# Heuristic Learning based on Genetic Programming

Nicole Drechsler, Frank Schmiedle, Daniel Große, Rolf Drechsler

Institute of Computer Science
Chair of Computer Architecture (Prof. Bernd Becker)
Albert-Ludwigs-University
79110 Freiburg im Breisgau
ndrechsl@informatik.uni-freiburg.de

**Abstract.** *In this paper we present an approach to learning heuristics based on Genetic Programming (GP). Instead of directly solving the problem by application of GP, GP is used to develop a heuristic that is applied to the problem instance. By this, the typical large runtimes of evolutionary methods have to be invested only once in the learning phase. The resulting heuristic is very fast. The technique is applied to a field from the area of VLSI CAD, i.e. minimization of Binary Decision Diagrams (BDDs). We chose this topic due to its high practical relevance and since it matches the criteria where our algorithm works best, i.e. large problem instances where standard evolutionary techniques cannot be applied due to their large runtimes. Our experiments show that we obtain high quality results that outperform previous methods, while keeping the advantage of low runtimes.*

## 1 Introduction

*Decision Diagrams* (DDs) are often used in CAD systems for efficient representation and manipulation of Boolean functions. The most popular data structure is the *Binary Decision Diagram* (BDD) [Bry86]. Recently, several approaches in logic synthesis have been presented that make use of BDDs. (For an overview see [DB98].) One drawback of this data structure is that it is very sensitive to the variable ordering, i.e. the size may vary from linear to exponential. Finding the optimal variable ordering is a difficult problem [BW96] and the best known algorithm has exponential runtime.

This is the reason why in the last few years many authors presented heuristics for finding good variable orderings. The most promising methods are based on dynamic variable ordering [FMK91],[Rud93],[PS95]: BDDs for some Boolean functions have been constructed for which all other topology oriented methods failed. New methods based on non-deterministic algorithms have been proposed for BDD minimization, e.g. genetic algorithms [DBG95] and simulated annealing [RBKM92],[BLW95]. The major drawback of these approaches is that in general they obtain good results with respect to quality of the solution, but the running times are often much larger than those of classical heuristics. Due to the high complexity of the design process in VLSI CAD often fast heuristics are used. These heuristics are developed by the designer himself. But they also often fail for specific classes of circuits. Thus it would help a lot, if the heuristics could learn from previous examples, e.g. from benchmark examples.

A theoretical model for learning heuristics by *Genetic Algorithms* (GAs) has been presented in [DB95]. The new aspect of this model is that the GA is not directly applied to the problem. Instead the GA develops a good heuristic for the problem to be solved. First applications to multi-level synthesis and to 2-level AND/EXOR minimization have been presented. There the model has not been fully used, i.e. only a part of the features has been implemented. Extensions have been proposed in [DGB96] and [DDB99], where learning of BDD heuristics has been studied based on GAs. But due to the fixed length encoding these approaches also have some disadvantages:

- The length of the heuristic is limited resulting in limitations with respect to quality.
- Decision procedures, like if-then-else, could not be integrated in the heuristics.

In this paper we present an approach to heuristic learning based on *Genetic Programming* (GP) [Koz92],[Koz94]. Due to the more flexible encoding based on tree structures, the disadvantages of the GA approaches described above can be avoided, while keeping the advantages. To keep the heuristic under development as compact as possible, reduction operators are introduced. Compared to GAs more flexible operators are defined and integrated in the GP run. Experimental results demonstrate the efficiency of the approach.

The paper is structured as follows: In the next section the problem definition is given and the model of heuristic learning is described. Then the solution based on GP is outlined and the (genetic) operators are introduced. Experimental results are reported and finally the paper is summarized.

## 2 Problem Description

In [DB95] a learning model has formally been introduced for GAs. In this section we briefly review the main notation and definitions to make the paper self-contained. Since the definition of the model is based on the evaluation of the fitness function only, it becomes obvious that even though it has been developed for GAs, it can be transferred to GPs directly.

