UNIVERSITY OF DORTMUND

REIHE COMPUTATIONAL INTELLIGENCE

COLLABORATIVE RESEARCH CENTER 531

Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods

Distributed Hybrid Genetic Programming for Learning Boolean Functions

Stefan Droste Dominic Heutelbeck
Ingo Wegener

No. CI-90/00

Technical Report ISSN 1433-3325 August 2000

Secretary of the SFB 531 · University of Dortmund · Dept. of Computer Science/XI
44221 Dortmund · Germany

This work is a product of the Collaborative Research Center 531, “Computational Intelligence”, at the University of Dortmund and was printed with financial support of the Deutsche Forschungsgemeinschaft.

Distributed Hybrid Genetic Programming for Learning Boolean Functions

Stefan Droste*, Dominic Heutelbeck*, and Ingo Wegener*

FB Informatik, LS 2, Univ. Dortmund, 44221 Dortmund, Germany
{droste,heutelbe,wegener}@ls2.cs.uni-dortmund.de

Abstract. When genetic programming (GP) is used to find programs with Boolean inputs and outputs, ordered binary decision diagrams (OBDDs) are often used successfully. In all known OBDD-based GP-systems the variable ordering, a crucial factor for the size of OBDDs, is preset to an optimal ordering of the known test function. Certainly this cannot be done in practical applications, where the function to learn and hence its optimal variable ordering are unknown.
Here, the first GP-system is presented that evolves the variable ordering of the OBDDs and the OBDDs itself by using a distributed hybrid approach. For the experiments presented the unavoidable size increase compared to the optimal variable ordering is quite small. Hence, this approach is a big step towards learning well-generalizing Boolean functions.

1 Introduction

A major goal in genetic programming (GP) ([8]) is to find programs that reproduce a set of given training examples and have good generalizing properties. This means that the resulting program should closely resemble the output values of the underlying function for the inputs not included in the training set, too.

One approach to achieve this is the principle of Occam's Razor, i. e., one tries to find the simplest function that outputs the correct values for the training set. Here one assumes, that functions with small representations yield a better generalization. Using a result from learning theory it was shown in [6] how the generalization quality of small programs found by GP can be lower bounded. Because redundant code can make it very difficult to measure the size of S-expressions, ordered binary decision diagrams (OBDDs) were used in this approach having an easy to compute minimal representation, called reduced.

OBDDs, introduced in [3], have proved to be the state-of-the-art data structure for Boolean functions $f\colon\{0,1\}^n \mapsto \{0,1\}$, i. e., $f \in B_n$: on the one hand, they allow the representation of many important Boolean functions in size polynomial in n, on the other hand, many algorithms with polynomial runtime in the size of the OBDDs are known for manipulating OBDDs (see [14]).

* This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (531).

Because of these advantages, OBDDs have been used successfully in GP-systems in [15], [5], and [7]. But all these systems do not only base their runs on the given training set, but also on a known optimal variable ordering for the given benchmark function. The variable ordering has a crucial influence on the size of an OBDD, i. e., depending on it a function can have a polynomially- or exponentially-sized OBDD. Because in all former approaches only known test functions were used, the optimal variable ordering was known in advance, which is naturally not the case in practical applications. Because there are important functions like the multiplexer-function, where only a very small fraction of all variable orderings allows even a good approximation with polynomially sized OBDDs, it is necessary to adapt the variable ordering.

Here, we present the first GP-system, where the variable ordering and the OBDDs itself are evolved using a distributed hybrid approach. In the next two sections we formally define OBDDs and discuss the variable ordering problem and its implications for OBDD-based GP. Then we describe a new GP-system that uses well-known heuristics for the variable ordering problem and methods from distributed evolutionary algorithms. Finally, we present some empirical results, showing that the unavoidable loss in quality with respect to a system using the optimal variable ordering is quite small.

2 Ordered binary decision diagrams

Definition 1. *Let π be a permutation on $\{1,\dots,n\}$ (called variable ordering). A π-OBDD is a directed acyclic graph $O=(V,E)$ with one source and two sinks, labelled by the Boolean constants 0 and 1. Every inner node is labelled by one of the Boolean variables $x_1,\dots,x_n$ and has two outgoing edges leading to the 0- and 1-successor. If an edge leads from an x_i-node to an x_j-node, then $\pi^{-1}(i)$ has to be smaller than $\pi^{-1}(j)$, i. e., the edges have to respect the variable ordering.*

In order to evaluate the function f represented by a π-OBDD for an input $(a_1,\dots,a_n)\in\{0,1\}^n$, one starts at the source and recursively goes to the 0- resp. 1-successor, if the actual node is labelled by x_i and $a_i=0$ resp. $a_i=1$. Then $f(a)$ is equal to the label of the finally reached sink.

