Causality in Genetic Programming

Justinian P. Rosca, Dana H. Ballard
Computer Science Department
University of Rochester
Rochester NY 14627

Abstract

Causality relates changes in the structure of an object with the effects of such changes, that is changes in the properties or behavior of the object. This paper analyzes the concept of causality in Genetic Programming (GP) and suggests how it can be used in adapting control parameters for speeding up GP search. We first analyze the effects of crossover to show the weak causality of the GP representation and operators. Hierarchical GP approaches based on the discovery and evolution of functions amplify this phenomenon. However, selection gradually retains strongly causal changes. Causality is correlated to search space exploitation and is discussed in the context of the exploration-exploitation tradeoff. The results described argue for a bottom-up GP evolutionary thesis. Finally, new developments based on the idea of GP architecture evolution (Koza, 1994a) are discussed from the causality perspective.

Proceedings of the Fifth International Conference (ICGA95)

Morgan Kaufmann, San Francisco, CA.

Causality 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

Causality relates changes in the structure of an object with the effects of such changes, that is changes in the properties or behavior of the object. This paper analyzes the concept of causality in Genetic Programming (GP) and suggests how it can be used in adapting control parameters for speeding up GP search. We first analyze the effects of crossover to show the weak causality of the GP representation and operators. Hierarchical GP approaches based on the discovery and evolution of functions amplify this phenomenon. However, selection gradually retains strongly causal changes. Causality is correlated to search space exploitation and is discussed in the context of the exploration-exploitation tradeoff. The results described argue for a bottom-up GP evolutionary thesis. Finally, new developments based on the idea of GP architecture evolution (Koza, 1994a) are discussed from the causality perspective.

1 INTRODUCTION

In optimization problems, small changes occur more often than larger ones. The principle of *strong causality* states that small alterations in the underlying structure of an object, or small departures from the cause determine small changes of the object's behavior, or small changes of the effects, respectively (Rechenberg, 1994). This principle is used in this form in applications of evolution strategies (ES) (surveyed in (Bäck and Schwefel, 1993)) to problems of structure evolution (Lohmann, 1992a; Lohmann, 1992b). Structure evolution is performed by letting approximately similar encodings of structures compete for existence within a population. The parameters of each structure are adjusted by an ES algorithm at a lower level of detail. Modification of structures is performed using ES genetic operators (Lohmann, 1992a). Lohmann outlined the necessity that the algorithm observe the principle of strong causality. Without obeying this principle, all the parameters of an already evolved structure would have to be thrown away, and no continuity would be ensured in the evolutionary process.

The focus of this paper is genetic programming (GP) causality, concentrating primarily on GP approaches based on discovery and evolution of functions through the crossover operator.

Crossover has been difficult to analyze. Various simplifying assumptions are often made, which limit the applicability of the results. However such results suggest heuristic sources of improvement of the algorithms. For example, (Rabinovich et al., 1992) prove that symmetric quadratic dynamical systems approach a fixed point and also prove various properties of such systems. They apply this theory to a GA for finding a large matching in a graph. In this application they suggest that the interaction between crossover and selection is easier to analyze if crossover is applied enough times for the system to converge to a near-stationary population.

The traditional analysis of GAs by Holland (Holland, 1992) focuses on the propagation of schemata from one generation to the next. The building block hypothesis (Holland, 1992; Goldberg, 1989) outlines the importance of small schemata, called building blocks, in the proper functioning of a GA. More recently, crossover has been considered the differentiating feature that gives a GA advantages over other stochastic methods in certain types of problems. For example (Eshelman and Schaffer, 1993) bring evidence that crossover with pair-wise mating helps propagating middle order building blocks.

A GP analogy along the lines of GA schemata theory and GA building block hypothesis has been attempted in (O'Reilly and Oppacher, 1994) by generalizing the definition of a GP schema from (Koza, 1992) to a collection of tree fragments, that is a collection of trees possibly having subtrees removed. An individual instantiates a schema in case it "covers" (matches) all the fragments, overlapping between fragments not being allowed. The probability of disruption by crossover is estimated based on these definitions and is used to analyze GP schema processing. The authors conclude that such an analysis is difficult and does not offer an appropriate perspective of the GP dynamics.

