A Controlled Genetic Programming Approach for the Deceptive Domain

Emin Erkan Korkmaz
Department of Computer Engineering
Middle East Technical University
Ankara-Turkey
korkmaz@ceng.metu.edu.tr
$+(90) - 312 - 210 - 5536$
Göktürk Üçoluk
Department of Computer Engineering
Middle East Technical University
Ankara-Turkey
ucoluk@ceng.metu.edu.tr
$+(90) - 312 - 210 - 5584$

***Abstract*— Traditional Genetic Programming randomly combines subtrees by applying crossover. There is a growing interest in methods that can control such recombination operations in order to achieve faster convergence. In this study, a new approach is presented for guiding the recombination process for *Genetic Programming*. The method is based on extracting the *global information* of the promising solutions that appear during the genetic search. The aim is to use this information to control the crossover operation afterwards. A separate control module is used to process the collected information. This module guides the search process by sending feedback to the genetic engine about the consequences of possible recombination alternatives.**

I. Introduction

It is clear that the random recombination used in traditional GP can easily disturb the building blocks. An attempt based on determining the beneficial building blocks and preventing them from disturbance during the recombination operations can be helpful. However, for the deceptive class of problems such an approach is questionable. The interaction between the partial solutions is high for these problems. In other words the contribution of a subpart of a chromosome to the overall fitness depends on the configuration of other parts. The global meaning of finding a possible solution goes beyond determining isolated, non-interacting building blocks and bringing them together.

In this study, a genetic search system which can deduce beneficial knowledge from its own experience is designed. A different approach which focuses on a new kind of abstraction on the structure of the chromosomes, is presented. A new representation is used for this scheme. The proposed representation reflects the characteristics which are important in denoting the overall organization of a chromosome. The information obtained by using the new representation is named as *Global Information*. The aim is to increase the performance of GP in the deceptive domain, by extracting the knowledge of what it is to be good globally and performing the right crossover operations which would keep the search among the localization of well-fit elements afterwards.

The proposed method has been applied to two different real-world domains namely the *Context-Free Grammar Induction* and the *N-Parity Problem*. Both of these domains can be considered highly deceptive. Traditional GP has exhibited quite a low performance for both problems. Furthermore, a tunable benchmark problem is defined and our approach is tested on different instances of this problem.

In the following section an overview of various approaches in the area are given. In section III our approach is presented in detail. In section IV the application of our approach to CFG induction is given. Then, in section V, the N-Parity problem is analyzed in the light of our approach. In section VI, the results obtained on the benchmark problem are presented. In section VII, statistical tests are presented which verify the performance increase obtained. Conclusions and discussions are presented in the last section.

II. Related Work

Various techniques have been proposed in order to increase the performance of GP. But a significant group of researchers focuses on controlling and guiding the search process in GP. Recombination in traditional GP is random and therefore researchers have been mainly interested in controlling this operation. The aim is to perform more intelligent recombination that would increase performance.

For instance [7] proposes a method called *Recombinative Guidance for GP*. The method is based on calculating the performance values for subtrees of a GP tree during evolution and then applying recombination operators so that the subtrees with high performance are not disturbed. On the other hand [23] uses a knowledge repository which is expected to guide the search towards better solutions. The knowledge repository collects code segments from the genetic population together with some associated information like fitness, number of occurrences, depth and so on. [23] proposes a method to calculate a single score for each segment that would reflect its overall contribution to the current task. The evolution proceeds by adding new code segments with high performance to the knowledge repository and excluding the ones which are subject to performance loss.

It is possible to find other studies where the aim is to control recombination. [22] uses a context free grammar to control crossover and mutation and [4] proposes a method which tries to preserve the context in which subtrees appear in the parent trees before the crossover operation.

Similar approaches trying to control recombination operators can be found in the area of Genetic Algorithms, too [2], [15], [18].

The attempts presented above are usually based on determining the important building blocks and preventing them from being disturbed by the recombination operations. However, [17] states that for some functions, even if it is possible to decompose the function into some components, the subcomponents could interact. In such a case it becomes impossible to consider each subfunction independently, optimize it and then obtain the optimum by combining the partial solutions. For this set of problems it is clear that an attempt based on determining building blocks is not expected to increase performance significantly.

III. Extracting the Global Information

A. General Framework

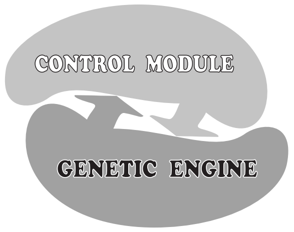

Fig. 1. The dual structure proposed.

The method that will be used for control of the crossover operation is based on the *global information* of the chromosomes. In order to process this information, we have designed a new module called the *Control Module*. Figure 1 displays the dual structure of our system. The genetic engine, which can be considered as the base structure, performs the standard genetic search. The control module, as a super structure, keeps an eye on the search carried out by the genetic engine. It focuses on the global information of the chromosomes and performs meta-level learning at certain periods to determine what is good globally. Once the first learning process has taken place, the control module starts sending feedback to the genetic engine about the consequences of possible crossover operations.

It is considered that the frequency of the elements used and the knowledge of how they are distributed in the chromosome might contribute to the global picture of the structure at hand. It can be claimed that these two forms of information are quite critical in terms of forming the global solution. What is more, traditional GP is not capable of analyzing such information. The *frequency* information is important, since using an element more or less than a certain number of times might be critical in terms of building the global solution. The *position of the elements* on the tree is an another critical factor in terms of the solution. The contribution of an element in the tree might depend on the distribution of the other elements. So, it can be claimed that by using these two forms of information it becomes possible to analyze the dependencies that might exist in different parts of the solution.

The learning will be carried out on feature vectors that reflect this global information about chromosomes (trees). The feature vectors are obtained by a mapping process that determines what is considered as global to a tree. Before going into the details of the mapping, it is worth taking a closer look at the importance of such global information processing in the GP search.