The size of O is the number of its inner nodes. An OBDD is a π-OBDD for an arbitrary π. An OBDD is reduced, if it has no node with identical 0- *and* 1-*successor and contains no isomorphic subgraphs.*

One can prove, that for a given $f\in B_n$ and a fixed π the reduced π-OBDD of f is unique up to isomorphism. Let $\{i_1,\dots,i_k\}\subseteq\{1,\dots,n\}$ be a set of indices and $a\in\{0,1\}^k$. The function $f_{|x_{i_1}=a_i,\dots,x_{i_k}=a_k}\colon\{0,1\}^{n-k}\mapsto\{0,1\}$ is the restriction of f, where for every $j\in\{1,\dots,k\}$ the variable x_{i_j} is set to a_j. A function $f\colon\{0,1\}^n\mapsto\{0,1\}$ depends essentially on x_i, iff $f_{|x_i=0}\neq f_{|x_i=1}$. One can show that a reduced π-OBDD representing $f\in B_n$ contains exactly as many nodes with label x_i, as there are different subfunctions $f_{x_{\pi(1)}=a_1,\dots,x_{\pi(j-1)}=a_{j-1}}$ depending essentially on x_i, for all $(a_1,\dots,a_{j-1})\in\{0,1\}^{j-1}$ with $j=\pi^{-1}(i)$, i. e., the size of a reduced π-OBDD is directly related to the structure of the function it represents.

Hence, the representation of Boolean functions by reduced OBDDs eliminates redundant code and automatically discovers useful subfunctions in a similar manner as automatically defined functions ([8]), as parts of the OBDD can be evaluated for different inputs. One receives these benefits without further expense by the application of reduced OBDDs for representation in a GP-system.

The problem we want to solve at least approximately is the following:

Definition 2. *In the minimum consistent OBDD problem we have as input a training set $T \subseteq \{(x, f(x)) \mid x \in \{0,1\}^n\}$ and want to compute the minimal π-OBDD, that outputs $f(x)$ for all x with $(x, f(x)) \in T$, over all variable orderings π, where $f\colon \{0,1\}^n \mapsto \{0,1\}$ is the underlying function.*

Certainly our main goal is to find the function f, but all we know about it is the training set T. Assuming that the principle of Occam's razor is valid for the functions we want to find, a solution to the minimum consistent OBDD problem would be a well generalizing function. In the next section we argue why the variable ordering is essential for the minimum consistent OBDD problem.

3 The variable ordering problem and OBDD-based GP

It is well-known, that the size of a π-OBDD depends heavily on π. A good example for this fact is the multiplexer-function, one of the major benchmark functions for GP-systems that try to learn Boolean functions:

Definition 3. *The multiplexer function on $n = k+2^k$ ($k \in \mathbb{N}$) Boolean variables is the function $MUX_n(a_0, \ldots, a_{k-1}, d_0, \ldots, d_{2^k-1}) = d_{|a|}$, where $|a|$ is the number whose binary representation is $(a_0, \ldots, a_{k-1})$.*

If π orders the variables as $a_0, \ldots, a_{k-1}, d_0, \ldots, d_{2^k-1}$, the OBDD for MUX_n has size $2^k - 1 + 2^k$, i. e., linear in n. But for the reverse order the OBDD for MUX_n has size at least $2^{2^k} - 1 = \Omega(2^n)$, i. e., exponential in n. For an example see Figure 1. Furthermore, MUX_n is almost ugly, i. e., the fraction of variable orderings leading to non-polynomially-sized OBDDs converges to 1 ([14]). So randomly choosing π will lead to non-polynomial π-OBDD size with high probability for large n.

Trying to exactly compute an optimal variable ordering is not a choice, since the computation of an optimal variable ordering is NP-complete ([2]) and even finding a variable ordering π, such that the size of the resulting π-OBDD approximates the optimal size over all variable orderings up to a constant factor, cannot be done in polynomial-time, if $NP \neq P$ ([12]).