This paper presents a qualitative analysis of the GP dynamics. Section 2 introduces the GP weak causality idea and analyzes the location, size, and role of subtrees involved in crossover operations. Section 3 shows that causality becomes a problem in GP extensions based on discovery of functions. However, GP naturally explores structures in a bottom-up manner and overcomes this problem. Section 4 brings experimental evidence in favor of the GP bottom-up evolution hypothesis. Finally, causality is viewed from the exploration-exploitation perspective. It offers insights for better control of an evolutionary algorithm in general. In the end, several conclusions and future work ideas are outlined.

2 CAUSALITY IN STANDARD GENETIC PROGRAMMING

GP departs from the genetic algorithm paradigm by considering tree structures for representing genotypes as computer programs (Cramer, 1985; Koza, 1992). Trees are variable length structures which offer a flexible representation for creating and manipulating programs. Problem solving is formulated as a search in the space of computer programs, structures of dynamically varying size and shape. Populations of computer programs are genetically bred based on three genetic operations: reproduction, crossover and mutation. We ignore mutation because crossover can generate any changes at any program node. GP mutation is a particular case of crossover and most experiments reported in the literature do not use mutation.

Crossover in GP is the process in which two new programs result from an exchange of genetic material between two parent programs. It consists of three steps. First, two parent individuals are selected from the population. Then a *pivot* node is randomly chosen in the tree representing each parent. Finally, subtrees rooted at the two pivots are swapped.

The main question in the analysis of GP causality is what are the effects of changes that result by applying crossover.

The intuition is that most crossover operations have a harmful effect. In particular, offspring of individuals that are already partially adapted to the "environment" and already have a complex structure are more likely to have a worse fitness. This is close to the conclusions on the role of mutation in natural evolution (Wills, 1993). It is also in agreement with our intuition that a small change in a program may drastically change the program behavior. In addition there is the following simple argument. Consider a partial solution to a hypothesis formation problem obtained using standard GP and represented by a tree T. Consider that T is selected as a parent and it is possible to obtain a solution by modifying T in such a way that a certain subtree T_i is not changed. Consider also that crossover points are chosen with uniform probability over the set of m nodes of T. The probability of choosing a crossover point v that does not lie within T_i is:

$$Prob(\text{Select}(v)|v \notin T_i) = 1 - \frac{Size(T_i)}{Size(T)}$$

The bigger T_i is (and this is true in the case of a hypothetical convergence to a solution) the smaller is the probability of keeping it unchanged. The dynamics of trees shows the phenomenon of *instability* or *weak causality* of GP structures. Next we discuss four important issues that show a more complete picture of the problem.

First, from a topological point of view, where are most of the crossover changes performed? Equivalently we can ask about the expected height of crossover pivot points. In current GP practice crossover points are chosen from internal nodes with a probability of 90% and from leaf nodes with a probability of 10%. A fundamental remark is that internal crossover nodes are chosen according to a uniform probability distribution. If we additionally assume, for a rough approximation, that trees operated upon are complete binary tress we can compute the expected height of crossover pivot points $E[h]$:

$$E[h] = \frac{9}{10} \sum_i Prob(i) \cdot h(i) = \frac{9}{10}(2 - \frac{H}{2^H - 1})$$

where $Prob(i)$ is the probability of choosing node i of height $h(i)$ as a crossover point, and H is the tree height. The leaf nodes of the complete binary tree all have height 0. This result, although based on an assumption, shows that most of the changes are closer to the tree bottom. The effect can be noticed on the structure tree[1] in Figure 1, which represents a typical case.

Second, what is the influence of selecting smaller or bigger subtrees to participate in the crossover operation? We hypothesized that the crossover operator generates non-causal changes in GP. A complete answer to this question would involve an analysis of the properties of the function set. The effect of a small change can be severe in problems of symbolic regression, or less severe in problems of regression of Boolean

[1]The notion of *structure tree* was introduced in (Rosca and Ballard, 1994a) with the goal of qualitatively analyzing program transformations during evolution.