In a genetic search, each individual (structure) in the population receives a measure of its fitness based on a defined objective function. Selection mechanism focuses on high fitness individuals in order to exploit fitness information. Recombination and mutation operations perturb those individuals in order to provide general heuristic for exploration. However, in order to achieve an efficient exploitation, the search space should be continuous. That is to say, chromosomes similar in terms of shape and size should have (mostly) similar fitness values. In such a case, it becomes easier to proceed towards better solutions throughout the crossover operation. However, for the deceptive class of problems the picture is quite different. Here, the interdependency among the subparts of a chromosome is the most important aspect. This property makes the fitness of recombined chromosomes fragile during crossover. A small change on the chromosome might be hazardous or beneficial in terms of the global dependencies that should exist in the chromosome. Hence, the fitness value can change dramatically during recombination. If we could use a function to denote the similarity of chromosomes, we could observe the following situation for the majority of the chromosomes in the search space.

$$Structually\_Similar(C_1, C_2) \Longleftrightarrow Fitness\_Alike(C_1, C_2), \quad (1)$$

where C_1 and C_2 are two non equal chromosomes. $Structually\_Similar$ stands for a boolean function that would determine if two given chromosomes are similar in terms of shape and size. $Fitness\_Alike$ is a boolean function that would check if the fitnesses of two chromosomes are alike or not. With an increasing number of chromosomes, that does not preserve the mentioned regularity, it becomes more difficult to obtain an efficient genetic search. On the other side, even for deceptive problems, some similar chromosomes would still demonstrate regularity in terms of their fitness values. However, the similarity in terms of shape and size is not sufficient condition for such regularity in the deceptive domain. Hence, there should be other aspects to be considered in order to achieve alike fitness values. The situation can be described as follows

$$Structually\_Similar(C_1, C_2) \wedge \psi(C_1, C_2) \Longrightarrow Fitness\_Alike(C_1, C_2). \tag{2}$$

In this implication $\psi(C_1, C_2)$ represents the function that denotes the missing aspects, which were not taken into consideration by the standard procedure. The main proposal of this study is that an approximation for these extra aspects can be formulated by focusing on the global structure of the chromosomes. The necessary information is extracted from the chromosomes by a mapping process from the set of chromosomes to a set of feature vectors.

In general, this mapping can be defined as

$$f : \mathcal{C} \mapsto \mathcal{F}, \tag{3}$$

where $\mathcal{C}$ is the set of chromosomes and $\mathcal{F} = \mathcal{X}_1 \otimes \mathcal{X}_2 \otimes ... \otimes \mathcal{X}_n$. Here, $\mathcal{X}_i \in \{\mathcal{Z}, \mathcal{Q}, \mathcal{R}, \mathcal{E}_i\}$, where $\mathcal{Z}$ is the set of integers, $\mathcal{Q}$ is the set of rational numbers, $\mathcal{R}$ is the set of real numbers and $\mathcal{E}_i$ is an ordered set of features.

Then, by using a learning algorithm, the aim is to induce a boolean function φ which would perform the following mapping.

$$\varphi : \mathcal{F} \mapsto \{True, False\}. \tag{4}$$

The learning algorithm would use a training set consisting of elements in $\mathcal{F}$ and the induced function φ would divide the space into two parts. Points having the value $True$ would correspond to chromosomes with high fitness values and the ones with value $False$ to low fitness elements. Hence we can write

$$\psi(C_1, C_2) \overset{\triangle}{=} \left[\varphi(f(\mathcal{C}_1)) \overset{?}{=} \varphi(f(\mathcal{C}_2))\right] \tag{5}$$

where ψ is the boolean function in Formula 2. The above formula denotes that if the points corresponding to C_1 and C_2 are in the same region of $\mathcal{F}$ according to the induced function φ, then we can conclude that the other condition needed to achieve the fitness alikeness is satisfied. This is general framework which can be used to increase the performance of the genetic search especially in a deceptive domain.

The critical decision about the formalization is the choice of the mapping function f. Different alternatives exist depending on the structure of the chromosomes used or the feature set aimed to be extracted throughout the process.

To define this mapping, we introduce a new similarity measure on the chromosomes, namely (statistical-spatial) $SS\_Similarity$. As the name implies, this new similarity measure is based on the statistical and spatial characteristics of the chromosomes. These characteristics are defined to be the collection of the frequency and positioning information of the terminal-functional elements that appear in the chromosome. In this domain, $SS\_Similarity$ will serve as the the function ψ mentioned in implication 2 of this domain. For the GP domain of interest, the claim of this study turns out to be

$$Structually\_Similar(C_1, C_2) \wedge SS\_Similar(C_1, C_2) \Longrightarrow Fitness\_Alike(C_1, C_2). \tag{6}$$

By this implication, it is claimed that, for the deceptive problems, if two chromosomes are similar in terms of their statistical and spatial characteristics in addition to their shapes and sizes, than it is possible to expect an alikeness between their fitness values too. We will experimentally prove this claim in several testbeds. Note that the crossover operation itself takes care of $Structually\_Similar(C_1, C_2)$ (based on the size and shape of a chromosome). So, an extra testing mechanism is needed to clarify if the $SS\_Similarity$ is destroyed or not during the genetic process. The mapping process used to extract the statistical and spatial characteristics of the chromosomes is defined in the following paragraphs.

As the chromosome of GP is a structured tree which can be defined as a tuple $\mathcal{T}(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of edges with some special constraints. We propose the following choice for f and will call this choice $\hat{f}$.

$$\hat{f} : \mathcal{T}(\mathcal{V}, \mathcal{E}) \mapsto \mathcal{Q}^n. \tag{7}$$

Note that the vector space $\mathcal{F}$ defined in Formula 3 has been chosen as Q^n for this study. The dimension of this space denoted by n is set as

$$n = \mid Terminal\_Set \mid + \mid Function\_Set \mid, \tag{8}$$

where $Terminal\_Set$ and $Function\_Set$ are the two sets consisting of the terminal and non-terminal elements used for the GP search. Given a tree $T(V, E)$, if

$$\left[\hat{f}(T(V, E))\right]_i = x_i \tag{9}$$

where $i = 1...n$ and $x_i \in \mathcal{Q}$, then there exists a terminal or a function element in tree T which would have the frequency and position information denoted by x_i. The integer part of x_i specifies how many times this element is used in the tree, namely the frequency of that element.

