International Conference on Computational Intelligence (Fuzzy Days'2001)
Dortmund, October 2001

Too Much Knowledge Hurts: Acceleration of Genetic Programs for Learning Heuristics

Frank Schmiedle, Daniel Große, Rolf Drechsler, and Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg, Germany
{schmiedl,grosse,drechsle,becker}@informatik.uni-freiburg.de

Abstract. Among many other applications, evolutionary methods have been used to develop heuristics for several optimization problems in VLSI CAD in recent years. Although learning is performed according to a set of training benchmarks, it is most important to generate heuristics that have a good generalization behaviour and hence are well suited to be applied to unknown examples. Besides large runtimes for learning, the major drawback of these approaches is that they are very sensitive to a variety of parameters for the learning process.
In this paper, we study the impact of different parameters, like e.g. stopping conditions, on the quality of the results for learning heuristics for BDD minimization. If learning takes too long, the developed heuristics become too specific for the set of training examples and in that case results of application to unknown problem instances deteriorate. It will be demonstrated here that runtime can be saved while even improving the generalization behaviour of the heuristics.

1 Introduction

Over the past 10 years, evolutionary approaches, e.g. *Genetic Algorithms (GAs)* and *Genetic Programming (GP)*, have become more and more popular [3, 10, 11] and they have been successfully applied to many optimization problems, particularly in the area of VLSI CAD (e.g. [5]).

Instead of directly solving an optimization problem by GAs, in [6] a model has been proposed where GAs are used to learn good heuristics that can be applied to problem instances after the learning phase. Recently, this model has been generalized to the use of the more powerful and flexible GP instead of GAs in [4]. A set of *Basic Optimization Modules (BOMs)* is given and a heuristics includes a combination of some of the BOMs. In GAs, heuristics are always sequences of BOMs. In contrast to this, "conditional" BOM sequences can be represented by the heuristics as well if GP is used. The genetic program uses a set of training examples to learn good heuristics and during evolution, the *fitness* of the heuristics is determined by the results when being applied to the training set.

Since the fundamental component in *Heuristic Learning* is the incorporated GA or GP, respectively, the learning process is based on simulation. Therefore, often runtimes are very large. Moreover, the results are largely dependent on parameter settings of the evolutionary optimization method. Parameters like stopping conditions or the starting points for the evolution (=the initial population) can have strong impact on the runtimes and as well on the quality of the resulting heuristics.

Different from the fitness, the *quality* of a heuristics is determined by its generalization behaviour, i.e. the results obtained when being applied to problem instances that do not belong to the training set. A larger fitness value does not always imply a higher quality. If the GP is directed too much towards "exploitation" of the training set by parameter settings, in the final stage of evolution improving the fitness can result in a decrease of quality. The reason for this is that in such cases development of heuristics focuses only on the results obtained by application to training set examples, while generalization abilities cannot be taken into consideration.

In this paper, we analyze the impact of parameters like reproduction rate, multi-objective optimization type, start points and stopping conditions on the quality of the results for the problem of learning heuristics for BDD minimization by variable re-ordering. This problem is suitable for our study since it is of high practical importance. Moreover, it has been studied intensively and therefore in the meantime it is well understood.

The best results for BDD minimization so far have been obtained by using evolutionary approaches, but runtimes are very large. We demonstrate for heuristic learning based on GP that in many cases, slight learning improvements come with decreasing quality of the resulting heuristics at a late stage of evolution. Consequently, about 50% of the runtime can be saved by using better stopping criteria while at the same time, the quality of the heuristics can be increased.

2 Preliminaries

We assume that the reader is familiar with GA and GP concepts and refer to [3, 10] for details. A model for heuristic learning with GA has been proposed in [6]. It has been applied to various optimization problems in VLSI CAD. Recently, a generalization to the GP domain has been reported and and the GP-based heuristic learning has been successfully applied to BDD minimization for small benchmark circuits [4]. Since in this paper, various parameters and their impact on resulting heuristics are investigated in order to optimize the learning process, we give a brief review of heuristic learning and also describe how this approach is used for GP-based BDD minimization. By this, the paper becomes self-contained. Theoretic aspects are discussed for both GA and GP, but in all experiments, the GP-based method has been used.