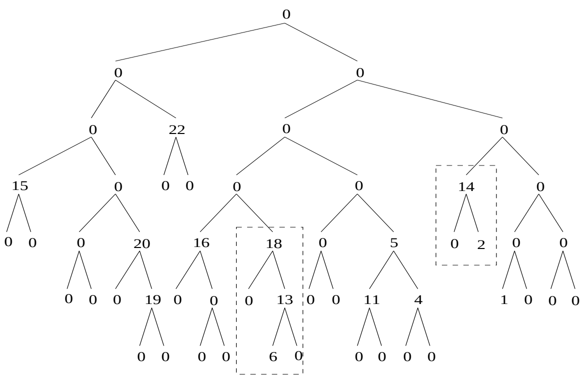

Figure 1: Structure tree for a GP solution to EVEN-5-PARITY. Nodes are labeled with the most recent generation number when the node played a pivot role in a crossover operation. Zero labeled nodes remained unchanged from the initial generation. The highlighted subtrees were not effective in the evaluation of their parents but are important in the final solution.

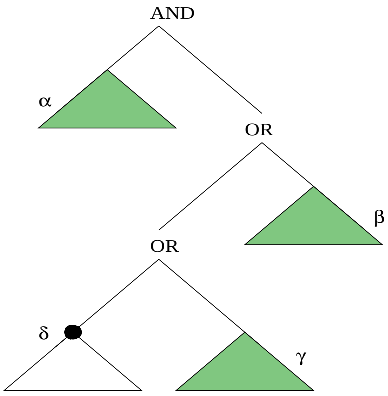

Figure 2: A small change in a Boolean tree will not necessarily determine a sharp change in the program behavior.

functions. For example, the result of the Boolean function represented in Figure 2 will be given by the result of evaluating δ on a given fitness case only if the evaluation of the following Boolean expression is true:

$$\alpha \cdot \neg\beta \cdot \neg\gamma$$

The longer the path to δ the higher will be the probability that δ plays a less important role in the overall evaluation.

Third, how much code of a parse tree representing an individual is effective? It is well known that GP evolves non-parsimonious trees if no size pressure is included in the fitness evaluation (Rosca and Ballard, 1994b), a phenomenon suggestively called “defense against crossover” (Altenberg, 1994). The increase in the size of unuseful code within population individuals would decrease the probability of disrupting useful code by means of non-causal crossover changes. However, the useless regions of code may represent reservoirs of genetic material (Angeline, 1994). They either preserve or evolve good fragments of code to be activated later during evolution as a result of crossover. This makes it difficult to estimate the size of effectively useful code. One such example is presented in Figure 1. Moreover, redundant regions of code may be created artificially, in analogy with the natural phenomenon of gene duplication, in order to evolve a better program by specializing its treatment of a subclass of inputs (Altenberg, 1994; Koza, 1994a).

Fourth, how does GP *exploit* structures? In contrast to GP crossover, the GA crossover operator is *homologous*, that is it maintains fixed positions for exchanged alleles. GP crossover is non-homologous in the sense that it does not preserve the position of the subtree on which it operates, but instead can paste a subtree at any tree level. The probability of choosing homologous crossover points in two structurally similar parents in order to transmit the parent functionality to offspring is inversely proportional to the product of the parent sizes, i.e. is very low. Moreover, if trees grow in size, this probability decreases even more and becomes negligible. This implies that even when the two parents are identical, offspring will most often have a totally different functionality, and most probably they will score less than parents. Selection favors crossover changes that recombine parts of the structure of the parents so as to improve performance, but how? In several problem domains one can observe the superposition of the parent behaviors in the offspring. In an example for the problem of finding an impulse response function (Koza, 1994b), Koza showed that crossover determines a better offspring performance by improving one parent's performance for one portion of the time domain, and inheriting the behavior of the other parent for the rest of the domain . Such a behavior has been interpreted as "case splitting": GP refines a partial solution by changing a subtree so that the program treats separately, in a more detailed way, a particular input case. In this case, structures are exploited through the function they have when computing fitness.

The four issues presented above are summarized by the following conclusions: (1) Low expected height of crossover points; (2) High functional specialization of deep subtrees; (3) Correlation of tree size and probability of non-causal changes; and (4) Functional inheritance by superposition. These arguments alleviate the non-causality problem of the GP crossover operation discussed at the beginning of this section. The issues discussed are actually under user control through the probability of selection of crossover nodes and function set properties. Later sections will suggest how fine controls, such as the probability of selection of crossover nodes, can be advantageously used in GP.