On the other hand, the fractional part of x_i holds the position information of the element. This is defined as the sum of the depth values of all occurrences of the element in the tree. This depth summation is transformed into a fractional value by using a multiplicative constant.

The presented $\hat{f}$ has a simple formalization about the global organization of the tree and does not have a heavy computational load. The following can be mentioned about $\hat{f}$.

- Each terminal and function element is mapped to a base vector.
- By using a bottom up construction, it is possible to obtain a single vector for the whole GP-tree.

A leaf node is only mapped to its base vector while the vector for an internal node is obtained by adding the vectors of its children plus the base vector corresponding to it. By this procedure, the frequency information of each element is held in a different component of the formed vector, as of the form of an integer. The depth information of the same element is stored as the fractional value of that rational.

Mathematically speaking, let $P(C_1, C_2, ..., C_k)$ be a subpart of a chromosome, where P is an internal node and C_i is a child node connected to this parent. The vector that would correspond to P can be obtained using the following formula.

$$\mathbf{V}_P = \mathbf{V}_{P_{base}}[1 + c_0 \cdot depth(P)] + \sum_{i=1}^{k} \mathbf{V}_{C_i}, \qquad (10)$$

where $\mathbf{V}_{C_i}$ is the vector corresponding to child C_i and c_0 is the constant used to transform the depth summation into a fractional value. Here, $c_0 \ll 1$ and the constant has to be chosen such that

$$c_0 \cdot \left[\underset{l \in TF}{\text{MAX}} \left(\sum_{\forall node \ni Label(node)=l} depth(node) \right) \right] \leq 1, \qquad (11)$$

where $TF = Terminal\_Set \cup Function\_Set$. The inequality states that the multiplication of c_o with the maximum possible depth summation should stay well below 1 so that the position information would not interfere with the frequency information.

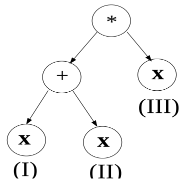

Fig. 2. A sample chromosome.

As an example, consider the tree in figure 2. The function and terminal sets are $F = \{+, -, *, /\}$ and $T = \{X\}$. The base vectors would be:

- $\mathbf{V}_+ = [0, 0, 0, 0, 1]$
- $\mathbf{V}_- = [0, 0, 0, 1, 0]$
- $\mathbf{V}_/ = [0, 0, 1, 0, 0]$
- $\mathbf{V}_* = [0, 1, 0, 0, 0]$.
- $\mathbf{V}_x = [1, 0, 0, 0, 0]$.

The dimension of the vectors, namely 5, is determined as the total number of function and terminal elements: $\mid F \mid + \mid T \mid$.

For the tree in Figure 2, the vector construction mechanism will be as follows. The base vectors are as specified above. The three different occurrences of the terminal element X are labeled as $(I), (II)$ and (III). Since $X(I)$ and $X(II)$ have the same depth value, their vectors will be the same. This vector would be $[1.02, 0, 0, 0, 0]$. The vector corresponding to $X(III)$ will be $[1.01, 0, 0, 0, 0]$ due to the depth value of 1. According to Equation 10, the vector of $'+'$ will be $[2.04, 0, 0, 0, 1.01]$. Finally, the vector corresponding to $'*'$ which would be the vector of the whole tree, can be obtained by using the vectors of $'+'$ and $X(III)$ this time. Hence, $V_* = [3.05, 1, 0, 0, 1.01]$. Note that, each dimension of this vector provides information about the usage of a terminal or a function element. For instance the first dimension is reserved for the terminal element X. The value in this dimension denotes that the terminal element occurs three times and the sum of the depths of these three different occurrences is five. On the other side, the second dimension denotes that $'*'$ operation is only used once as the root of the tree.

The constant value that is used to transform the depth value into a fractional one is 0.01. However, if the sum of the depths exceeds 100, depth and frequency information will interfere with each other. If this is possible, a smaller constant has to be used. At least, it should be guaranteed that the number of such ill formed vectors is kept small enough that the learning process is not affected.

Also note that different chromosomes can be mapped to the same vector. However, this does not contradict our assumption since different elements in the base structure could be similar in terms of the super structure.

B. *Implementation of the idea*

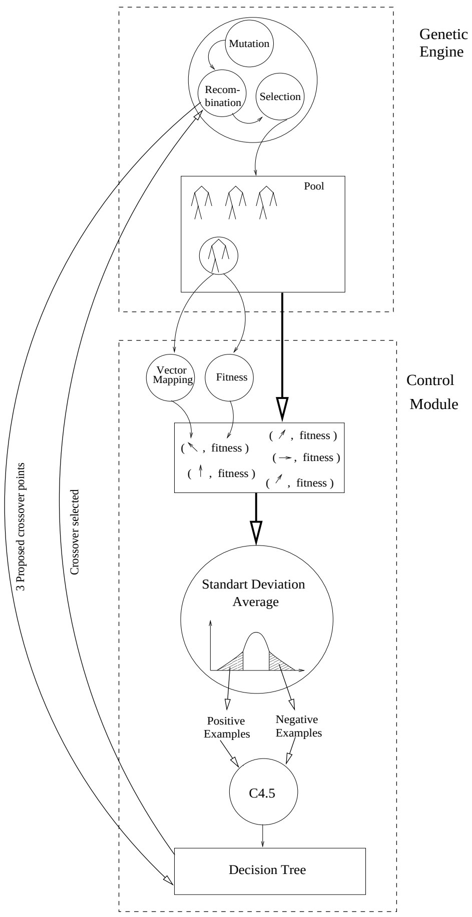

Fig. 3. Interaction between the Control Module and the Genetic search.

