

# Learning Probabilistic Tree Grammars for Genetic Programming

*Peter A.N. Bosman*

*Edwin D. de Jong*

institute of information and computing sciences, utrecht university

technical report UU-CS-2004-048

[www.cs.uu.nl](http://www.cs.uu.nl)

# Learning Probabilistic Tree Grammars for Genetic Programming

Peter A.N. Bosman and Edwin D. de Jong

Institute of Information and Computing Sciences, Utrecht University,  
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands  
{*Peter.Bosman, dejong*}@cs.uu.nl

**Abstract.** Genetic Programming (GP) provides evolutionary methods for problems with tree representations. A recent development in Genetic Algorithms (GAs) has led to principled algorithms called Estimation-of-Distribution Algorithms (EDAs). EDAs identify and exploit structural features of a problem’s structure during optimization. Here, we investigate the use of a specific EDA for GP. We develop a probabilistic model that employs transformations of production rules in a context-free grammar to represent local structures. The results of performing experiments on two benchmark problems demonstrate the feasibility of the approach.

## 1 Introduction

GP [5, 7] offers algorithms to search highly expressive classes of functions, and has been applied to a diverse range of problems including circuit design, symbolic regression, and control. Most GP methods employ the subtree-crossover operator, which exchanges randomly selected subtrees between individuals. Due to the particular structure of trees, subtrees (rather than e.g. any combination of nodes and arcs) appear a reasonable choice as the form of the partial solutions that will be exchanged since the functionality of a subtree is independent of its place within the tree. However, basic GP does not make informed choices as to *which* partial solutions will be exchanged; both the size of the removed subtree and the position where it will be inserted are chosen randomly.

Several subtree encapsulation methods exist that make more specific choices as to which partial solutions are selected to be further propagated, e.g. GLiB [1], ADFs [8], and ARL [14]. Subtree encapsulation methods have been found to substantially improve performance on a variety of problems. Yet, the criteria for the selection of partial solutions they employ are still heuristic; typically, either the fitness of the tree in which a subtree occurs is used as an indication of its value, or the partial solution is itself subject to evolution.

Here, we explore whether a more principled approach to the selection of partial solutions and their recombination is possible. If the distribution of high-fitness trees can be estimated, this would directly specify which combinations of elements are to be maintained in the creation of new individuals and thus which combinations of partial solutions may be fruitfully explored. We investigate how the principle of distribution estimation may be employed in the context of tree-based problems. In GAs, the development of EDAs and other linkage learning

techniques has yielded a better insight into the design of competent GAs by rendering the assumptions implicitly made by algorithms explicit. Our aim is that the application of distribution estimation techniques to tree-based problems may likewise clarify the design of principled methods for GP. This paper represents a first step in that direction.

To estimate distributions over the space of trees, a representation must be chosen. In Probabilistic Incremental Program Evolution (PIPE) [15], trees are matched on a fixed-size template, such that the nodes becomes uniquely identifiable variables. The size of the template may increase over time however as trees grow larger. Using this representation, all nodes are treated equally. While this permits the encapsulation of any combination of nodes, it does not exploit the particular non-fixed-size and variable-child-arity structure of trees. The complexity of such an algorithm is determined by the maximally allowed shape for the trees, which must be chosen in advance. In PIPE the distribution is the same for each node, which corresponds to a univariately factorized probability distribution over all nodes. In Extended Compact Genetic Programming (ECGP) [16], trees are represented in the same way as in PIPE. However, the probability distribution that is estimated is a marginal-product factorization that allows the modelling of dependencies between multiple nodes that are located anywhere in the fixed-size template. Moreover, the size of the template is fixed beforehand in ECGP and does not increase over time. In this paper, a method will be employed that estimates the distribution of trees based on the subtrees that actually occur. The representation specifies a set of rules whose expansion leads to trees. The rules capture local information, thereby offering a potential to exploit the specific structure of trees, while at the same time their use in an EDA offers a potential for generalization that is not provided by using fixed-size templates. Shan *et al.* [17] use a similar approach (based on stochastic grammars). The main difference is that their approach starts from an explicit description of a specific set of trees and then generalizes the description to represent common subtrees. In our approach we do the reverse. Our approach starts from a description of all possible trees and then tunes it to more specifically describe the set of trees.