2.1 Heuristic Learning

For learning heuristics in order to find good solutions to an optimization problem, it is necessary that several (non-optimal) strategies to solve the problem can be provided. Typically, for different classes of problem instances there are also different strategies that perform best. A strategy that behaves well on one problem class may return poor results when being applied to another problem instance that belongs to some different class. Thus it is promising to learn how to combine the strategies to heuristics that can be applied successfully to most or even all classes of problems.

The learning process is controlled either by GA or by GP. For a better understanding, some fundamental terms are introduced by the following

Definition 1. *Given an optimization problem P and a non-empty set of different non-optimal strategies $B = \{b_1, \ldots, b_{max}\}$ to solve the problem. Then the elements of B are called* BOMs (Basic Optimization Modules). *A* heuristics *for P is*

- *a non-empty sequence of BOMs in the case of learning by GA.*
- *an algorithm that can generate a sequence of BOMs if GP is used for heuristic learning.*

Furthermore, a flat sequence *of a heuristics is a concrete BOM sequence that has been calculated by the heuristics. If the heuristics has been generated by GA, its only flat sequence is the heuristics itself.*

Example 1. For an optimization problem P, a set of BOMs $B = \{A, B, C, D, E\}$ is given. A heuristics h which has been learned during a GP-driven evolutionary process is illustrated in Figure 1. Possible flat sequences of h are e.g. $ABDE$, $ACDE$ and $ABDDDDE$.

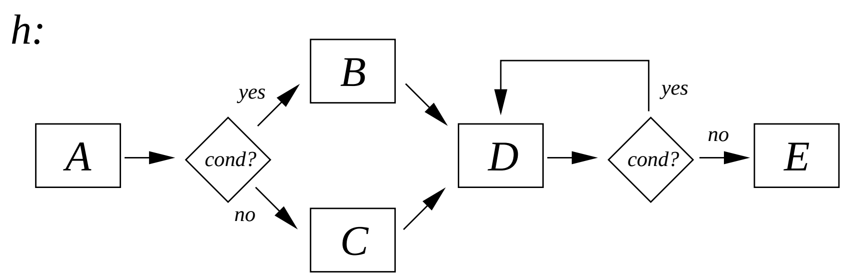

Fig. 1. Example for a heuristic generated during GP-based heuristic learning

Heuristics can be applied to p (an instance of P) by subsequently calling the BOMs included in the flat sequence to p. If GP is used, the flat sequence is generated on-the-fly during application of the heuristics (see Section 2.2).

During evolution, the strategies/BOMs are combined to generate heuristics that are the individuals in the population. (The fitness of) an individual can be evaluated by application of the heuristics to a training set of problem instances. In the learning phase, the target is that heuristics perform well according to some given optimization criteria when being applied to the training set. But for the success of the overall method, mainly a good generalization is important, i.e. the intention is that a heuristics that returns good results for the training set examples also performs well on unknown instances of P. Please note that for this, the handling of the different criteria, i.e. the special multi-objective optimization approach, plays a critical role for the success of the evolutionary process.

A scheme for the scenario of heuristic learning is outlined in Figure 2 where learning by GA and also learning by GP are included.

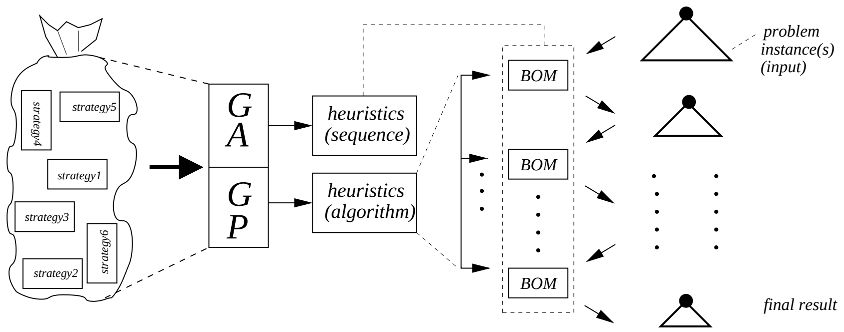

Fig. 2. Heuristic learning process

On the left, there is a set of problem solving strategies — this is the set of BOMs that are provided. Although GA *and* GP are included in the figure, only one of them is used for heuristic learning at the same time. In both cases, the result consists of one or several heuristics that can be applied to instances of P as illustrated at the right side of the figure. When a heuristics h is applied, the slots denoted by BOM are filled by elements of B. Problem instances are given as triangles[1] in the figure. Calling some BOM b_i returns an intermediate result that is passed to b_j which is the next one in the flat sequence. This is proceeded until the last BOM in the flat sequence of the heuristics terminates. As already mentioned, heuristics produced by GA are identical to their flat sequence. How flat sequences are generated for heuristics learned by GP is described in the following section.