The interaction between the genetic engine and the control module is as follows. For each chromosome in the population, the corresponding vector is formed and sent to the control module together with its fitness value. The control module collects the vectors and fitness values for a certain period of generations, which we call the *learning period*. Then the average and the standard deviations of the fitness values are calculated.

The control module forms the training set using those elements whose fitness values deviate from the average more than the standard deviation. Those with positive deviations are marked as positive examples and the others as negative. The *“C4.5, Decision Tree Generator”* [20] is used to generate the abstraction over the training set. Then, for each crossover operation to be performed, the genetic engine sends to the control module three different alternative crossover points. The control module predicts if the alternative offsprings will be in the positive or the negative class by using the decision tree generated by *C4.5*. The best alternative is chosen by the genetic engine and the learning process is repeated periodically. The best case would be finding out crossover points which can produce two positive-class offsprings. If this cannot be achieved, certainly the best alternative would turn out to be the operation which produces one positive-class offspring. Lastly, if no positive class offspring could be achieved by all of the alternatives, crossover points are determined randomly. Also, note that the use of three alternatives is an empirical choice.

IV. $TESTBED_1$: CONTEXT FREE GRAMMAR INDUCTION

Natural language sentences have been used in order to form the training set for the CFG-induction problem. The training set consists of 21 positive examples and 17 negative examples. The sentences formalize a subset of English including sentences consisting of structures like NP, VP and PP. The Noun phrase (NP) is quite simple and consists of a determiner (D) followed by a noun (N) or compound noun. On the other hand, the verb phrase (VP) can be intransitive, transitive or ditransitive and the prepositional phrase (PP) could be attached to a VP or to an NP. The aim is to induce a CFG that can parse the positive examples and reject the negative ones. Each chromosome in the population is a candidate grammar and the details of this representation can be found in [11].

The problem can be considered as a highly deceptive one. It is possible to divide a grammar into subparts like NP, VP or PP, however these subparts do not have clear borders. Overlapping exists due to the fact that NP can be a part of VP and PP.

The fitness function used is the standard one. For grammar G, if S is the set of sentences consisting of the positive examples that G cannot parse and the negative examples that G parses, then the fitness of G is defined as:

$$F(G) = \sum_{S_i \in S} SENTENCELENGTH(S_i) \qquad (12)$$

So the aim is to minimize the fitness function. For the test data used, the worst fitness for a grammar could be 243 which is the sum of the length of all sentences both in the positive and the negative set. And the best fitness is, of course, zero which can be achieved when a grammar parses all of the examples in the positive set and rejects all of the negative set.

The grammar evolved is subject to only one restriction. The number of right hand side elements in a grammar can be at most two. The terminal and function sets are $T = \{D, N, V, P\}$ and $F = \{F_1, F_2, ..., F_{10}\}$, where F_i can be used as a variable in an evolved grammar and the terminal elements denote the syntactic categories of the words used in the training set, namely determiner (D), noun (N), verb (V) and preposition (P). Considering the restriction specified above each element of the function set can have one or two arguments.

The mapping process described in section III is used to form the vectors for the control module. Since the total number of elements in the function and the terminal set is 14, the vectors will be formed in $\mathcal{R}^{14}$.

For the first trial the learning period has been set as 30. The genetic parameters used for the trials are as follows:

- Population size $= 100$
- Crossover at function point fraction $= 0.1$
- Crossover at any point fraction $= 0.7$
- Reproduction fraction $= 0.1$
- Mutation fraction $= 0.1$
- Number of Generations $= 5000$
- Selection Method: Fitness Proportional.

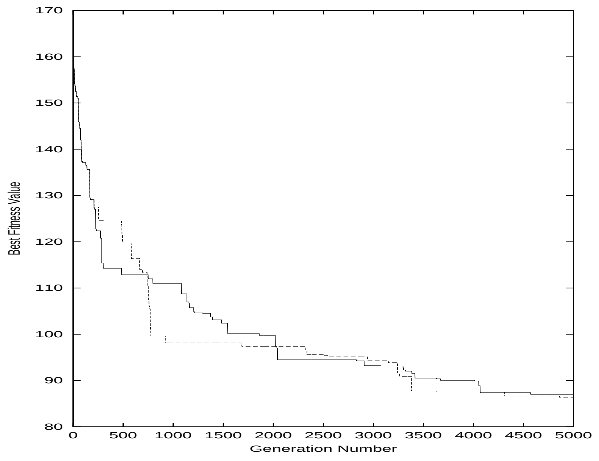

Fig. 4. Comparison of controlled search and basic GP for the CFG-induction problem. The broken line denotes the performance of the controlled search. The learning period is 200. Number of Sample runs is 8.

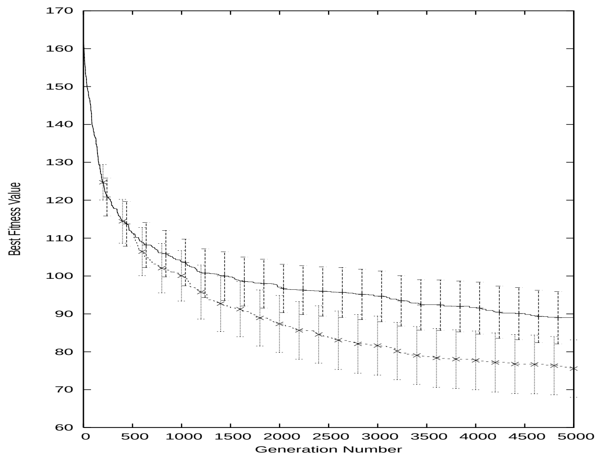

Fig. 5. Comparison of controlled search and basic GP for the CFG-induction problem. The broken line denotes the performance of the controlled search. The learning Period is 500. Number of Sample runs is 80

Both the controlled search and the straightforward application of GP were run using 8 different random seeds. Certainly, the random seeds are kept same for each method. Surprisingly, it was observed that the controlled search performed worse than the straightforward application. It seems that the information sent by the control module to the genetic engine was misleading and directed the search to a local minima, resulting a performance worse than random crossover. An increase in the performance was obtained with simpler data and with a smaller number of function elements. The details of this initial attempt can be found in [11]. The main difference with this initial attempt is the total number of function and terminal elements used. This total number is 14. It is thought that the learning period of 30 might be too low for making reasonable abstractions over vectors with this dimension. On the other hand, it was observed that the decision trees induced for this case were simple and contained less information. Therefore it was decided to increase the learning period.