3 HIERARCHICAL GENETIC PROGRAMMING

We overview two GP extensions relevant to our goal of analyzing GP causality. Automatic definition of functions (ADF) (Koza, 1992) is based on the idea of evolution of reusable subroutines. The adaptive representation (AR) approach (Rosca and Ballard, 1994c) is based on the discovery of useful building blocks of code. These blocks are attained by analyzing the evolution trajectory, generalizing blocks of code to define new functions, and finally adapting the problem representation on-the-fly. Although ADF and AR implement the ideas of discovery, evolution and use of new functions in different ways, both actually evolve a hierarchy of functions that improves search efficiency. This paper refers to both mechanisms by hierarchical genetic programming (HGP).

3.1 AUTOMATIC DEFINITION OF FUNCTIONS

The automatic definition of functions approach (ADF-GP) assumes that parsimonious problem solutions can be specified in terms of a main program and a hierarchical collection of subroutines. The main program invokes a subset of the subroutines to perform the overall computation, while those subroutines may in turn call other subroutines computing partial results.

Genetic programming is used both to search for appropriate subroutines, and to find a way of composing discovered subroutines and primitive functions into a complete solution. In this approach each individual program has a dual structure. The structure is defined by a fixed number of components or *branches* to be evolved: the function branches and a main program branch. Each function branch (for instance ADF_0, ADF_1) has a fixed number of arguments. The main program branch (*Program-Body*) produces the result. Each branch is a piece of code built out of specific primitive terminal and function sets, and is subject to genetic operations. The set of function-defining branches, the number of arguments that each of the function possesses and the "alphabet" (function and terminal sets) of each branch define the *architecture* of a program. The references allowed between function branches determine a hierarchical organization of the set of functions. Although the number and interconnectivity of ADFs are fixed, the definitions of ADFs are evolved. Genetic operations on ADFs are syntactically constrained by the components on which they can operate. For example, crossover can only be performed between subtrees of the same type, where subtree type depends on the function and terminal symbols used in the definition of that subtree. An example of a simple typing rule for an architecturally uniform population of programs is *branch typing*. Each branch of a program is designated as having a distinct type.

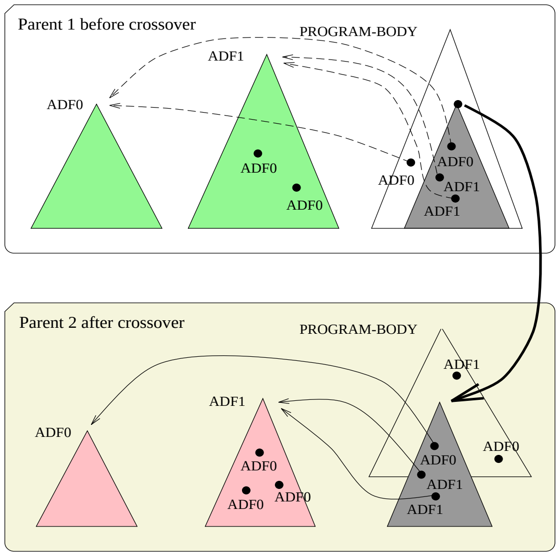

Figure 3: The non-causality of ADF-GP: Definitions for ADFs are local. Thus, a fragment of code copied from donor parent 1 into receiving parent 2 will be evaluated in the new lexical environment of parent 2.

In this case the crossover operator can only swap subtrees from analogous branches.

3.1.1 Causality in ADF-GP

The ADF approach simultaneously attacks the search problem at different structural levels. GP has to discover both the definitions for a fixed set of subfunctions, each with a predefined number of parameters, and how to combine calls to the automatically defined functions within the main body. This corresponds roughly to discovering a way to decompose the problem and solving the subproblems given only the maximum number of subproblems and the general structure of the subproblems (i.e. number of parameters and "alphabet"). Due to the imposed ordering of ADFs we can consider each ADF as a different structural level.