[1] They can be considered e.g. as BDDs that have to be minimized.

2.2 BDD Minimization by Genetic Programming

Binary Decision Diagrams (BDDs) [2] are a state-of-the-art data structure often used in VLSI CAD for efficient representation and manipulation of Boolean functions. BDDs suffer from their size being strongly dependent on the variable ordering that is used. In fact, BDD sizes may vary from linear to exponential for different orderings. Optimization of variable orderings for BDDs is difficult [1], but nevertheless, strategies for BDD minimization that are based on dynamic variable ordering, i.e. *sifting* have been reported e.g. in [8, 13, 12].

For BDD minimization by heuristic learning, we use the GP-based approach that has been introduced in 2.1. The strategies sifting (SIFT), group sifting (GROUP), symmetric sifting (SYMM), window permutation of size 3 and 4, respectively, (WIN3, WIN4) serve as BOMs. For all these techniques there is an additional BOM that iterates the method until convergence is reached. These BOMs are denoted by the appendix CO, e.g. iterated sifting is denoted by SIFTCO. The "empty" operator called NOOP completes the set of BOMs B and hence $|B| = 11$.

The individuals in the GP consist of trees with leaf nodes labeled with BOMs and inner operator nodes that belong to the three different types CONCAT, IF and WHILE. During evaluation of the heuristics, the tree is traversed by a *depth-first-search*-based method in order to generate a flat sequence. The types of the inner nodes decide if

- both subtrees are evaluated subsequently (CONCAT),
- according to the truth value of a given condition either the left or the right son is considered (IF) or
- evaluation of the sons is iterated until a truncation condition is fulfilled (WHILE)

For recombination, two crossover operators are provided. While CAT concatenates two parents, the more sophisticated MERGE does the same for subtrees of the parents and by that, bloating can be prevented. In addition to this, there are four mutation operators that exchange BOMs (BMUT), node types (CIMUT, CWMUT) and modify conditions of IF-nodes (IFMUT), respectively. A probability table, i.e. Table 1, determines the frequencies for using the different recombination operators.

Table 1. Probabilities for recombination operators

Operator	CAT	MERGE	BMUT	CIMUT	CWMUT	IFMUT
Probability	5	45	20	10	10	5

An additional problem specific operator REDUCE is applied to each individual during evaluation. It eliminates redundant parts from the trees and hence speeds up the learning process. For evaluation of the heuristics, a training set of small

BDD benchmarks is used. Heuristics are subsequently applied to each of the benchmarks in that training set and the resulting sizes are the optimization criteria. To calculate the fitness of an individual, the multi-objective optimization techniques proposed in [14] are used.

3 Influence and Selection of Parameters

For the method introduced in Section 2, a standard algorithm for the evolutionary loop has been used [7, 9]. The algorithmic scheme of GA/GP leaves still some degrees of freedom to the user like e.g. generation of the initial population. In the following sections, such issues are addressed and the appropriate parameters and subroutines are discussed.

3.1 Initial Population

The initial population defines the point in the search space where evolutionary optimization is started. If by some problem specific method for generation of the initial population, a starting point can be found that is not so far from the optimum, it is likely that the optimum will be found early during the optimization process. Furthermore, the risk of getting stuck in a local optimum is also decreased in that case.

If on the other hand the starting point is close to some local optimum O_l, optimization will be directed towards O_l and with high probability it will not be possible to escape from O_l again. E.g. this happens if the individuals included in the initial population are too similar to each other.

As a consequence, for the problem of BDD minimization by GP-based heuristic learning, this means that at least the types IF and WHILE of inner nodes have to be part of some of the initial individuals. CONCAT nodes can be inserted, too, but it is not so essential that they occur in the beginning since such nodes are frequently inserted by the recombination operators during evolution. Besides the different node types, a large variety of BOMs should be included as well. By randomized insertion of BOMs into individuals, similarity can be avoided.