It is assumed that the problem to be solved has the following property: There is defined a non empty set of optimization procedures that can be applied to a given (non-optimal) solution in order to further improve its quality. These procedures are called *Basic Optimization Modules* (BOMs) in the following. The heuristics are sequences of BOMs. The goal of the approach is to determine a good (or even optimal) sequence of BOMs such that the overall results obtained by the heuristics are improved. From the flexibility of the tree-like data structure in GPs we are later even able to define more powerful operators.

In the following we assume that the reader is familiar with the basic concepts and notation of evolutionary approaches. In [DB95] and [DGB96], a multi-valued string encoding of fixed length has been used, but this already by definition limits the quality of the result. Due to the GP concept, there is in principle no limit on the size of the resulting heuristics.

The set of BOMs defines the set H of all possible heuristics that are applicable to the problem to be solved in the given environment. H may include problem specific heuristics but can also include randomized techniques.

To each BOM h we associate a cost function, i.e. a value that determines how expensive it is to call this module. This value can be chosen dependent on different criteria, like memory consumption or run time. Furthermore, the quality function determines the quality resulting from the application of this module. These two values determine the fitness of the BOM. Summation over all BOMs in a heuristic determines the overall fitness. The elements are evaluated on a set of benchmark examples, the so-called training set. The multiple objectives are optimized in parallel by the method described in [DDB99].

### 2.1 BOMs

Most of the algorithms that are used here as BOMs for heuristic learning are well-known BDD minimization techniques They are based on dynamic variable ordering. *Sifting* (SIFT) is a local search operation for variable ordering of BDDs which allows hill climbing. *Group sifting* (GROUP) and *symmetric sifting* (SYMM) additionally make use of symmetry aspects of the variables of the considered Boolean functions. The BOM *window permutation* of size 3 (4), denoted as WIN3 (WIN4), tests all permutations of 3 (4) adjacent variables, where the window of size 3 (4) "goes" through the whole BDD from the top to the bottom variables. 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 next BOM is called *inversion* (I) and inverts the variable ordering of a BDD between two randomly chosen variables. Finally, the set of BOMs contains an "empty" element, denoted as NOOP. In the GA approach this operator was used to model strings of various sizes, while the underlying data structure was a fixed-length string. Using GPs this problem by definition does not occur. Nevertheless, NOOP is important to describe a condition, e.g. it is possible to describe an IF-condition without a statement in the ELSE-branch. These types of operators could not be introduced using Gas, underlining the flexibility of the approach introduced below.

For more details about the learning model, the definition of BDD, and BOMs see [DB95], [DGB96], and [DDB99].

## 3 GP Solution

For basic definitions of GP we refer to [Koz92],[Koz94]. In the following only the problem specific description is given.

### 3.1 Representation

For GP-based heuristic learning, heuristics are represented by trees with outdegree  2, i.e. each node has two outgoing edges. The heuristics are the individuals generated during the evolutionary process. The trees include BOMs that are located at the leaf nodes. There are three additional node types that cover all the inner nodes:

1. CONCAT: The left son is evaluated and afterwards the result is used to evaluate the right son.
2. IF: An IF node always comes with a value x that represents a condition. The left son of this node is evaluated if the ratio (# symmetric variables / # variables in total) is smaller than x. Otherwise, the right son is evaluated instead.
3. WHILE: The procedure of subsequent evaluation of the left and the right son is iterated as long as it reduces the size of the BDD.

There may be CONCAT nodes at the leaves as well, while the types IF and WHILE can only occur at inner nodes. To evaluate a heuristic, the root of the corresponding tree has to be recursively evaluated by a *depth first search*-based technique.
If the root node is a leaf, then the BOM covering this leaf has to be applied to the BDD. If the leaf is a CONCAT node, nothing has to be done. Notice, that algorithmic descriptions of a heuristic using IF- and WHILE-conditions are not possible with this learning model if a string representation in a GA is used. On the other hand, each string generated by a GA can also be constructed using the GP approach.