During GP search, modifications are alternatively made at each of the structural levels. A code fragment brought from another individual changes its function entirely if it contains calls to ADFs. For example, consider a piece of code involving calls to lower order ADFs that is pasted in a higher-order function or the main body as a result of a crossover operation, and suppose the definitions of the ADFs in the two parents are entirely different. Lexical scope dictates the definition to be used when invoking a sub-function, so that the calls to ADFs from the transplanted piece of code will refer to the definition of a totally different function from the new lexical scope.

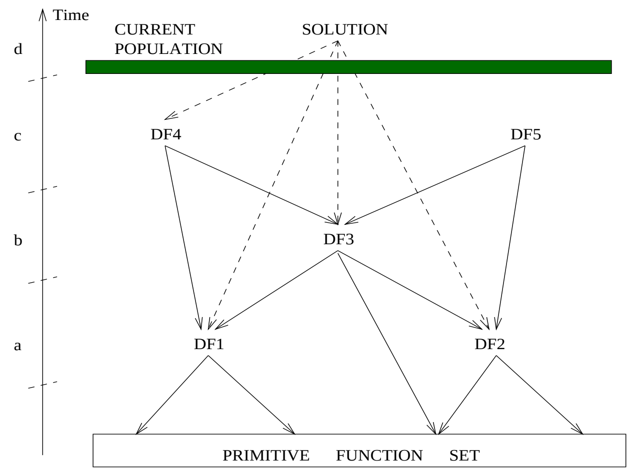

Figure 4: A hypothetical call graph of the extended function set in the AR method. The primitive function set is extended hierarchically with functions ($DF1, DF2$, etc.) discovered at generation numbers a, b, c.

This quite frequent situation is depicted in Figure 3 and demonstrates the *non-causality* of ADF-GP. The non-causality property of ADF-GP is in total opposition with the principle of *strong causality* previously stated in section 1.

It is useful to visualize how the search for a solution may generally proceed in ADF-GP. Each of the ADF functions represents a different sub-function. Consider the last modification imposed on a program tree before it becomes an acceptable solution. It is very unlikely, but not impossible that this last change has been a change with a large influence, for example a change in one of the functions at the basis of the hierarchy. This situation represents a lucky change. Most probably, though, it was a change at the highest level, in the program body. We conjecture the following general principle: As evolution progresses, changes according to the principle of strong causality become more important. In other words, as better and better individuals are found, selection most often favors small, causal changes which have the biggest chance of being successful. The effect of this principle is a stabilization on useful lower level ADFs. Evolution will freeze good subroutines and will eventually find beneficial changes at higher levels.

3.2 ADAPTIVE REPRESENTATION

In contrast to the uninformed changes in function definitions in ADF-GP, AR explicitly attempts to discover useful functions. A hierarchy of automatic functions is created in a bottom-up fashion as the problem is being solved (see Figure 4).

At the base of the function hierarchy lie the primitive functions from the initial function set. More complex functions are dynamically built on the primitive functions, and become stable components of the representation. The levels in the hierarchy are discovered by using heuristic information as conveyed by the environment and captured using heuristic block fitness functions, as a result of several steps:

1. Select candidate blocks appearing in population individuals
2. Generalize candidate blocks to create new functions
3. Extend the representation with the new functions, noticing if progress is made.

The generation intervals with no function set changes represent evolutionary *epochs*. At the beginning of each epoch, part of the population is replaced with random individuals built from the extended function set. This *extinction* step is taken in order to ensure the propagation of newly discovered functions in the population.

The discovery of functions in AR can be guided either by statistical information, like the fitness distribution of a newly generated subpopulation, or by measures of the short-term influence of the new functions on the evolutionary process. Functions that do not support progress by not being invoked by fit population individuals may be deleted. However, such domain independent methods are computationally expensive. In the experiments reported, only domain dependent building block functions have been used, which represents a limitation.

The evolution of discovered functions is not supported in AR-GP. Although the AR-GP strategy eliminates the possibility of performing highly non-causal changes, it might not work well in early stages of the algorithm when increased search space exploration is desired.