Figure 4 presents the comparison with basic GP when the learning period is increased to 200. Again the results denote an average of 8 runs with different random seeds. The performance of the controlled search clearly increased, compared to the trial with a learning period of 30. However, it was still not the case that the controlled search outperformed the straightforward application.

The increase in the performance parallel to the increase in the learning period was however, encouraging. Therefore another trial was carried out, this time with a learning period of 500 generations. Figure 5 presents the best fitness averages obtained in the new trial, together with the confidence intervals. This time the average of 80 different runs were used in order to increase the reliability of the result. As can be seen in the figure, the desired performance increase was obtained. From this experiment, it can be concluded that the learning period is a critical parameter of the proposed system; and setting it to a small value may even worsen the performance compared to ordinary GP. Significance in performance increase starts with a learning period of 500 generations.

V. $TESTBED_2$: N-Parity Problem

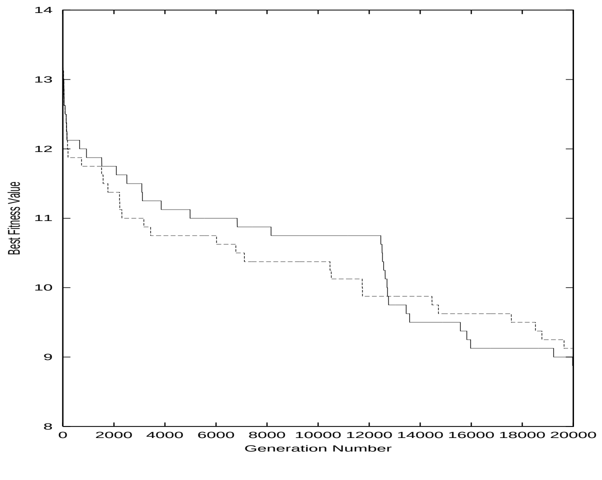

Fig. 6. Comparison of controlled search and basic GP for the N-Parity problem. The broken line denotes the performance of the controlled search. The learning period is 200.Number of Sample runs is 8

The N-parity problem was selected as the second to analyze our approach. As is well known, in this test problem, the aim is to induce a function which takes a binary sequence of length n and returns true if the number of ones in the sequence is odd, and is false otherwise. The function would consist of internal operators $AND, OR, NAND$ and NOR, [12].

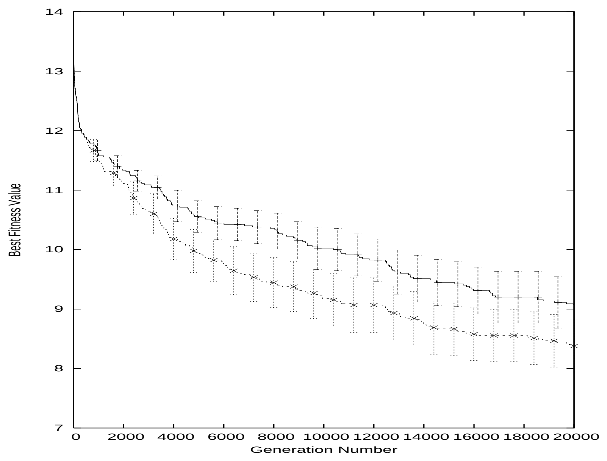

Fig. 7. Comparison of controlled search and basic GP for the N-Parity problem. The broken line denotes the performance of the controlled search. The learning period is 500. Number of Sample runs is 45

The problem is relevant to our purposes as it is highly deceptive. [3] states that the problem quickly becomes more difficult with increasing order. He also notes that flipping any bit in the sequence inverts the outcome of the parity function and states this as a fact to denote the hardness of the problem.

The 5-parity problem was chosen for the test cases since [3] denotes that GP is unable to provide a solution to it.

The function and the terminal sets are $F = \{AND, OR, NAND, NOR\}$ and $T = \{X_1, X_2, X_3, X_4, X_5\}$. T represents the binary input sequence of length five. The number of possible input binary sequences is 32 for the 5-parity problem. The fitness function simply adds a penalty of one if the induced function returns the wrong answer for an input sequence. Hence, the fitness value ranges between 0 and 32.

The genetic parameters used for the trials were as follows:

- Population size $= 100$
- Crossover at function point fraction $= 0.1$
- Crossover at any point fraction $= 0.7$
- Reproduction fraction $= 0.1$
- Mutation fraction $= 0.1$
- Number of Generations $= 20000$
- Selection Method: Fitness Proportional.

The first test case was repeated using a learning period of 30. Similarly, 8 different runs were carried out with various random seeds both for basic and controlled GP. The results obtained were consistent with the CFG-induction problem. Again the controlled search exhibited a worse performance. Considering the total number of terminal and function set elements, which is 9, obtaining such a performance was not surprising. Therefore, for the second test case a period of 200 generations was used. The results of this test case are presented in figure 6. Again the results are consistent with the results obtained for CFG-induction. The controlled search can compete with the straightforward application but still cannot outperform it. A test with a learning period of 500 generations was carried out and the results are presented in figure 7. Again, the number of sample runs is increased for this successful case and the average of 45 different runs are presented for this learning period, together with the confidence intervals. This period is sufficient for the 5-parity problem too and the performance increase is outstanding.

VI. Determining the Limits of our Approach

In the previous sections the method was tested on two real world problems. Obviously obtaining performance increase in only two domains is not sufficient to claim a general improvement for all deceptive problems. In order to get more insight into the contribution of the control module, it was decided to use a tunable benchmark problem. It was hoped that applying the method to different instances of the problem would confirm that the significance of the method increases as the problem is made more deceptive.