The remainder of this paper is organized as follows. In Section 2 we introduce basic notation and terminology. In Section 3 we define the probability distribution over trees that we work with. We also show how to draw new samples from the distribution and how the distribution is estimated from data. In Section 4 we perform experiments on two benchmark problems and compare the performance of standard GP and an EDA based on the proposed distribution. We present our conclusions and a discussion of future research in Section 5.

## 2 Terminology

A grammar  $\mathbf{G}$  is a vector of  $l_r$  production rules  $R_j, j \in \{0, 1, \dots, l_r - 1\}$ , that is,  $\mathbf{G} = (R_0, R_1, \dots, R_{l_r-1})$ .

A production rule is denoted by  $R_j : \mathcal{S}_k \rightarrow E_j$ , where  $\mathcal{S}_k$  is a symbol that can be replaced with the expansion  $E_j$  of the production rule. Let  $K$  be the number of available symbols, then  $k \in \{0, 1, \dots, K - 1\}$ . We will use only one symbol and allow ourselves to write  $\mathcal{S}$  instead of  $\mathcal{S}_k$ .

An expansion  $E_j$  of production rule  $R_j$  is a tree. We will therefore generally call  $E_j$  an expansion tree. The internal nodes of an expansion tree are functions with at least one argument. The leaves are either symbols, constants or input variables. An example of an expansion tree is  $E_j = +(\sin(\mathcal{S}), (-\log(\mathcal{S}), \cos(\mathcal{S})))$  which, in common mathematical notation, represents  $\sin(\mathcal{S}) + (\log(\mathcal{S}) - \cos(\mathcal{S}))$ .

A sentence is obtained from a grammar if a production rule is chosen to replace a symbol repeatedly until all symbols have disappeared. Sentences can therefore be seen as trees. We denote a sentence by  $s$ . We will denote a subtree of a sentence by  $t$  and call it a sentence subtree.

A sentence subtree  $t$  is said to be matched by an expansion tree  $E_j$ , denoted  $t \in E_j$  if and only if all nodes of  $E_j$  coincide with nodes found in  $t$  following the same trails, with the exception of symbol nodes, which may be matched to any non-empty sentence subtree.

The following grammar  $\mathbf{G} = (R_0, R_1, \dots, R_{5+n})$  with  $l_r = 6+n$  is an example of a grammar that describes certain  $n$ -dimensional real-valued functions:

$$\begin{array}{ll} R_0 : \mathcal{S} \rightarrow c \text{ (a constant } \in \mathbb{R}) & R_{n+1} : \mathcal{S} \rightarrow +(\mathcal{S}, \mathcal{S}) \\ R_1 : \mathcal{S} \rightarrow i_0 \text{ (input variable 0)} & R_{n+2} : \mathcal{S} \rightarrow \cdot(\mathcal{S}, \mathcal{S}) \\ \vdots & R_{n+4} : \mathcal{S} \rightarrow \sin(\mathcal{S}) \\ R_n : \mathcal{S} \rightarrow i_{n-1} \text{ (input variable n-1)} & R_{n+5} : \mathcal{S} \rightarrow \cos(\mathcal{S}) \end{array}$$

### 3 Probability distribution

#### 3.1 Definition

To construct a probability distribution over sentences, we introduce a random variable  $S$  that represents a sentence and a random variable  $T$  that represents a sentence subtree. Because sentence subtrees are recursive structures we define a probability distribution for sentence subtrees recursively. To do so, we must know where the tree terminates. This information can be obtained by taking the depth of a sentence subtree into account. Let  $P^{\mathbf{G}}(T = t|D = d)$  be a probability distribution over all sentence subtrees  $t$  that occur at depth  $d$  in a sentence. Now, we define the probability distribution over sentences  $s$  by:

$$P^{\mathbf{G}}(S = s) = P^{\mathbf{G}}(T = s|D = 0) \tag{1}$$

Since sentence subtrees are constructed using production rules, we can define  $P^{\mathbf{G}}(T = t|D = d)$  using the production rules. Since there is only one symbol, we can also focus on the expansion trees. Although depth can be used to model the probability of terminating a sentence at some node in the tree, sentences can be described more precisely if depth is also used to model the probability of occurrence of functions at specific depths. Preliminary experiments indicated that this use of depth information leads to better results.