4 A BOTTOM-UP GP EVOLUTIONARY THEORY

In order to test the hypothesis that causality plays an increasing role as evolved individuals become more complex and fit we have studied the most recent part of the genealogy tree for EVEN-n-PARITY parity problems[2]. This was done by giving each individual a *birth certificate* that specifies its parents and the method of birth that corresponds to the branch type where crossover is performed (one of ADF0 crossover, ADF1 crossover, main program body crossover, or reproduction). We hoped that an analysis of the birth

[2]The parity problem definition for a GP implementation is presented in (Koza, 1992; Rosca and Ballard, 1994b).

certificates starting with the final solution and tracing backwards its origin, would shed light on the GP dynamics, as hypothesized by the general principle in section 3.1.1. In order to determine the effect of the different types of birth operations, we have computed a temporally discounted frequency factor bcf for a given solution tree T and a type of birth:

$$bcf(T, type, d) = \left(\frac{1-\gamma}{1-\gamma^d}\right) \sum_{i=0}^{k_T} \chi_{\{type\}}(i) \cdot \gamma^{depth(T_i)}$$

where k_T is the number of programs in the genealogy tree of T down to a depth d, and $\chi_{\{type\}}(T_i)$ is the characteristic function of ancestor T_i of T, returning 1 if T_i has a birth certificate of birth-type *type* and 0 otherwise.

Table 1 presents the results for several successful runs of ADF-GP for EVEN-5-PARITY, with two ADFs and three arguments each. These results show that ADF-GP search relies in most cases on changes at higher and higher structural levels which make it possible to exploit good code fragments that appear in the population.

Table 1: Statistics of birth certificates in successful runs of EVEN-5-PARITY using ADF-GP with a zero mutation rate and a population size of 4000. Each certificate of a given type counts one unit and is temporally discounted with a discount factor $\gamma = 0.8$ based on its age. Only certificates at most 8 generations old have been considered.

GP Run	Programs Explored	Birth Certificate Freq.			Final Gen.
		ADF0	ADF1	Body	
1	123,009	0.295	0.0	0.704	32
2	110,892	0.221	0.472	0.416	32
3	62,699	0.077	0.526	0.397	17
4	35,162	0.447	0.102	0.451	9
5	55,748	0.1	0.214	0.685	15
6	55,438	0.093	0.202	0.704	15

The above numerical results have taken into account only a small time window compared to the entire number of generations. A more detailed picture of the importance of various types of crossover during the entire GP evolution is based on a complete analysis of birth certificates, starting with the initial generation. Such an analysis is depicted in Figure 5 for a typical case. The overlapping distribution trends of birth certificates suggest both the overall importance of a birth certificate type as well as its trend over the entire evolution period, from generation 0 till the solution is found.

The stabilization of changes in the hierarchy occurs bottom-up. Crossover changes in automatically discovered functions are highly non-causal and are performed in early generations, unlike changes in the main program body, mostly changes at low tree heights, that are performed in later generations. The results presented confirm that, as the population evolves, increasingly causal changes become more important and are selected.

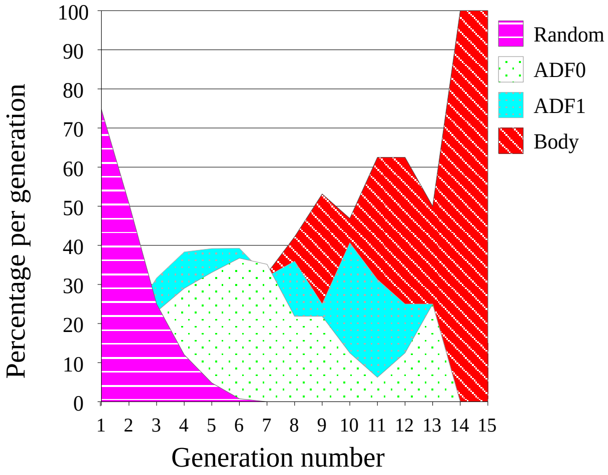

Figure 5: Variation of the percentage of crossover types over generations, while looking for a solution to EVEN-5-PARITY that was found in generation 15. *Random* indicates the propagation of random individuals from the initial population due to reproduction.

5 CAUSALITY AND SEARCH SPACE EXPLOITATION

The choice of the search strategy to pursue exemplifies a well-known problem, the exploration-exploitation trade-off (Epstein, 1967; Holland, 1992). In the reinforcement learning (RL) literature, this tradeoff is made explicit by the control structure of the algorithms (Kaelbling, 1993).