To determine the best suited structure of the trees that represent the initial individuals is a non-trivial problem and furthermore it is not obvious whether or not this structure should differ for different individuals of the initial population. To investigate this, we implemented the 3 approaches FIXED1, FIXED2 and VARIED that are explained in the following:

- FIXED1: The individuals in the initial population have a unique structure with only one level of inner nodes. All trees consist of 3 nodes: the root and its 2 sons that are leafs. The leaves are labeled with BOMs that are randomly selected out of B. The type of the root node is CONCAT in 50% of the cases, IF and WHILE are assigned to 25% of the root nodes each.
- FIXED2: All the individuals have 2 levels of inner nodes and 3 leaf nodes. Figure 3 illustrates 5 different classes of trees and each class is used for 20%

of the initial population. Analogously to the FIXED1 method, the BOMs are selected at random.

- VARIED: Here, the number of levels in the trees is arbitrary while only the *number of BOMs* in a heuristics is limited. A tree of infinite size with unlabeled nodes is traversed in a breadth first search manner, and each node is labeled with either IF (probability 20%), WHILE (15%) or BOM (65%). If a BOM is chosen, it is selected out of the set B at random and the sub-trees of the node are eliminated. When the BOM limit is reached, all following nodes are labeled with NOOP and their sub-trees are removed. The individual is complete when there are no unlabeled nodes any more.

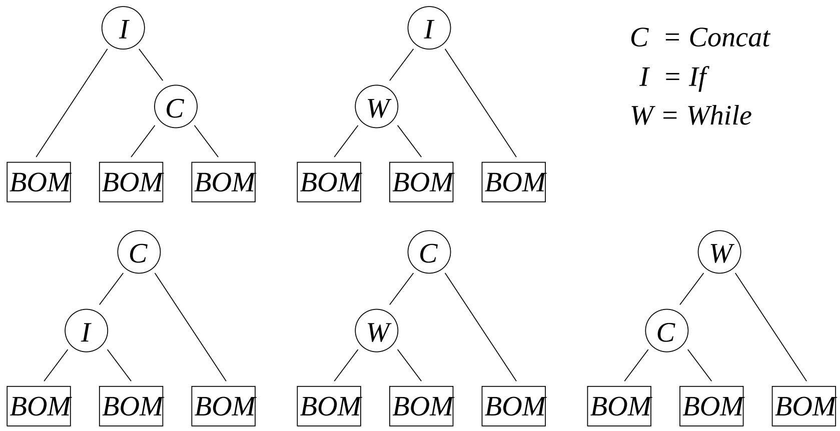

Fig. 3. Classes of trees for FIXED2

3.2 Training Set

The training set is needed during evolution for evaluation of the individuals' fitness. The heuristics represented by an individual is subsequently applied to each of the training set examples and based on the resulting BDD sizes, the fitness value of the heuristics is calculated. During the learning phase, search is directed towards individuals with good fitness values, i.e. optimization is done by trying to find heuristics that produce best results for the set of training examples.

Different from the fitness, the *quality* of a heuristics is determined by its generalization behaviour, i.e. the results obtained when being applied to benchmarks that the training set does not include. Since the overall task of heuristic learning is to find heuristics with good quality, it is essential that the training set is chosen in a way that keeps the average of the difference between fitness and quality as small as possible. As a consequence, a good training set has to be *representative* which means that the properties of each possible benchmark circuit have to be represented by one or several elements of the training set.

We use training sets that consist of 8 elements each. The training examples are benchmark circuits that are taken from LGSYNTH91 [15]. For all training example candidates, we measured the initial BDD size, i.e. the number of BDD nodes that was needed to construct the BDD without further minimization steps by one of the BOMs. Then we selected benchmarks with varying initial size and numbers of inputs/outputs in order to have more representative training sets. In total, 3 different training sets T_1, T_2 and T_3, respectively, have been used. Names, numbers of inputs/outputs and initial sizes of the benchmarks included in these sets are listed in Table 2.