Considering a GP system that searches for small OBDDs fitting a random set of training examples for MUX_n, [9] provide the following theorem:

Theorem 1. *For every large enough $n = k+2^k$, if we choose $m = k^{\Theta(1)}$ training examples of MUX_n under the uniform distribution and choose a random ordering π of the variables, then with probability at least $1 - k^{-1/2}$, there is no π-OBDD of size $\frac{1}{10} m / \log m$ matching the given training examples.*

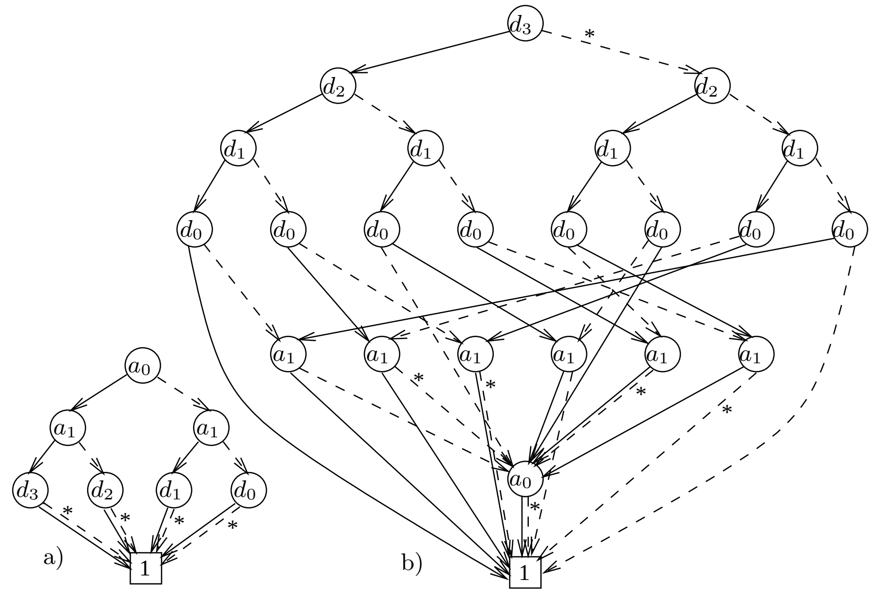

Fig. 1. Two OBDDs representing the function MUX_6, where the edges to 0-successors are dotted and complemented edges are marked by a $*$. a) With an optimal variable ordering. b) With a bad variable ordering.

This theorem implies, that if we start a OBDD-based GP run even with a random variable ordering with high probability it is impossible to find a small OBDD matching the training examples. In contradiction to the early OBDD-based GP-systems we consider the knowledge of the optimal variable ordering as unknown, since the cited results show that it is hard to obtain, even if the complete function is known. Hence, we have to optimize the variable ordering during the GP run. A possibility to do this is presented in the next section.

4 A distributed hybrid GP-system

Because one has to optimize the variable ordering during the GP run to approximately solve the minimum consistent OBDD problem, we have to decide how to do this in our GP-system. The usual GP approach would be to add independent variable orderings to each individual and to tailor the genetic operators to this new representation. But since this approach would imply the reordering of OBDDs in almost every single genetic operation, this would lead to inefficient genetic operators with exponential worst-case run-time. For OBDDs of the same variable ordering we know efficient genetic operators ([5]). Hence, we should try to do as few reorderings as possible without losing too much genetic diversity.

Therefore, we use a distributed approach similar to the distributed GA from [13]. In our approach all OBDDs in a subpopulation have the same variable ordering, but the subpopulations can have different ones. This fact allows us to use efficient genetic operators in the subpopulations. When migration between the subpopulations occurs every M-th generation, the migration strategy decides how to choose the new variable ordering of each subpopulation. In order to exchange good individuals as fast as possible we use a completely connected topology, i. e., every subpopulation sends migrants to every other subpopulation.

Because this alone would limit the number of variable orderings to the number of subpopulations, every N-th generation we use a heuristic to optimize the variable ordering in each subpopulation separately. The following heuristics are suitable for our setting: sifting ([11]), group sifting ([10]), simulated annealing ([1]), and genetic algorithms ([4]).