If the search process chooses the action prescribed by its current knowledge, then the system *exploits* its current knowledge by acting in order to gain reward. It occasionally explores actions at random in order to experiment with more state-action pairs and eventually improve its policy. If the search process acts to gain information or experience then it *explores* the search space. In RL, the system must have an explicit policy of balancing exploration and exploitation.

In contrast GP is a search technique that implicitly balances exploration and exploitation. Discovery and evolution of functions amplifies the exploratory ability of the GP search process (Rosca, 1995). However, as the best-of-generation program fitness increases, the probability of falling upon good individuals by exploration decreases substantially. The GP search process should exploit the structures already discovered and avoid spending unnecessary effort with state space exploration. This corresponds to the idea that genetic operations should become more and more causal towards later stages of evolution. Causality can be obviously correlated with search space exploitation (see Table 2). This suggests that GP is able to dynamically balance exploration and exploitation. This tradeoff can also be adaptively controlled by modifying the rate at which causal or non-causal genetic operations are applied so that GP spend its search effort in a more efficient way. Weakly causal changes are desirable for search space exploration initially, while strongly causal changes are desirable for increased exploitation later in the process. The tradeoff may be controlled, for instance, by the relative average fitness increases.

Table 2: Correlation between causality and exploitative ability in GP search.

Time (Generation)	**Early**	**Advanced**
Crossover changes	Non-causal	Causal
Exploration	High	Low
Exploitation	Low	High

In the following we discuss recent improvements of HGP algorithms from the perspective of this tradeoff. The architecture of an application in ADF-GP is defined by the number of function branches, number of arguments for each branch and the function and terminal primitives for each function branch. Work on the evolution of architecture has been motivated mainly by the problem of determining the appropriate architectural choices in ADF-GP. The architecture itself can be evolutionarily selected in case the initial population is architecturally diverse and care is taken when crossing over individuals having different architectures (Koza, 1994b).

Koza introduced six new genetic operations for varying the architecture of an individual program: branch duplication, argument duplication, branch deletion, argument deletion, branch creation and argument creation (Koza, 1994a). All operations of duplication respect the principle of strong causality discussed before. They are performed such that they preserve the behavior of the resulting programs. They merely increase the potential of refinement of the programs and thus they resemble the process of gene duplication in natural evolution. The duplication of a program architecture element (branch or argument) is done in conjunction with a random replacement of the invocations of the corresponding element in the duplicated copy. Such an operation not only respects the principle of strong causality while allowing for future behavior improving changes but it also decreases the probability that a future random change will drastically change the behavior of the program. A similar conclusion can be drawn for creation operations. The deletion operations do not possess the nice properties mentioned above. They have the antagonist role of confining the increase in size of the evolved programs.

The emergent structure in ADF-GP as an effect of causality is an explicit policy in AR-GP. Future work aims at an improved version of AR-GP that will combine competition between individuals in a population of programs with cooperation between individuals in a populations of evolvable functions. This extension will not require explicit block fitness functions, but will allow reinforcement signals based on the global individual behavior to be used to evaluate functions. Block fitness functions may only be used if supplementary domain knowledge is available. ADF-GP is based on a "blind" competition between individuals. In contrast, in this version of AR-GP the population of functions evolved can be viewed as an environment which allows cooperation between its elements while partial solutions compete with one another. Functions beneficial to some individuals could be invoked also by other individuals. The end result is that the collection of functions acquires the necessary structure for solving the problem, taking the form of a hierarchy of functions as in Figure 4. The population of functions evolves by using both exploitative and exploratory operators (Rosca, 1995).

6 CONCLUSIONS

The paper analyzes the causality of the crossover operator in GP and suggests how search control parameters can be adapted for speeding up GP search. Standard GP presents a characteristic instability or weak causality of the structures evolved, which can be varied by changing the probability of selecting crossover points. This effect is amplified in GP with function discovery. Weak causality has been discussed related to the exploration-exploitation tradeoff in search problems.