We define  $P_j^E(J = j|D = d)$  to be a discrete conditional probability distribution that models the probability of choosing expansion tree  $E_j$ ,  $j \in \{0, 1, \dots, l_r\}$  at depth  $d$  when constructing a new sentence.

We assume that the values of the constants and the indices of the input variables in an expansion tree are not dependent on the depth. Conforming to

this assumption we define  $2l_r$  multivariate probability distributions that allow us to model the use of constants and inputs inside production rules other than the standard rules  $\mathcal{S} \rightarrow c$  and  $\mathcal{S} \rightarrow i_k$ ,  $k \in \{0, 1, \dots, n-1\}$ :

- $P_j^C(\mathbf{C}_j)$ ,  $j \in \{0, 1, \dots, l_r - 1\}$ , a probability distribution over all constants in expansion tree  $E_j$ , where  $\mathbf{C}_j = (C_{j0}, C_{j1}, \dots, C_{j(n_{C_j}-1)})$ . Each  $C_{jk}$  is a random variable that represents a constant in  $E_j$ .
- $P_j^I(\mathbf{I}_j)$ ,  $j \in \{0, 1, \dots, l_r - 1\}$ , a probability distribution over all inputs in expansion tree  $E_j$ , where  $\mathbf{I}_j = (I_{j0}, I_{j1}, \dots, I_{j(n_{I_j}-1)})$ . Each  $I_{jk}$  is a random variable that represents all input variables, i.e.  $I_{jk} \in \{0, 1, \dots, n-1\}$ .

The above definition of  $P_j^I(\mathbf{I}_j)$  enforces a single production rule  $\mathcal{S} \rightarrow i$ , where  $i$  represents all input variables, instead of  $n$  production rules  $\mathcal{S} \rightarrow i_j$ ,  $j \in \{0, 1, \dots, n-1\}$ . This reduces the required computational complexity for estimating the probability distribution, especially if  $n$  is large. However, it prohibits the introduction of production rules that make use of specific input variables.

We will enforce that any sentence subtree  $t$  can be matched by only one expansion tree  $E_j$ . The probability distribution over all sentence subtrees at some given depth  $D = d$  is then the product of the probability of matching the sentence subtree with some expansion tree  $E_j$  and the product of all (recursive) probabilities of the sentence subtrees located at the symbol-leaf nodes in  $E_j$ .

Let  $\mathcal{S}_{jk}$  be the  $k$ -th symbol in expansion tree  $E_j$ ,  $k \in \{0, 1, \dots, n_{\mathcal{S}_j} - 1\}$ . Let  $stree(\mathcal{S}_{jk}, t)$  be the sentence subtree of sentence subtree  $t$  at the same location where  $\mathcal{S}_{jk}$  is located in the expansion tree  $E_j$  that matches  $t$ . Let  $depth(\mathcal{S}_{jk})$  be the depth of  $\mathcal{S}_{jk}$  in expansion tree  $E_j$ . Finally, let  $match(t)$  be the index of the matched expansion tree, i.e.  $match(t) = j \Leftrightarrow t \in E_j$ . We then have:

$$P^G(T = t | D = d) = \tag{2}$$

$$P^E(J = j | D = d) P_j^C(\mathbf{C}_j) P_j^I(\mathbf{I}_j) \prod_{k=0}^{n_{\mathcal{S}_j}-1} P^G(T = stree(\mathcal{S}_{jk}, t) | D = d + depth(\mathcal{S}_{jk}))$$

where  $j = match(t)$

Summarizing, formulating a probability distribution for sentences requires:

- $P^E(J | D)$ ; a univariate discrete conditional probability distribution over all possible production rules, conditioned on the depth of occurrence.
- $P_j^C(\mathbf{C}_j)$ ;  $l_r$  multivariate probability distributions over  $n_{C_j}$  variables.
- $P_j^I(\mathbf{I}_j)$ ;  $l_r$  multivariate probability distributions over  $n_{I_j}$  variables.