In order to exactly describe our algorithm, we make the following definitions:

Definition 4. a) *Let $\mu \in \mathbb{N}$ be the population size and $\lambda \in \mathbb{N}$ the number of offspring.*

b) *Let $I = \{i_1, \ldots, i_k\}$ be a multi-set of k subpopulations, where $i_j = \{O_1^j \ldots, O_\mu^j\}$ is a multi-set with π_j-OBDDs O_l^j for all $j \in \{1, \ldots, k\}$ and $l \in \{1, \ldots, k\}$.*

c) *Let $B \in \mathbb{N}$ be the migration rate, $M \in \mathbb{N}$ the length of the migration interval, and $N \in \mathbb{N}$ the length of the reordering interval.*

d) *Let In_j $(1 \le j \le k)$ be the lists of incoming OBDDs.*

Then the rough outline of our GP-system is as follows, where all sets of individuals are multi-sets, i. e., we allow duplicates:

Algorithm 1 *Distributed Hybrid GP-system*
Input: *the training set T.*

1. ***Initializiation:*** *Choose a uniformly distributed variable ordering π_j and a random initial population $i_j = \{O_1^j, \ldots, O_\mu^j\}$ for all $j \in \{1, \ldots, k\}$. Set $g = 0$.*
2. ***Reordering:*** *If $g \bmod N \equiv 0$: Execute the chosen variable ordering optimization heuristic on i_j for every $j \in \{1, \ldots, k\}$.*
3. ***Generation of offspring:*** *For every $j \in \{1, \ldots, k\}$ generate λ offspring $O_{\mu+1}^j, \ldots, O_{\mu+\lambda}^j$ from i_j by doing mutation resp. recombination with probability p resp. $1 - p$.*
4. ***Selection:*** *For every $j \in \{1, \ldots, k\}$ let i_j be the μ individuals with the highest fitness values from $O_1^j, \ldots, O_{\mu+\lambda}^j$.*
5. ***Selection of migrants:*** *If $g \bmod M \equiv 0$: For every $j \in \{1, \ldots, k\}$ set $In_j = \emptyset$. For every $j \in \{1, \ldots, k\}$ and $j' \neq j$ choose a set of B individuals $a = \{a_1, \ldots, a_B\}$ fitness proportionally from $i_j = \{O_1^j, \ldots, O_\mu^j\}$ and set $In(i_{j'}) = In(i_{j'}) \cup a$.*
6. ***Calculate new variable ordering:*** *If $g \bmod M \equiv 0$: For every $j \in \{1, \ldots, k\}$ let $\pi_j = migration\_strategy(j)$.*
7. ***Migration:*** *If $g \bmod M \equiv 0$: For every $j \in \{1, \ldots, k\}$ let $In_j = \{a_1, \ldots, a_\nu\}$ for $\nu = B \cdot (k-1)$, delete ν randomly under the uniform distribution chosen individuals in i_j and insert the π_j-OBDD $O_{\mu+\lambda+l}^j$ with $f_{O_{\mu+\lambda+l}} = f_{a_l}$ for every $l \in \{1, \ldots, \nu\}$ into i_j.*
8. ***Main loop:*** *Set $g = g + 1$ and go to step 2, until $g > G$.*
9. ***Output:*** *Output the smallest consistent OBDD of the last generation.*

Now we describe the different steps of our GP-system in more detail. Because initialization and recombination are based on [5], more details can be found there.

4.1 Representation

In the j-th subpopulation all individuals are represented by reduced π_j-OBDDs. Because they all share the same variable ordering, we use a reduced shared binary decision diagram (SBDD) for every subpopulation. An SBDD representing $g\colon \{0,1\}^n \mapsto \{0,1\}^m$ is an OBDD with m sources representing the coordinate functions of $g = (g_1, \ldots, g_m)$. In OBDD-based GP the population of m reduced π-OBDDs representing functions $g_1, \ldots, g_m$ from B_n is identified with the multi-output function $g = (g_1, \ldots, g_m)$ and stored in the reduced π-SBDD representing g. By doing this, the individuals share isomorphic sub-graphs. Experiments have shown, that in an OBDD-based GP-system the SBDD size will go down fast in comparison to the sum of the sizes of the OBDDs, because the individuals become more and more similar. This observation still holds, if one avoids duplicates.