Table 2. Training sets for heuristic learning

Training Set T_1				Training Set T_2				Training Set T_3			
Name	#in	#out	init size	Name	#in	#out	init size	Name	#in	#out	init size
bc0	26	11	589	*term1*	34	10	579	*alu4*	14	8	1181
ex7	16	5	172	*b9*	41	21	177	*s208.1*	18	9	1032
frg1	28	3	203	*count*	35	16	233	*s832*	23	24	2650
ibm	48	17	835	*x4*	94	71	890	*cm151a*	12	2	510
in2	19	10	2360	*s820*	23	24	2650	*i7*	199	67	504
s1196	32	31	2293	*i9*	88	63	2277	*apex6*	135	99	2759
ts10	22	16	4390	*vda*	17	39	4344	*example2*	85	66	468
x6dn	39	5	274	*s386*	13	13	280	*c432*	36	7	1732

3.3 Stopping Condition

During heuristic learning, a stopping condition is checked each time a new generation (of individuals) has been calculated. If the condition evaluates to *true*, then the evolutionary learning process is terminated.

Up to a certain number of generations g_{sat}, the computation of a larger number of generations leads to an increased fitness in most cases. But a saturation of fitness may be reached if g_{sat} is exceeded. Then the fitness of the best individual does not increase significantly in the following generations and therefore runtime is wasted if computation is continued. Even worse, a better fitness does not always imply a higher quality. Especially at a late stage of evolution, improvements of the fitness can result in a decrease of the quality which is the overall optimization task. The reason for this is that development of heuristics only focuses on the training set examples while the quality on other instances is not considered. Hence, overfitting on the training set can be observed in such cases.

In Figure 4, a progression of fitness and quality during evolution can be seen and in this case, overfitting can be observed. After g_{sat} is exceeded, only a slight increase of fitness can be obtained. Regardless of the fitness, evolution should be stopped no later than g_{opt}, since afterwards, the quality of the individuals in the population decreases.

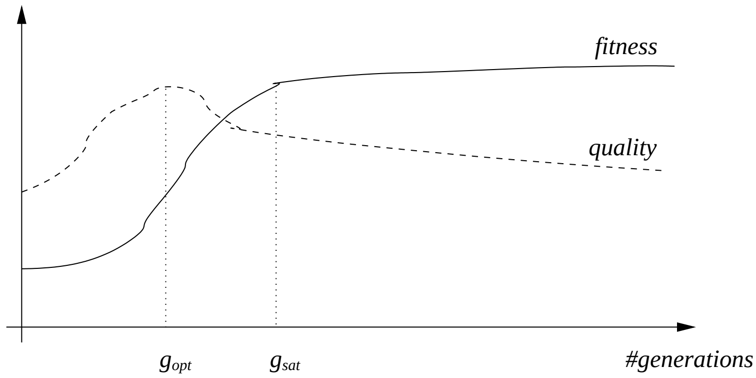

Fig. 4. Progression of quality and fitness during evolution

For BDD minimization by heuristic learning with GP, different absolute limits in numbers of generations have been used as stopping conditions for our method. By this, it is possible to investigate whether saturation and/or overfitting occur and if so, g_{sat} and g_{opt} can easily be determined as well.

4 Experimental Results

In this section, experimental results are presented. In the experiments that have been conducted, the population size has been set to 20 in each case and *tournament selection* has been used to select individuals for recombination. We have used reproduction rates of 50% (10 offsprings per generation) and 80% (16 offsprings per generation), respectively. The training sets T_1–T_3 have been utilized and for calculation of the fitness, we have applied the two different multi-objective approaches PREFERRED and PRIORITY. One of these techniques assumes that the importance of the benchmarks in the training set is unique, the other one deals with priorities (for details see [14]). Since the duration of computation for one generation is roughly constant during evolution, runtimes have been measured in number of generations.

The quality of heuristics learned by the GP is measured according to the results on a set C of the circuits *apex2, apex7, bcd, chkn, cps, in7, pdc, s1494, t1* and *vg2*. These benchmarks have not been considered during the evolutionary