### 3.2 Estimation from data

**General greedy approach** We choose a greedy approach to estimate  $P^G(S)$  from a set  $\mathcal{S}$  of sentences. Greedy approaches have been used in most EDAs so far and have led to good results [2, 9–11]. The algorithm starts from a given initial grammar. Using transformations, the grammar is made more involved. A

transformation results in a set of candidate grammars, each slightly different and each more involved than the current grammar. The goodness of each candidate grammar is then evaluated and the best one is accepted if it is better than the current one. If there is no better grammar, the greedy algorithm terminates.

**Transformations** The only transformation that we allow is the substitution of one symbol of an expansion tree with one expansion tree from the base grammar. The base grammar is the grammar that is initially provided. To ensure that after a transformation any sentence subtree can be matched by only one expansion tree we assume that the base grammar has this property. To transform the grammar we only allow to substitute symbols with expansion trees of the base grammar. Here is an example of expanding the base grammar (first column) using a single expansion (second column):

| <u>Base grammar</u>   | <u>Single expansion</u>   | <u>Full expansion</u>   |
|---|---|---|
| $R_0 : \mathcal{S} \rightarrow c$                           | $R_0 : \mathcal{S} \rightarrow c$   | $R_0 : \mathcal{S} \rightarrow c$   |
| $R_1 : \mathcal{S} \rightarrow i$                           | $R_1 : \mathcal{S} \rightarrow i$   | $R_1 : \mathcal{S} \rightarrow i$   |
| $R_2 : \mathcal{S} \rightarrow f(\mathcal{S}, \mathcal{S})$ | $R_2 : \mathcal{S} \rightarrow f(\mathcal{S}, \mathcal{S})$                 | $R_2 : \mathcal{S} \rightarrow f(\mathcal{S}, \mathcal{S})$                 |
| $R_3 : \mathcal{S} \rightarrow g(\mathcal{S}, \mathcal{S})$ | $R_3 : \mathcal{S} \rightarrow g(\mathcal{S}, \mathcal{S})$                 | $R_3 : \mathcal{S} \rightarrow g(c, \mathcal{S})$                           |
|   | $R_4 : \mathcal{S} \rightarrow g(f(\mathcal{S}, \mathcal{S}), \mathcal{S})$ | $R_4 : \mathcal{S} \rightarrow g(i, \mathcal{S})$                           |
|   |   | $R_5 : \mathcal{S} \rightarrow g(f(\mathcal{S}, \mathcal{S}), \mathcal{S})$ |
|   |   | $R_6 : \mathcal{S} \rightarrow g(g(\mathcal{S}, \mathcal{S}), \mathcal{S})$ |

Note that the set of expansion trees can now no longer be matched uniquely to all sentence trees. For instance, sentence  $g(f(c, c), c)$  can at the top level now be matched by expansion trees 3 and 4. To ensure only a single match, we could expand every expansion tree from the base grammar into a production rule and subsequently remove the original production rule that has now been expanded (third column in the example above). However, this rapidly increases the number of production rules in the grammar and may introduce additional rules that aren't specifically interesting for modelling the data at hand. To be able to only introduce the rules that are interesting, we equip the symbols with a list of indices that indicate which of the production rules in the base grammar may be matched to that symbol. Once a substitution occurs, the symbol that was instantiated may no longer match with the expansion tree that was inserted into it. For example:

| <u>Base grammar</u>   | <u>Expanded grammar</u>   |
|---|---|
| $R_0 : \mathcal{S} \rightarrow c$   | $R_0 : \mathcal{S} \rightarrow c$   |
| $R_1 : \mathcal{S} \rightarrow i$   | $R_1 : \mathcal{S} \rightarrow i$   |
| $R_2 : \mathcal{S} \rightarrow f(\mathcal{S}^{0,1,2,3}, \mathcal{S}^{0,1,2,3})$ | $R_2 : \mathcal{S} \rightarrow f(\mathcal{S}^{0,1,2,3}, \mathcal{S}^{0,1,2,3})$                           |
| $R_3 : \mathcal{S} \rightarrow g(\mathcal{S}^{0,1,2,3}, \mathcal{S}^{0,1,2,3})$ | $R_3 : \mathcal{S} \rightarrow g(\mathcal{S}^{0,1,3}, \mathcal{S}^{0,1,2,3})$                             |
|   | $R_4 : \mathcal{S} \rightarrow g(f(\mathcal{S}^{0,1,2,3}, \mathcal{S}^{0,1,2,3}), \mathcal{S}^{0,1,2,3})$ |

A sentence can now be preprocessed bottom-up in  $\mathcal{O}(n)$  time to indicate for each node which expansion tree matches that node, where  $n$  is the number of nodes in the tree. The sentence can then be traversed top-down to perform the frequency count for the expansion trees. It should be noted that this approach means that if a symbol list does not contain all indices of the base

grammar, then it represents only the set of the indicated rules from the base grammar. In the above example for instance  $\mathcal{S}^{0,1,3}$  represents  $\mathcal{S} \rightarrow c$ ,  $\mathcal{S} \rightarrow i$  and  $\mathcal{S} \rightarrow g(\mathcal{S}^{0,1,2,3}, \mathcal{S}^{0,1,2,3})$ . Therefore, the probability associated with this particular symbol is not the recursive application of the distribution in equation 2, but is uniform over the indicated alternatives. This is comparable to the approach of default tables for discrete random variables in which instead of indicating a probability for all possible combinations of values for the random variables, only a subset of them is explicitly indicated. All remaining combinations are assigned an equal probability such that the distribution sums to 1 over all possible values.

**Goodness measure** The goodness of a grammar is determined using its associated probability distribution. This distribution can be estimated by traversing each sentence and by computing the proportions of occurrence for the expansion trees at each depth. Probability distributions  $P_j^C(\mathcal{C}_j)$  and  $P_j^I(\mathcal{I}_j)$  can be estimated after filtering the sentences to obtain for each expansion tree a set of sets of constants and a set of sets of variables (one set of constants or variables for each match of the expansion tree). From these sets, the distributions  $P_j^C(\mathcal{C}_j)$  and  $P_j^I(\mathcal{I}_j)$  can be estimated using well-known density estimation techniques (such as proportion estimates for the discrete input variable indices).

Once all parameters have been estimated, the probability-distribution value of each sentence in the data can be computed to obtain the likelihood (or the negative log-likelihood). The goodness measure that we ultimately use to distinguish between probability distributions is the MDL metric [13], which is a form of likelihood penalization, also known as the extended likelihood principle [4]:

$$\text{MDL}(P^{\mathcal{G}}(S)) = - \sum_{i=0}^{|\mathcal{S}|-1} \ln(P^{\mathcal{G}}(S = \mathcal{S}_i)) + \frac{1}{2} \ln(|\mathcal{S}|) |\boldsymbol{\theta}| + \zeta \quad (3)$$

where  $|\boldsymbol{\theta}|$  denotes the number of parameters that need to be estimated in probability distribution  $P^{\mathcal{G}}(T)$  and  $\zeta$  denotes the expected additional number of bits required to store the probability distribution. In our case,  $|\boldsymbol{\theta}|$  equals:

$$|\boldsymbol{\theta}| = 2d_{\max} + l_r - 4 + \left( \sum_{j=0}^{l_r-1} |\boldsymbol{\theta}| \leftarrow P_j^C(\mathcal{C}_j) \right) + \left( \sum_{j=0}^{l_r-1} n^{n_{I_j}} - 1 \right) \quad (4)$$