Furthermore, we use OBDDs with complementary edges for representation, which allow memory savings of a factor up to two by using a complementary flag bit for every edge labelled by 0 and pointers referencing the sources. During evaluation a complementary flag displays if the referenced subOBDD is negated. Hence, to represent a subfunction and its complement only one subOBDD is necessary, whereby we only consider OBDDs with a 1-sink. By using the OBDD-package CUDD all our π-SBDDs are reduced and use complementary edges.

These are syntactic aspects of our representation, but we also make a semantic restriction: we only use OBDDs that are consistent with the given training set, i. e., have the correct output values for the training set T. This is done by explicitly creating consistent OBDDs during initialization and testing new offspring to be consistent, otherwise replacing them by one of the parents. This method reduces the size of the search space by a factor of $2^{|T|}$, allowing us to measure the fitness of an individual by its size only. This was shown empirically in [5] to be advantageous for test problems of our kind.

4.2 Initialization

While the variable orderings π_j are chosen independently using the uniform distribution from all possible permutations of $\{1, \ldots, n\}$, the π_j-OBDDs itself are created as follows (for easier notation we assume that $\pi_j = id$): starting from the source with label x_1, for every actual node labelled x_i the number of different outputs of the training examples consistent with the path to the actual node is computed, where a path is identified with its corresponding partial assignment of the variables. If the number of these outputs is two, the procedure is called recursively to create the 0- and 1-successor with label x_{i+1}; if it is one or zero, a random subOBDD is returned by creating the 0- and 1-successor with labels $x_{i+\delta_0}$ and $x_{i+\delta_1}$, where δ_0 and δ_1 are geometrically distributed with parameter α. If $i + \delta_0$ resp. $i + \delta_1$ is at least $n + 1$ the corresponding sink is returned, or a random one, if the actual path is not consistent with any training example.

Thus, the way consistent OBDDs are randomly generated is influenced by the parameter α: for $\alpha = 1$ the resulting function is uniformly distributed from all functions being consistent with the training set, for $\alpha = 0$ every path being not consistent with the training set leads to a randomly chosen sink via at most one additional inner node.

4.3 Reordering

All our heuristics for optimizing the variable ordering work on the whole π_j-SBDD representing the j-th subpopulation. If we would apply the heuristic on every π_j-OBDD we would eventually get smaller OBDDs, but then the subpopulation would consist of OBDDs of different variable orderings. Hence, we apply the chosen heuristic on the whole SBDD. Hereby we hope to achieve an approximation of the optimal variable orderings of the individual OBDDs. To see how a heuristic can look like, we give a short description of sifting ([11]):

First all variables are sorted according to the number of nodes in the SBDD labelled by it. Then, starting with the variable with the lowest number, the variable is stepwise swapped with its neighbours: first to the near end of the variable ordering and then to the far end. Because the SBDD-size after such a swap can be computed quite efficiently, the variable is put to the position where the SBDD size was minimal. This procedure is repeated for the variable with the second-lowest number and so on. To avoid blow-up, this process is stopped if the SBDD-size grows beyond a factor of c (we choose $c = 2$).

4.4 Recombination

In recombination and mutation the parents are chosen proportionally to the normalized fitness $1/(1 + s)$, where s is the size of the OBDD. Recombination of two π_j-OBDDs chooses uniformly a node v_a in the first parent and then a node v_b in the second parent from all nodes having a label at least that of v_a according to π_j. Then the subOBDD starting with v_a is replaced by the subOBDD starting with v_b: As there can be many paths to v_a, we choose one of the paths randomly and replace the edge to v_a by an edge to v_b. For all other paths to v_a we replace the edge to v_a with probability 1/2, hence considering the role of shared subOBDDs as ADFs. If this new OBDD is not consistent with the training set, it is replaced by one of its parents. If the offspring is already in the population, this procedure is repeated up to ten times, to avoid duplicates.

4.5 Mutation

For mutating a π_j-OBDD a node v_a of the OBDD to be mutated is chosen randomly under uniform distribution. For a path leading to v_a it is checked, if its 0- or 1-successor is relevant for consistency with the training set. If not, it is replaced by the other successor. Otherwise, a π_j-OBDD with source v_b is created randomly using the same algorithm as was applied during initialization for an empty training set, where all nodes have a label at least v_a with respect to π_j. On one randomly chosen path to v_a the edge to v_a is replaced by an edge to v_b. On all other paths this is done with probability 1/2. Again, a not-consistent offspring is replaced by its parent and this procedure is repeated up to ten times, if the offspring is already in the population.