Table 3. BDD sizes for heuristics learned by GP

Experiments with Training Set T_1

set of bench.	rep. rate	MOO type	Sum of BDD sizes for #generations: 5	10	15	20	25	30	40	50	60
T_1	50%	PREF.	2307	**2093**	**2093**	**2093**	**2093**	**2093**	**2093**	**2093**	**2093**
C			**3923**	4145	4144	4144	4144	4144	4144	4144	4144
T_1	80%	PREF.	2097	2096	2260	2416	2084	2570	**2085**	2110	2377
C			3894	3849	3815	**3777**	3898	3881	3841	3938	3951
T_1	50%	PRIO.	2245	**2095**	**2095**	**2095**	**2095**	**2095**	**2095**	**2095**	**2095**
C			3911	**3892**	**3892**	**3892**	**3892**	**3892**	**3892**	**3892**	**3892**
T_1	80%	PRIO.	2095	2095	2095	**2093**	**2093**	**2093**	**2093**	**2093**	**2093**
C			3904	**3884**	**3884**	4146	4146	4146	4146	4146	4146

Experiments with Training Set T_2

set of bench.	rep. rate	MOO type	Sum of BDD sizes for #generations: 5	10	15	20	25	30	40	50	60
T_2	50%	PREF.	2560	2578	2646	2205	2571	2698	2563	**2118**	2679
C			3999	4005	4055	**3848**	3946	4032	4022	3925	4048
T_2	80%	PREF.	2240	2510	2501	2511	**2106**	2458	2601	2599	2311
C			3879	4021	4008	4009	3949	**3861**	3974	4168	3884
T_2	50%	PRIO.	2647	2419	**2331**	**2331**	**2331**	**2331**	**2331**	**2331**	**2331**
C			4122	3976	3975	3975	3975	3975	**3973**	**3973**	**3973**
T_2	80%	PRIO.	2579	2537	**2101**	2610	2608	2144	2652	2490	2658
C			4156	4075	3796	4097	4029	**3786**	4277	4003	4175

Experiments with Training Set T_3

set of bench.	rep. rate	MOO type	Sum of BDD sizes for #generations: 5	10	15	20	25	30	40	50	60
T_3	50%	PREF.	**2117**	2239	2239	2239	2239	2239	2239	2239	2239
C			3872	**3860**	**3860**	**3860**	**3860**	**3860**	**3860**	**3860**	**3860**
T_3	80%	PREF.	2243	**2123**	2124	2124	**2123**	**2123**	**2123**	**2123**	**2123**
C			3889	**3870**	3874	3874	**3870**	**3870**	**3870**	**3870**	**3870**
T_3	50%	PRIO.	2450	2123	2093	**2113**	**2113**	**2113**	**2113**	**2113**	**2113**
C			3999	3870	4139	**3869**	**3869**	**3869**	**3869**	**3869**	**3869**
T_3	80%	PRIO.	2410	**2243**	**2243**	**2243**	**2243**	**2243**	**2243**	**2243**	**2243**
C			3926	**3889**	**3889**	**3889**	**3889**	**3889**	**3889**	**3889**	3900

optimization process. The smaller the sum of BDD nodes of all elements in C after application of a heuristics, the higher is the quality of the heuristics.

In a first set of experiments, the initial population was generated by the VARIED approach and Table 3 illustrates the results for different numbers of generations computed during evolution. The table is divided into 3 parts whereas in the different parts, the use of training set T_1, T_2 and T_3, respectively, is considered. Each (double) row includes results on the training set and the set of unknown circuits C, respectively, for varying stopping conditions in numbers of generations but fixed reproduction rates and multi-objective optimization types. In the first column of each row, the appropriate set of benchmarks is given. The second and third column include the reproduction rate and the multi-objective optimization type, respectively. Sums of BDD sizes for all elements included in the benchmark set are given in columns 4–12 for a heuristics that has the best quality among the elements in the population after certain numbers of generations. The best quality and fitness values are illustrated in bold face.

It can be seen that the use of each of the training sets leads to the best quality result for at least one combination of reproduction rate and multi-objective optimization approach. Thus none of the training sets is superior to another one. The best quality values itself found during evolution only slightly differ if the set of training examples is exchanged. The largest deviation can be observed if the PRIORITY method and a reproduction rate of 50% are used: in that case, the best quality found by learning with T_3 is 2.6% better than if T_2 is used while optimization with training set T_1 returns a result in between. If PREFERRED is used instead of PRIORITY, the deviation even decreases to 1.9%. For the combination of the PRIORITY method and a reproduction rate of 50%, an early saturation can be observed regardless of the training set used. Nevertheless, in general the behaviour during evolutionary optimization is not unique for different training sets. This can be observed for the combination reproduction rate 50%/PREFERRED: if T_1 is used, then there is an early saturation while for T_2, no saturation occurs up to generation 60.