Since in the latter case, the number of parameters grows exponentially with the number of input variables in an expansion tree, this is likely to be a strong restriction on the number of input variables that the greedy algorithm will allow into any expansion tree. As an alternative, independence can be enforced between input variables. The resulting distribution can then no longer model correlation between multiple input variables in one expansion tree. The number of parameters to be estimated reduces drastically however and this in turn will allow more input variables to be incorporated into the expansion trees. Effectively, we thus enforce  $P_j^I(\mathcal{I}_j) = \prod_{k=0}^{n_{I_j}-1} P_j^I(I_{jk})$ , which changes the last term in the number of parameters  $|\boldsymbol{\theta}|$  into  $\sum_{j=0}^{l_r-1} (n-1)n_{I_j}$ .

The expected additionally required number of bits  $\zeta$  in our case come from the production rules. The more production rules, the more bits are required. Moreover, longer production rules require more bits. The number of bits required to store a production rule is  $\ln(l_{r_{base}})$  times the number of internal nodes in the expansion tree where  $l_{r_{base}}$  is the number of production rules in the base grammar. Finally, symbols have lists of production–rule indices. To store one such list, the number of required bits equals  $\ln(l_{r_{base}})$  times the length of that list. The value of  $\zeta$  equals the sum of these two terms over all production rules.

### 3.3 Sampling

1. Initialize the sentence tree of the new sentence: a single symbol as the root.
2. Set  $d = 0$ .
3. Visit the symbol at the root.
4. Draw an integer  $j \in \{0, 1, \dots, l_r - 1\}$  randomly from  $P^E(J|D = d)$  if the list of production–rule indices associated with the visited symbol is complete, or draw a random integer from the list otherwise.
5. Make a copy of expansion tree  $E_j$ .
6. Draw all constants in the copied tree randomly from  $P_j^C(\mathbf{C}_j)$ .
7. Draw all input indices in the copied tree randomly from  $P_j^I(\mathbf{I}_j)$ .
8. Expand the current sentence tree by replacing the symbol that is being visited with the copied tree.
9. Extract the symbols from the copied tree and their relative depths.
10. Recursively visit each symbol in the copied tree after setting the depth  $d$  properly (current depth plus relative symbol depth).

## 4 Experiments

We have performed experiments with the probability distribution from Section 3 by using it in an EDA. We compared the performance of the resulting EDA with a standard GP algorithm that only uses subtree–swapping crossover. Both algorithms select 50% of the population and apply recombination (i.e. either distribution estimation and sampling or crossover) to obtain new solutions that replace the remaining 50% of the population. We applied the algorithms to two GP–benchmarking problems, namely the royal tree problem by Punch, Zongker and Goodman [12] and the tunable benchmark problem by Korkmaz and Üçoluk [6], which we will refer to as the binary–functions problem. A more detailed description of these problems may be found in the aforementioned references or in related work [3]. Both problems are non–functional, which means that fitness is defined in terms of the structure of the sentence and input variables have an empty domain. The binary–functions problem is harder because it contains a higher order of dependency and it has multiple (sub)optima.

For both problems, we have computed convergence graphs for both algorithms and three different population sizes. The curves in the graphs are averaged over 25 runs. For the Royal Tree problem (see Fig. 1, left), GP achieves



reasonable results for population sizes 500 and 1000, but does not reliably find the optimum. The EDA algorithm identifies the optimal solution in every single run and for all population sizes. The Royal Tree problem is a benchmark problem for GP that features a depth-wise layered structure. Clearly, the use of depth information therefore renders the EDA particularly appropriate for problems of this kind. Still, this result demonstrates the feasibility of our EDA to GP. Figure 1 (right) shows the results when recombination is performed in only 90% of the cases, and copying a random parent is performed in the remaining 10%. Although GP is in this case also able to reliably find the optimum this is only the case for a population size of 1000 whereas the EDA is still able to reliably find the optimum for all population sizes. Although lowering the probability of recombination normally speeds up convergence, in this case the EDA is only hampered by it because the distribution can perfectly describe the optimum and therefore using the distribution more frequently will improve performance. Moreover, copying introduces spurious dependencies that are estimated in the distribution and will additionally hamper optimization.