4.6 Migration strategy

The migration strategy decides how to choose the new variable ordering of the j-th population after migration has taken place. Because changing the variable ordering can cause an exponential blow-up, we choose an introverted strategy by changing the variable orderings of all incoming OBDDs to the variable ordering of the jth subpopulation, i. e., $\mathit{migration\_strategy}(j) = \pi_j$.

5 Experimental results

For our experiments we use only the multiplexer-function, because we know by Theorem 1 that it is one of the hardest functions, when it comes to finding small OBDDs that approximate it or even a random sampling of it. So we want to see if our GP-system is capable of finding small OBDDs, where the inputs of the training set are randomly and independently chosen for every run. Furthermore, we are interested in the generalization capabilities of the OBDDs found. Hence, we also investigate the fraction of all inputs, where the smallest OBDD found has the same output as the multiplexer function. We emphasize that no knowledge whatsoever of the multiplexer function influences our GP-system.

Number of subpopulations	$k = 4$	Size of subpopulations	$\mu = 40$
Number of generations	$G = 3000$	Migration rate	$B = 5$
Length of migration interval	$M = 10$	Length of reordering interval	$N = 20$
Reordering heuristic	group sifting	Initial size parameter	$\alpha = 0.2$
Mutation probability	$p = 0.1$	Size of training set	$T = 512$

Fig. 2. Parameter settings for our experiments

The parameters of our GP-system in the experiments are set as shown in Figure 2, where the size of the training set is chosen to match the examples of older OBDD-based GP-systems. The results are shown in Figures 3 and 4, where we choose $n = 20$ and average over 10 runs. We compare our GP-system with three GP-systems, where the variable ordering is fixed to an optimal and a random variable ordering. These systems use only one population of size 160 and no variable ordering heuristic, but the same genetic operators as in our system.

We see in Figure 3, that our GP-system, although being worse than the system using an optimal variable ordering, produces smaller OBDDs than using a random variable ordering: after 3000 generations the average size of the smallest OBDD found is 126.97 in comparison to 185.10 (where a minimal OBDD for MUX_{20} has size 32, as we also count the sink here). Taking the results from Figure 3 and Figure 4 we see that Occam's razor seems to be valid for MUX_n, because the generalization capabilities of the found OBDDs behave according to their sizes: while using the best variable ordering by far results in the best generalization capabilities, our GP-system with sifting outperforms a fixed GP-system with fixed random variable ordering (56.73% in comparison to $54,72\%$

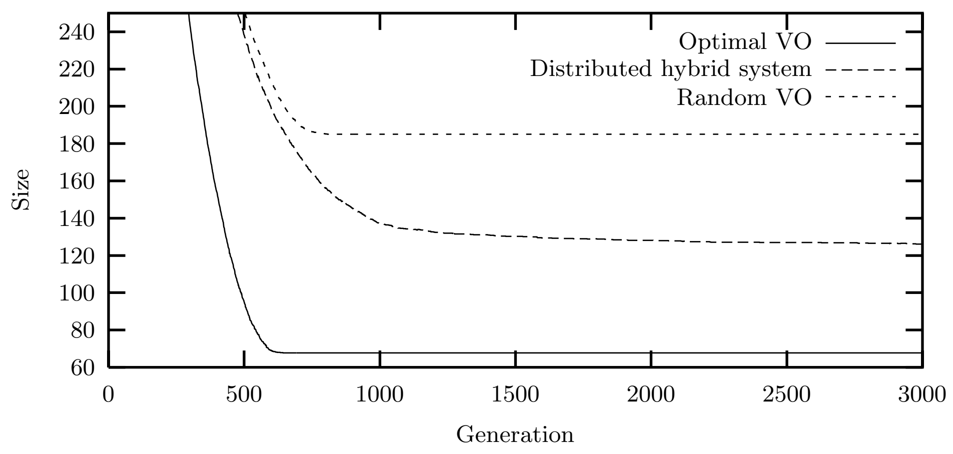

Fig. 3. Sizes of the smallest OBDDs per generation (over 10 runs).

hits after 3000 generations). But one can also notice, that our system is more capable to reduce the size of the OBDDs than to increase the hit rate. One could conclude that the excellent hit rates of the previous OBDD-based GP-systems are based on the information about the variable ordering.