After a certain number of generations, slight improvements on the training set T often can only be achieved at the cost of a considerably decreased quality. E.g. this can be seen in the last rows of part 1 (training set T_1). However, the most important observation that can be made is that the heuristics with the best quality has been found within the first 30 generations in each of the 12 experiments unless one. Even by the exceptional setup (training set T_2/reproduction rate 50%/PRIORITY), the improvement of the quality after generation 30 is very small, i.e. $< 1\%$. As a consequence, by stopping the evolution after 30 generations, approximately 50% of the runtime can be saved while the quality of the resulting heuristics even increases on average.

The different methods for the construction of the initial population have been compared in further experiments. T_1 has been selected as training set and the reproduction rate was 50%. The fitness values have been computed by the PREFERRED approach for multi-objective optimization. Results are illustrated in Table 4, where analogously to Table 3, sums of BDD sizes for all benchmark examples are given.

Table 4. Comparison of different methods for generation of the initial population

generation of initial population	set of bench marks	Sum of BDD sizes for #generations: 5	10	15	20	25	30	40	50	60
VARIED	T_1	2307	**2093**	**2093**	**2093**	**2093**	**2093**	**2093**	**2093**	**2093**
	C	**3923**	4145	4144	4144	4144	4144	4144	4144	4144
FIXED1	T_1	2097	2096	2096	2096	2095	2093	**2091**	**2091**	**2091**
	C	4147	3892	**3891**	**3891**	4156	4144	4144	4144	4144
FIXED2	T_1	2096	2096	2094	2094	**2093**	**2093**	**2093**	**2093**	**2093**
	C	**3891**	**3891**	4144	4144	4144	4146	4146	4146	4146

By using FIXED1 and FIXED2, respectively, slight improvements in quality can be achieved. The saturation point g_{sat} can be “delayed”, i.e. for FIXED1, and FIXED2, the best fitness is observed approximately 30 and 15 generations later than in the case of using VARIED. In each case, there is an overfitting behaviour during the learning process, but the point of time when the heuristics with the best quality occur is different. For the given setup, FIXED2 is the best method to generate the initial population since best qualities for heuristics that occur in any of the three experiments are obtained very early.

5 Conclusions

In this paper, GP-based heuristic learning for BDD minimization has been considered. Several parameters of the GP have been analyzed, i.e. generation of the initial population, selection of the training set examples and stopping conditions. Particularly, the generalization behaviour of the heuristics that are learned during evolution has been investigated. It can be seen that the quality of a heuristics does not always correspond directly to its fitness during the GP. Experimental results have been given that prove that by determination of suitable stopping conditions, approximately 50% of the runtime can be saved while the quality of heuristics learned is even improved.

References

1. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Comp.*, 45(9):993–1002, 1996.
2. R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

3. L. Davis. *Handbook of Genetic Algorithms.* van Nostrand Reinhold, New York, 1991.
4. N. Drechsler, F. Schmiedle, D. Große, and R. Drechsler. Heuristic learning based on genetic programming. In *EuroGP*, volume 2038 of *LNCS*, pages 1–10. Springer Verlag, 2001.
5. R. Drechsler. *Evolutionary Algorithms for VLSI CAD.* Kluwer Academic Publisher, 1998.
6. R. Drechsler and B. Becker. Learning heuristics by genetic algorithms. In *ASP Design Automation Conf.*, pages 349–352, 1995.
7. R. Drechsler, B. Becker, and N. Göckel. Minimization of OKFDDs by genetic algorithms. In *Int'l Symposium on Soft Computing*, pages B:271–B:277, 1996.
8. M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.
9. D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addision-Wesley Publisher Company, Inc., 1989.
10. J. Koza. *Genetic Programming - On the Programming of Computers by means of Natural Selection.* MIT Press, 1992.
11. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1994.
12. S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Int'l Conf. on CAD*, pages 74–77, 1995.
13. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.
14. F. Schmiedle, N. Drechsler, D. Große, and R. Drechsler. Priorities in multi-objective optimization for genetic programming. In *Genetic and Evolutionary Computation Conference*, 2001.
15. S. Yang. Logic synthesis and optimization benchmarks user guide. Technical Report 1/95, Microelectronic Center of North Carolina, 1991.