A. Choosing a Benchmark Problem

Not so many benchmark problems have been proposed in the area of GP. The benchmark problem proposed in [19] is notable since their formalization shares some characteristics with the *Royal Road Problem*. This is a commonly used benchmark problem for tuning the genetic parameters in the field of GA [8]. The formalization in [19] is based on the definition of a *"perfect tree"* of some depth. For instance a level-a tree is perfect when its root is the function a with a single child. Similarly a perfect level-b tree's root should be function b having two perfect level-a trees as children, a perfect level-c tree will have three perfect level-b trees connected to root c, and so on. The terminal set consists of a single element x. Note that the series of functions $a, b, c, d...$ are defined with increasing arity. The raw fitness of such a tree is defined as the score of its root. On the other side the score of each function is calculated by adding the weighted scores of its children. The weight is a constant larger than one (bonus) if a child is a perfect tree of the appropriate level. When the child is not a perfect tree, the weight turns out to be a constant smaller or equal to one (Penalty or Partial Bonus) depending on the child's configuration. It is suggested that perfect trees of different depths can be used as benchmark problems for GP.

The proposed definition in [19] has certain drawbacks. Our main focus is to be able to control the degree of deception in the problem. Deception usually appears as an outcome of *Epistasis*. That is the interdependency among the subparts of the chromosome should have an influence on the global fitness of the chromosome. However, the formalization provided by [19] enables each child to contribute to the global fitness independent of other children. What is more, since the functions are defined with increasing arity, the search space increases rapidly for high levels. The problem becomes too difficult after level d and e. On the other hand, levels a, b and c are too simple. It is possible for the solution to appear in the initial random population in these levels. Lastly, more than one constant is used in [19] and this makes the definition unnecessarily complicated.

B. A New Benchmark Problem

Considering the drawbacks mentioned in the previous section, it was decided to simplify and reorganize the definition of a perfect tree. The aim is to obtain a tunable benchmark problem where epistasis can easily be controlled. The new perfect tree is defined as a full binary tree of some depth. The function set consists of n functions with arity two $(F = \{F_1, F_2, ..F_n\})$. The terminal set consists of a single terminal element t, $(T = \{t\})$. The raw fitness of a tree is again defined as the score of its root. The fitness of a terminal node is simply defined as 1. The fitness of an internal node is defined as the sum of the fitnesses of its two children. The two constraints used to create epistasis for the problem are the following.

When the children of an internal node are not terminal elements:

(i) The index of the parent function should be smaller than the index of its children.
(ii) The index of the right-hand child should be larger than the index of the left-hand child.

Based on these constraints, the fitness function for an internal node is defined as:

$$f(P) = \begin{cases} f(C_1) + f(C_2) & (i) \\ C_{epis} \cdot f(C_1) + f(C_2) & (ii) \\ f(C_1) + C_{epis} \cdot f(C_2) & (iii) \\ C_{epis} \cdot f(C_1) + C_{epis} \cdot f(C_2) & (iv) \end{cases} \tag{13}$$

In equation 13, part (1) is chosen if both of the children are terminal elements or none of the constraints are violated. If the first constraint is violated by any child, the fitness of that child is multiplied with a constant smaller than one. This constant is called the *epistasis constant* (C_{epis}). Parts (2) and (3) in equation 13 reflect this situation. Lastly, part (4) is used when both of the children violate the first constraint or when the second constraint cannot be achieved.

These two constraints create interdependency among the subparts of a chromosome. When the epistasis constant is decreased the interdependency increases. Hence it becomes impossible for a subpart of the chromosome to contribute to the global fitness independent of other parts. A well-fit chromosome can easily be ruined when a node that violates a constraint appears after a recombination operation. Hence fitnesses of similar chromosomes might differ remarkably and the search space becomes discontinuous. In addition to this, when the function set is kept small, it becomes more probable for the search to get stuck in a local minima. Since the index of the functions should increase as you go down in a chromosome, a wrong choice close to the root might make it impossible to form a perfect tree at all.

C. $TESTBED_3$: The Benchmark Problem

For the testing phase the function set is fixed as $F = \{F_1, F_2, ..., F_8\}$ and the terminal set is $T = \{t\}$. Also the depth of the perfect tree to be searched is determined as 4. In figure 8 a possible solution is given for depth 4. Here the function F_8 has not been used, however other solutions exist that would include the eighth function too.

The different epistasis constants used for the testing process were 0.5,0.35, 0.2, 0.05, 0.002. The genetic parameters were set as follows:

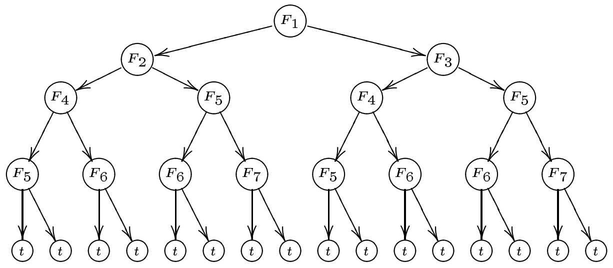

Fig. 8. A possible solution of depth four.

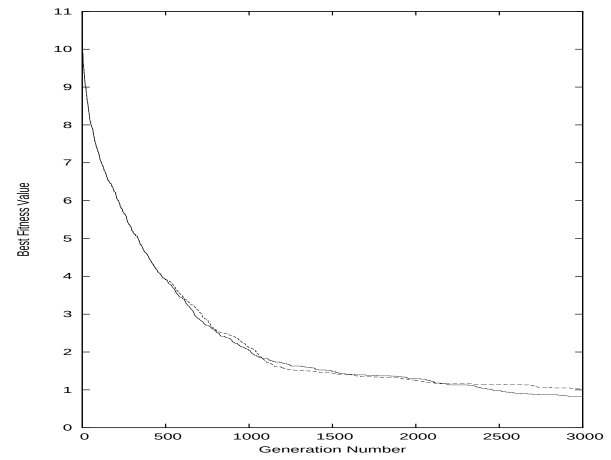

Fig. 9. **(Non-deceptive Case)** Comparison of controlled search and basic GP. The broken line denotes the performance of the controlled search. The learning period is 500. The epistasis constant is 0.5. Number of sample runs is 100.

