a minimum description when a maximum number of automatically discovered functions are used, and the sizes of the trees representing the functions are all equal.

Acknowledgements

This research was sponsored by the National Science Foundation under Grant numbered IRI-8903582 by NIH/PHS research grant numbered 1 R24 RR06853-02 and by a Human Science Frontiers Program research grant.

The authors wish to thank Jonas Karlson, Virginia de Sa and Andrew McCallum for valuable comments on an earlier draft of this paper. We thank Nick Szabo for suggesting that it might be useful to do an algebraic simplification of trees for eliminating dead code before estimating their MDL complexity. We also thank the reviewers for their constructive comments.

References

[Angeline and Pollack, 1994] Peter J. Angeline and Jordan B. Pollack. Coevolving high level representations. In *The Proceedings of Artificial Life III (to appear)*, 1994.

[Angeline, 1994] Peter J. Angeline. Genetic programming and emergent intelligence. In Kim Kinnear, editor, *Advances in Genetic Programming*. MIT Press, 1994.

[Beckenbach and Bellman, 1965] Edwin F. Beckenbach and Richard Bellman. *Inequalities*. Springer Verlag, 1965.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[Hillis, 1990] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In Stephanie Forrest, editor, *Proceedings of the Ninth Annual International Conference of the Center for Nonlinear Studies on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks*, pages 228–234. North Holland, 1990.

[Hinton and Nowlan, 1987] Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. *Complex Systems*, pages 495–502, 1987.

[Holland, 1992] John H. Holland. *Adaptation in Natural and Artificial Systems, An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1992.

[Iba *et al.*, 1994] Hitoshi Iba, Hugo de Garis, and Taisuke Sato. Genetic programming using a minimum description length principle. In Kim Kinnear, editor, *Advances in Genetic Programming*. MIT Press, 1994.

[Kinnear, 1994] Kim Kinnear. Alternatives in automatic function definition. In Kim Kinnear, editor, *Advances in Genetic Programming*. MIT Press, 1994.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts, 1992.

[Koza, 1994] John R. Koza. *Genetic Programming II*. MIT Press, Cambridge, Massachusetts, 1994.

[Li and Vitanyi, 1993] Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, 1993.

[Michalski, 1983] Ryszard S. Michalski. A theory and methodology of inductive learning. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach*, pages 83–129. Morgan Kaufmann, 1983.

[Quinlan and Rivest, 1989] J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, pages 227–248, 1989.

[Rosca and Ballard, 1994] Justinian P. Rosca and Dana H. Ballard. *Learning by Adapting Representations in Genetic Programming*. To appear in the proceedings of the IEEE World Congress on Computational Intelligence, Orlando, Florida, 1994.

[Rosca and Ballard, February 1994] Justinian P. Rosca and Dana H. Ballard. Genetic programming with adaptive representations. Technical Report 489, University of Rochester, Computer Science Department, February 1994.

[Simon, 1973] Herbert A. Simon. The organization of complex systems. In G. Braziller Howard H. Pattee, editor, *Hierarchy Theory; The Challenge of Complex Systems*, pages 3–27. New York, 1973.

[Tackett, 1993] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, Inc, 1993.