Figure 2 shows the results for the binary-functions problem (maximum fitness is 16). This problem has dependencies that are much harder for the distribution to adequately represent and reliably reproduce. Very large subfunctions are required to this end. Moreover, the multiple (sub)optima slow down convergence for the EDA as can be seen in Figure. 2 on the left. Crossover in GP is much more likely to reproduce large parts of parents. Hence, crossover automatically biases the search towards one of these solutions, allowing for faster convergence. The use of copying leads to faster convergence of the EDA. However, the deficiency of the distribution with respect to the dependencies in the problem still hamper the performance of the EDA enough to not be able to improve over standard GP. Additional enhancements may be required to make the distribution used more suited to cope with dependencies such as those encountered in the binary-functions problem, after which an improvement over standard GP may be expected similar to the improvement seen for the Royal Tree problem.

## 5 Discussion and conclusions

In this paper we have proposed a probability distribution over trees to be used in an EDA for GP. The distribution basically associates a probability with each production rule in a context-free grammar. More involved production rules or subfunctions can be introduced using transformations in which one production rule is expanded into another production rule. This allows the probability distribution to become more specific and to express a higher order of dependency. We have performed experiments on two benchmark problems from the literature. The results indicate that our EDA for GP is feasible. It should be noted however that learning advanced production rules using the greedy algorithm proposed in this paper can take up a lot of time, especially if the number of production rules and the arity of the subfunctions increase. To speed up this greedy process, only a single rule can be randomly selected into which to expand each pro-

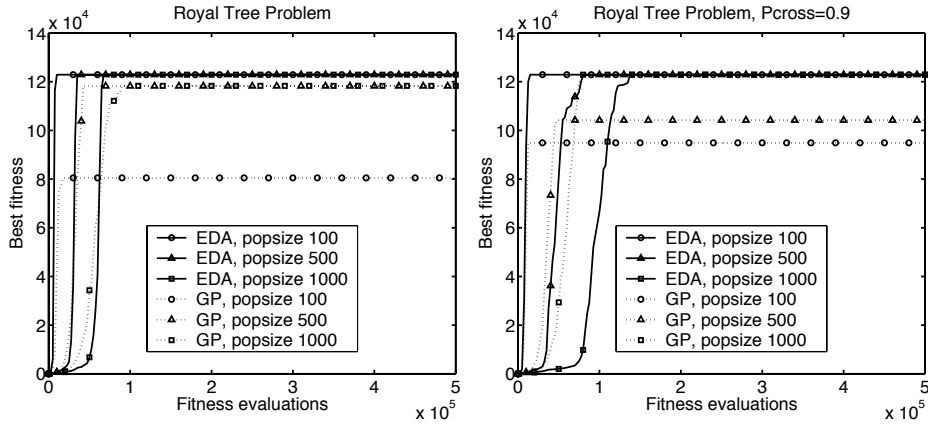


Fig. 1. Results for the Royal Tree problem.

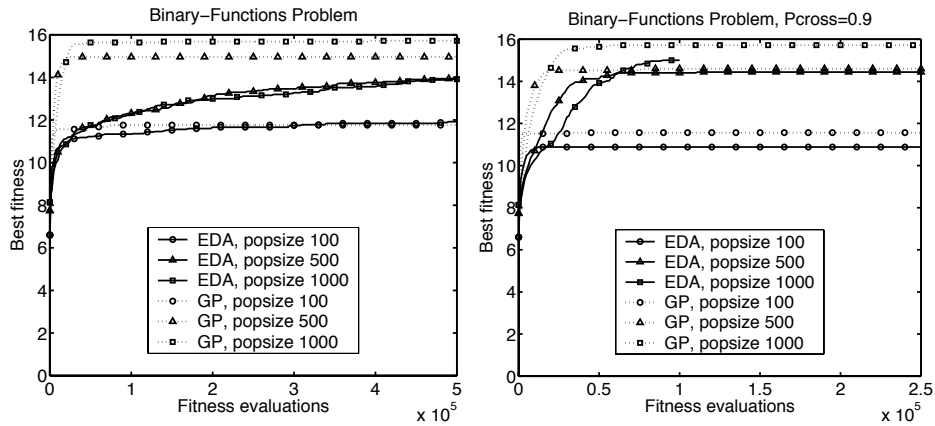


Fig. 2. Results for the binary-functions problem.