### 3.2 Operators

For recombination of individuals during evolution, two crossover operators are provided:

1. CAT: The root nodes of the parents are linked by a new CONCAT node c. Parent 1 becomes the left son and parent 2 becomes the right son of c. The node c itself is the root of the offspring.

2. MERGE: The operator extracts a sub-tree of parent 1 and one of parent 2 and combines these sub-trees by an additional CONCAT node in the same way.

By definition it is obvious that bloating can be observed if CAT is used, while MERGE generates offspring with sizes that are similar to the parents' size.

Additionally, four mutation operators are used:

1. B-MUT: A BOM leaf is selected at random and the BOM at this leaf is replaced by a different one.
2. CI-MUT: A randomly selected inner CONCAT node is replaced by an IF node. The value x is also determined at random.
3. CW-MUT: This mutation is similar to CI, but the CONCAT node is replaced by a WHILE node rather than an IF node.
4. IF-MUT: Value x is either incremented or decremented by 0.1.

### 3.3 Algorithm and Parameter Setting

In this section we describe the parameter settings used in our GP. For the overall flow standard techniques as described in [Koz92],[Koz94] are used.

First, 20 individuals are generated for the initial population. To keep the run time moderate, it turned out by experiments that it is clever to limit the number of BOMs in a heuristic. In contrast to the GA approach, where fixed-length strings are used, in the GP approach this is not a strict limitation, since more powerful operators are available that allow to describe complex constructs, like loops based on WHILE. The initial elements are generated as follows: A tree with unlabelled 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, the sub-trees of the node are eliminated and when the BOM limit is reached, all following nodes are labeled with CONCAT and their sub-trees are removed. The individual is complete when there are no unlabelled nodes any more.

In each generation, 10 offspring are generated. The probabilities for the operators are given in the following table:

| operator | CAT | MERGE | B | CI | CW | IF |
|---|---|---|---|---|---|---|
| probability | 5 | 45 | 20 | 10 | 10 | 10 |

Two parents are selected in the case of crossover and one in the case of mutation by roulette-wheel-selection. The operator is applied and afterwards, the fitness of the offspring is calculated. The concept of multi-objective minimization [DDB99] is used for that and the optimization criteria are the resulting BDD sizes on a training set of BDD benchmarks. Evolution is aborted if either there has been no improvement for 100 generations or a limit of 1000 generations has been reached.

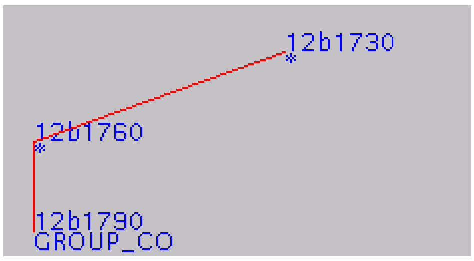
### 3.4 Size Reduction

To prevent bloating, a limit on the number of BOMs in an individual is set. If this limit is exceeded by crossover, the offspring is removed immediately and selection and recombination are repeated. Furthermore, a reduction mechanism has been included. If the sub-tree of a node did not cause changes in BDD size for any of the benchmarks in the training set, the node and its sub-trees are removed from the tree and replaced by a single CONCAT node since obviously this sub-tree does not contribute anything to the minimization of one of the benchmarks. It can be determined efficiently during evaluation if there has been an influence of a sub-tree to minimization, and by this reduction mechanism, sizes of the individuals are reduced considerably on average. Furthermore, this has a positive effect on the quality of the generated heuristics, since only parts are kept that have an effect on the quality of the result. This guarantees that no run time is wasted by application of BOMs that might be expensive, but do not produce any (positive) change. In application in VLSI CAD this is an important factor, since most of the tools are interactive.

## 4 Experimental Results

In this section we describe our experimental results obtained by the methods above. For all experiments we used a SUN Sparc 20 and all run times are given in CPU seconds. The results are compared with the GA approach from [DDB99] and the experiments are carried out in the same software environment using multi-objective optimization. For this, after each run a *set* of optimal (non comparable) solutions - instead of a single element - is returned. The same training set as in [DDB99] is chosen during the learning phase of the GP.