- Population size $= 100$
- Crossover at function point fraction $= 0.1$
- Crossover at any point fraction $= 0.7$
- Reproduction fraction $= 0.1$
- Mutation fraction $= 0.1$
- Number of Generations $= 3000$
- Selection Method: Fitness Proportional.

As seen in figure 9, it is not possible to obtain an improvement when the epistasis constant is 0.5. However, as the problem is made more deceptive by decreasing the epistasis constant, the performance increase becomes more significant. In figure 10, the comparison between basic GP and the controlled search is presented for the smallest epistasis constant 0.002. The controlled search clearly outperforms basic GP when such a low epistasis constant is used.

The behavior of basic GP and the controlled search can best be seen in figure 11. In this figure the comparison of the best fitness averages obtained at the end of 3000 generations is presented for each epistasis constant. As seen in the figure, the controlled search is misleading for the epistasis constant 0.5. Basic GP can achieve a better average at the end of the search. However, as the epistasis constant is decreased the performance of basic GP rapidly goes down. On the other hand, the controlled search can compensate the deception, and the performance decrease is not dramatic as deception increases. It can be claimed that the results obtained are consistent with our proposal and the significance of the controlled search becomes more apparent with high deception.

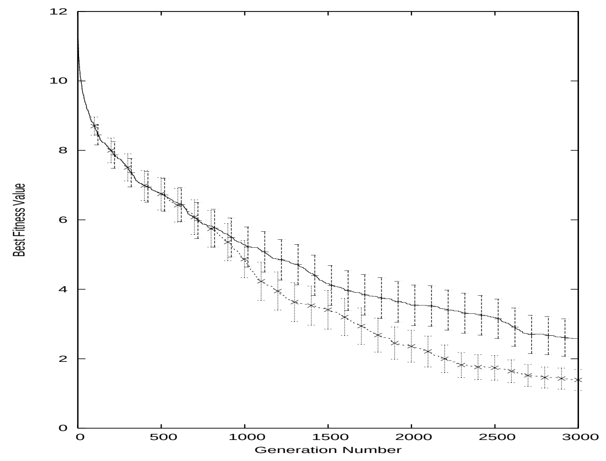

Fig. 10. **(Highly-deceptive Case)** Comparison of controlled search and basic GP. The broken line denotes the performance of the controlled search. The learning period is 500. The epistasis constant is 0.002. Number of sample runs is 100.

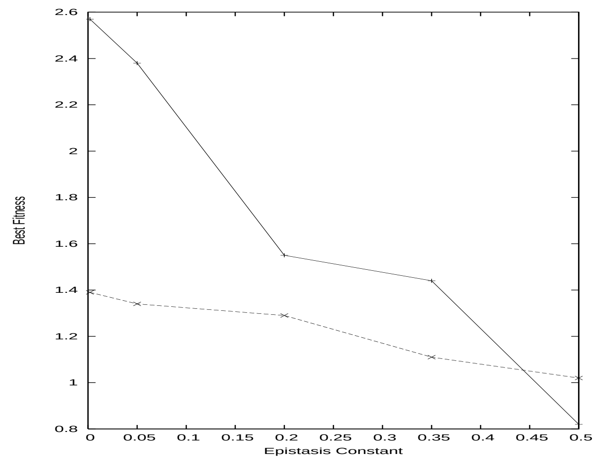

Fig. 11. Best fitness averages obtained at the end of 3000 runs.

VII. Testing the Reliability of the Results Obtained

The results presented in the previous sections are the best fitness averages obtained by a certain number of sample runs. This is a common comparison method used in the GP community. Usually $20 - 30$ runs are considered sufficient for a satisfactory comparison between two methods. However, this is questionable in terms of statistics. Statistical significance between two means depends on two parameters, the *population size* (number of sample runs for our case) and the *variance* in the population. Therefore it is not possible to set a general lower limit for the number of sample runs that can be satisfactory for all problems. This lower limit changes depending on the standard deviation that appears throughout the runs. Statistical tests are needed to determine if the number of runs used are enough for a satisfactory comparison between methods.

Although this approach is not common in evolutionary computation community, some researchers have started to highlight the importance of the subject. In [16], for instance, it is noted that performance comparison is an important subject in GP research, since much published research includes the comparison of one technique with another.

T-test is offered as a statistical method that can be used for comparing small samples. The formulation for calculating the t value is given in [6]. The *t-value* obtained by the equation corresponds to a risk level depending on the initial hypothesis. Our task is to determine if the best fitness averages obtained during the controlled search are really smaller than the ones obtained by basic GP. Therefore we used the *one-tailed* version of the test, [16]. The risk level denotes the probability of obtaining the difference between the two means by chance. In statistics a risk level smaller than 0.05 is usually considered to be statistically significant.

In order to apply the test to our method, the t-value for the difference between the two means were calculated for each generation. Then, using an automated tool corresponding risk levels were obtained. There is no difference between the two means until the end of the first learning period. Therefore the risk level was 0.5 before the controlled search started. After the first learning period, the risk level was expected to decrease below 0.05 in a certain amount of generations.

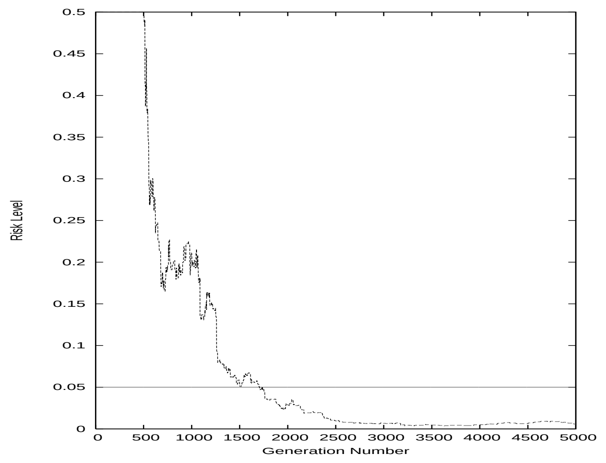

Fig. 12. T-test for the CFG induction problem. The learning Period is 500 and number of sample runs is 80.