duction rule from the base grammar instead of expanding each production rule from the base grammar into each currently available production rule. Although this significantly reduces the number of candidate distributions in the greedy algorithm, it also significantly improves the running time. Moreover, since the greedy algorithm is iterative and the probability distribution is estimated anew each generation, the most important subfunctions are still expected to emerge.

Because our approach to estimating probability distributions over trees does not fix or bound the tree structure beforehand, our approach can be seen as a more principled way of identifying arbitrarily-sized important subfunctions than by constructing subfunctions randomly (e.g. GLiB [1]) or by evolving them (e.g. ADFs [8] and ARL [14]). As such, this paper may provide one of the first steps in a new and interesting direction for GP that allows to detect and exploit substructures in a more principled manner for enhanced optimization performance.

## References

1. Peter J. Angeline and Jordan B. Pollack. The evolutionary induction of subroutines. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 236–241, Hillsdale, NJ, 1992. Lawrence Erlbaum Associates.
2. Peter A. N. Bosman. *Design and Application of Iterated Density–Estimation Evolutionary Algorithms*. PhD thesis, Utrecht Univ., Utrecht, the Netherlands, 2003.
3. Peter A. N. Bosman and Edwin D. de Jong. Grammar transformations in an EDA for genetic programming. In M. Pelikan et al., editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO–2004*, Berlin, 2004. Springer–Verlag.
4. Wray Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions On Knowledge And Data Engineering*, 8:195–210, 1996.
5. Michael L. Cramer. A representation for the adaptive generation of simple sequential programs. In John J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 183–187, Hillsdale, NJ, 1985. Carnegie-Mellon University, Lawrence Erlbaum Associates.
6. Emin E. Korkmaz and Göktürk Üçoluk. Design and usage of a new benchmark problem for genetic programming. In *Proceedings of the 18th International Symposium on Computer and Information Sciences ISCIS–2003*, pages 561–567, Berlin, 2003. Springer–Verlag.
7. John R. Koza. *Genetic Programming*. The MIT Press, Cambridge, MA, 1992.
8. John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. The MIT Press, Cambridge, MA, May 1994.
9. Pedro Larrañaga and José A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic, London, 2001.
10. Martin Pelikan. *Bayesian optimization algorithm: From single level to hierarchy*. PhD thesis, University of Illinois at Urbana–Champaign, Urbana, Illinois, 2002.
11. Martin Pelikan, David E. Goldberg, and Fernando G. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.
12. William F. Punch, Douglas Zongker, and Erik D. Goodman. The royal tree problem, a benchmark for single and multiple population genetic programming. In P. J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, pages 299–316. The MIT Press, Cambridge, MA, USA, 1996.
13. Jorma Rissanen. Hypothesis selection and testing by the MDL principle. *The Computer Journal*, 42(4):260–269, 1999.
14. Justinian P. Rosca and Dana H. Ballard. Discovery of subroutines in genetic programming. In P.J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 9, pages 177–202. The MIT Press, Cambridge, MA, 1996.
15. Rafal P. Salustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.
16. Kumara Sastry and David E. Goldberg. Probabilistic model building and competent genetic programming. In Rick L. Riolo and Bill Worzel, editors, *Genetic Programming Theory and Practise*, chapter 13, pages 205–220. Kluwer, 2003.
17. Y. Shan, R.I. McKay, R. Baxter, H. Abbass, D. Essam, and H.X. Nguyen. Grammar model-based program evolution. In *Proceedings of the 2004 Congress on Evolutionary Computation – CEC2004*, Piscataway, New Jersey, 2004. IEEE Press.