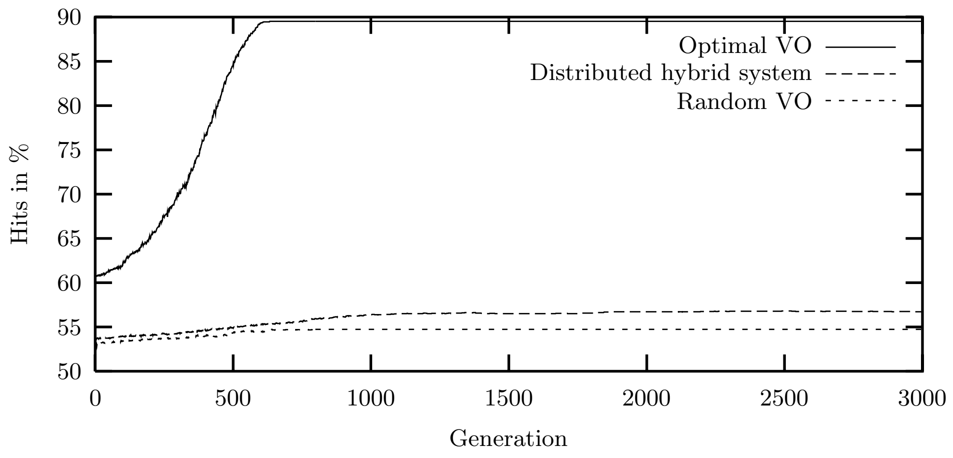

Fig. 4. Hits of the smallest OBDDs per generation compared to the underlying function of the training set (over 10 runs).

Hence, our distributed hybrid GP-system empirically improves static approaches using a random variable ordering. The static approach with the optimal variable ordering allows no fair comparison, as in practical applications the variable ordering is unknown and even approximations are hard to compute.

6 Conclusion

OBDDs are a very efficient data structure for Boolean functions and are therefore a succesfully used representation in GP. But every OBDD-based GP-system so far uses the additional information of an optimal variable ordering of the function to learn, which is only known if the test function is known and has strong influence on the size of the OBDDs. Hence, we presented a distributed hybrid GP-system, that for the first time evolves the variable ordering and the OBDDs itself. Empirical results show that this approach is advantageous to a GP-system, where the variable ordering is randomly fixed, and also more successful than a simple hybrid approach, in which the number of subpopulations is set to one and the population size is set to 160. Hence, this is a great step towards the practical applicability of GP-systems with OBDDs, since there is no additional neccesary input needed beside the training set.

References

1. B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. In *Int. Workshop on Logic Synthesis*, pages 5.1–5.10, 1995.
2. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Computers*, 45:993–1002, 1996.
3. R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.
4. R. Drechsler, B. Becker, and N. Göckel. A genetic algorithm for variable ordering of OBDDs. In *Int. Workshop on Logic Synthesis*, 1995.
5. S. Droste. Efficient genetic programming for finding good generalizing boolean functions. In *Genetic Programming 1997*, pages 82–87, 1997.
6. S. Droste. Genetic programming with guaranteed quality. In *Genetic Programming 1998*, pages 54–59, 1998.
7. S. Droste and D. Wiesmann. Metric based evolutionary algorithms. In *Third European Conference on GP (EuroGP2000)*, pages 29–43, 2000.
8. J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.
9. M. Krause, R. Savický, and I. Wegener. Approximations by OBDDs and the variable ordering problem. In *Int. Colloquium on Automata, Languages, and Programming (ICALP99)*, pages 493–502, 1999. LNCS 1644.
10. S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Proceedings of the International Conference on CAD*, pages 74–77, San Jose, CA, 1995.
11. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *IEEE/ACM Int. Conference on CAD*, pages 42–47, 1993.
12. D. Sieling. *The nonapproximability of OBDD minimization.* Univ. Dortmund, 1998. Technical report (Submitted to Information and Computation).
13. R. Tanese. Distributed genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439, 1989.
14. I. Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications.* SIAM, 2000. (In print).
15. M. Yanagiya. Efficient genetic programming based on binary decision diagrams. In *IEEE Int. Conf. on Evolutionary Computation (ICEC95)*, pages 234–239, 1995.