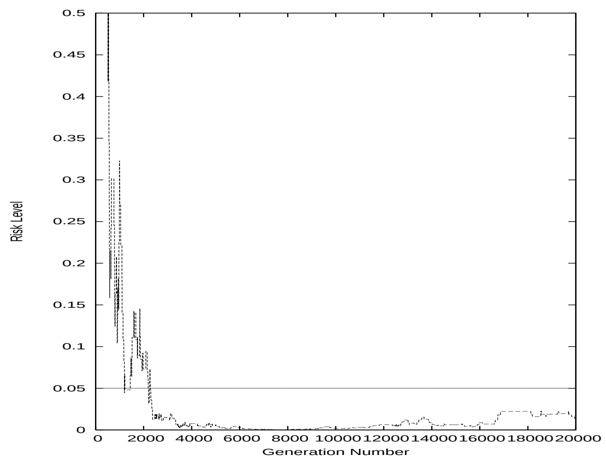

Fig. 13. T-test for the N-Parity problem. The learning Period is 500 and number of sample runs is 45.

The test was first applied to the instances of the benchmark problem except the instance with epistasis constant 0.5. Note that no improvement had been obtained for that case. The number of sample runs used for this problem was 100. This can be considered as quite a big population for the test and, as expected, all four instances of the problem were able to pass the test. It was possible to obtain risk levels smaller than 0.01 towards the end of the search. This denotes that the performance increase obtained is statistically significant with a probability larger than 0.99.

The test was also applied to the two real world domains. In figures 5 and 7, the output of the t-test is presented for the two problems. As seen in the figures, the difference between the means turns out to be statistically significant after about 2000 generations and the risk level stays far below the critical value 0.05 throughout the search.

VIII. Conclusion and Future Work

Our initial question was, whether it could be possible to extract information during genetic evolution and use this information to increase the performance of GP by controlling recombination operations afterwards. Our focus has been on highly deceptive problems, therefore we have tried to extract information about the global structure of chromosomes. The results obtained in different domains exhibit strong evidence of the success of our approach.

The performance increase obtained in only two domains is not sufficient to claim a general improvement for GP in the deceptive domain. However, the experiments carried out on the benchmark problem provides an insight into the contribution of the approach. The different instances of the problem has made it possible to carry out a controlled experiment to trace the behavior of the new method. The results obtained clearly show that the performance decrease that occurs due to high epistasis can be blocked by the control module. Interestingly, the contribution of the module becomes significant when inter-dependency among the subparts of chromosomes is increased.

References

[1] Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Genetic programming and deductive-inductive learning: A multistrategy approach. In Jude Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning, ICML'98*, pages 10–18, Madison, Wisconsin, USA, July 1998. Morgan Kaufmann.

[2] Shumeet Baluja and Scott Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proc. 14th International Conference on Machine Learning*, pages 30–38. Morgan Kaufmann, 1997.

[3] Edwin D. de Jong, Richard A. Watson, and Jordan B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 11–18, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[4] Patrik D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[5] J. Eggermont and J. I. van Hemert. Stepwise adaptation of weights for symbolic regression with genetic programming. In *Proceedings of the Twelveth Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'00)*, 2000.

[6] Gary A. Freund, John E. Simon. *Modern Elementary Statistics*. Prentice Hall, 1992.

[7] Hitoshi Iba and Hugo de Garis. Extending genetic programming with recombinative guidance. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 4, pages 69–88. MIT Press, Cambridge, MA, USA, 1996.

[8] Terry Jones. A description of holland's royal road function. *Evolutionary Computation*, 2(4):409–415, 1995.

[9] Hugues Juille and Jordan B. Pollack. A sampling-based heuristic for tree search applied to grammar induction. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) Tenth Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, Madison, Wisconsin, USA, 26-30 1998. AAAI Press Books.

[10] B. Keller and R. Lutz. Evolving stochastic context-free grammars from examples using a minimum description length principle. In *Proceedings of the Workshop on Automata, Inductive Grammatical Inference and Language Acquisition. ICML-97*, 1997.

[11] Emin Erkan Korkmaz and Gokturk Ucoluk. Genetic programming for grammar induction. In Erik D. Goodman, editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 245–251, San Francisco, California, USA, 9-11 July 2001.

[12] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, 1992.

[13] George Lakoff. *Women, Fire, and Dangerous Things*. Chicago University Press, Chicago, Mi, 1987.

[14] Simon Lucas. Structuring chromosomes for context-free grammar evolution. In *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1994.

[15] Heinz Muhlenbein and Gerhard PaaB. From recombination of genes to the estimation of distributions: I. binary parameters. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving From Nature-PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer-Verlag, Berlin, 1996.

[16] Norman Paterson and Michael Livesey. Performance comparison in genetic programming. In Darrell Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 253–260, Las Vegas, Nevada, USA, 8 July 2000.

[17] Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation*, 9(4):311–340, 2000.

[18] Martin Pelikan and Heinz Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chawdhry, editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London, 1999. Springer-Verlag.

[19] William F. Punch, Douglas Zongker, and Erik D. Goodman. The royal tree problem, a benchmark for single and multiple population genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 15, pages 299–316. MIT Press, Cambridge, MA, USA, 1996.

[20] J.Ross. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[21] T.C. Smith and I.H. Witten. A genetic algorithm for the induction of natural language grammars. In *Proceedings of IJCAI-95 Workshop on New Approaches to Learning for Natural Language Processing*, pages 17–24, Montreal, Canada, 1995.

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

[23] Elena Zannoni and Robert G. Reynolds. Learning to control the program evolution process with cultural algorithms. *Evolutionary Computation*, 5(2):181–211, summer 1997.