The paper brings arguments for a bottom-up evolutionary thesis of hierarchical GP. Early stages of evolution in GP usually discover stable components (Simon, 1973). Replaying backwards the genealogy tree that resulted in a problem solution shows that most changes in later stages of evolution are performed at the higher hierarchical levels. This suggests that in the unconstrained HGP approach such as ADF-GP there are implicit constraints. Early in the process the changes are focused towards the evolution of more primitive functions. Later in the process the changes are focused towards the evolution of program control structures. This also suggests that in the later stages of evolution of an HGP algorithm exploration is automatically traded for exploitation.

A hierarchy of functions amplifies both good and bad genetic changes, making selection more efficient. The bottom-up evolution thesis justifies the search for building blocks and the expansion of the problem representation used in AR-GP.

AR-GP had a specific search control structure, that allowed it to explore causality. ADFs suggest the tradeoffs in the search control parameters. One could use these insights to come up with a more refined AR-GP-like system, that involves automatic adaptation of the control parameters. This remains a goal for future research. Without adaptation mechanisms artificial intelligence systems are "inherently biased by the features and representation schemes defined by the system designer" (Kitano, 1993).

Acknowledgments

We thank Peter Angeline for valuable suggestions made on earlier drafts of this paper. This research was sponsored by the National Science Foundation under grant numbered IRI-9406481 and by DARPA research grant numbered MDA972-92-J-1012.

References

Altenberg, L. (1994). The evolution of evolvability in genetic programming. In Kinnear, K., editor, *Advances in Genetic Programming*. MIT Press.

Angeline, P. J. (1994). *Evolutionary Algorithms and Emergent Intelligence*. PhD thesis, Computer Science Department, Ohio State University.

Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In *Proceedings of the First International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, Inc.

Epstein, R. (1967). *The Theory of Games and Statistical Logic*. Academic Press, New York.

Eshelman, L. J. and Schaffer, J. D. (1993). Crossover's niche. In Forrest, S., editor, *Proceedings of the International Conference on Genetic Algorithms*, pages 9–14. Morgan Kaufmann Publishers, Inc.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Holland, J. H. (2nd edition, 1992). *Adaptation in Natural and Artificial Systems, An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT Press.

Kitano, H. (1993). Challanges of massive parallelism. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 813–834. Morgan Kaufmann Publishers, Inc.

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

Koza, J. R. (1994a). Architecture-altering operations for evolving the architecture of a multipart program in genetic programming. Computer Science Department STAN-CS-TR-94-1528, Stanford University.

Koza, J. R. (1994b). *Genetic Programming II*. MIT Press.

Lohmann, R. (1992a). Structure evolution and incomplete induction. In *Parallel Problem Solving from Nature 2*, pages 175–185. Elsevier Science Publishers.

Lohmann, R. (1992b). Structure evolution and neural systems. In *Dynamic, Genetic and Chaotic Programming*, pages 395–411. John Wiley & Sons, Inc.

O'Reilly, U.-M. and Oppacher, F. (1994). The troubling aspects of a building block hypothesis for genetic programming. In *Proceedings of the Third Workshop on Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers, Inc.

Rabinovich, Y., Sinclasir, A., and Widgerson, A. (1992). Quadratic dynamical systems. In *Proceedings of FOCS*, volume 33, pages 304–313. IEEE Press.

Rechenberg, I. (1994). Evolution strategy. In Zurada, J. M., Marks-II, R. J., and Robinson, C. J., editors, *Computational Intelligence - Imitating Life*, pages 147–159. IEEE Press.

Rosca, J. P. (1995). Genetic programming exploratory power and the discovery of functions. In *Proceedings of the Fourth National Conference on Evolutionary Programming*. MIT Press.

Rosca, J. P. and Ballard, D. H. (1994a). Genetic programming with adaptive representations. Technical Report 489, University of Rochester, Computer Science Department.

Rosca, J. P. and Ballard, D. H. (1994b). Hierarchical self-organization in genetic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 251–258. Morgan Kaufmann Publishers, Inc.

Rosca, J. P. and Ballard, D. H. (Orlando, 1994c). Learning by adapting representations in genetic programming. In *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 407–412. IEEE Press.

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

Wills, C. (1993). *The Runaway Brain*. BasicBooks.