As a first result it can be observed that already very simple solutions give very high quality results, e.g. the following tree was contained in the final solution after the learning phase, that took about 25 CPU minutes:




The * denotes a node of type CONCAT. Furthermore, at each node some program specific information about the pointer structure is provided. This simple heuristic obtains the following results:

| | | | Genetic Programming | | [DDB99] | | |
|---|---|---|---|---|---|---|---|
| **name** | in | out | size | time | size | time | optimal |
| **add6** | 12 | 7 | 28 | 0.25 | 28 | 0.10 | 28 |
| **addm4** | 9 | 8 | 163 | 0.18 | 163 | 0.10 | 163 |
| **cm85a** | 11 | 3 | 27 | 0.18 | 27 | 0.10 | 27 |
| **m181** | 15 | 9 | 54 | 0.27 | 54 | 0.20 | 54 |
| **risc** | 8 | 31 | 65 | 0.13 | 65 | 0.10 | 65 |
| **sum** | | | 337 | 1.01 | 337 | 0.60 | |
| **average** | | | 67.40 | 0.20 | 67.40 | 0.12 | |

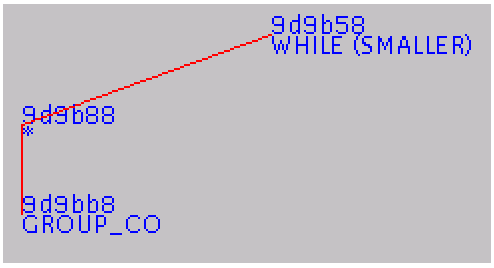
The name of the benchmark is given in the first column followed by the number of inputs and outputs. Then the results for quality measured in the size of the BDD and the run time are given first for the GP approach and then for the approach of [DDB99]. As can be seen, on the training set both approaches obtain the same results with respect to quality, i.e. both determine the optimal values. The GP approach is slightly slower, but the situation changes, if the generated heuristic is applied to benchmarks that were unknown during the learning phase:

| | | | Genetic Programming | | [DDB99] | | |
|---|---|---|---|---|---|---|---|
| **Name** | in | out | size | time | size | time | optimal |
| **5xp1** | 7 | 10 | 41 | 0.11 | 50 | 0.10 | 41 |
| **alu4** | 14 | 8 | 353 | 0.49 | 354 | 1.40 | 349 |
| **cm151.a** | 12 | 2 | 16 | 0.16 | 16 | 0.10 | 16 |
| **cm162a** | 14 | 5 | 30 | 0.17 | 37 | 0.10 | 29 |
| **cm163a** | 16 | 5 | 25 | 0.20 | 26 | 0.10 | 25 |
| **Cmb** | 16 | 4 | 27 | 0.16 | 27 | 0.10 | 27 |
| **Cu** | 14 | 11 | 31 | 0.18 | 31 | 0.10 | 31 |
| **Sqn** | 7 | 3 | 48 | 0.12 | 56 | 0.10 | 48 |
| **Gary** | 15 | 11 | 289 | 0.38 | 306 | 0.20 | 289 |
| **s1494** | 14 | 25 | 386 | 0.30 | 383 | 0.40 | 368 |
| **Tial** | 14 | 8 | 579 | 0.50 | 611 | 1.00 | 563 |
| **Sum** | | | 1825 | 2.77 | 1897 | 3.70 | |
| **Average** | | | 165.91 | 0.25 | 172.45 | 0.34 | |

In all but one case, the GP approach is at least as good as the GA. On average the quality and run time are improved. This could already be observed for the simplest heuristic computed that even did not make use of the powerful operators, like WHILE and IF. Especially in the application of heuristic learning it is important to develop heuristics that are not only good for the training set, but to construct sequences of BOMs that generalize very well. Since in later application of the heuristic in general the benchmarks are not known in advance, the GP result is of very high quality: The BDD sizes measured in number of nodes **and** the run times are improved in parallel.

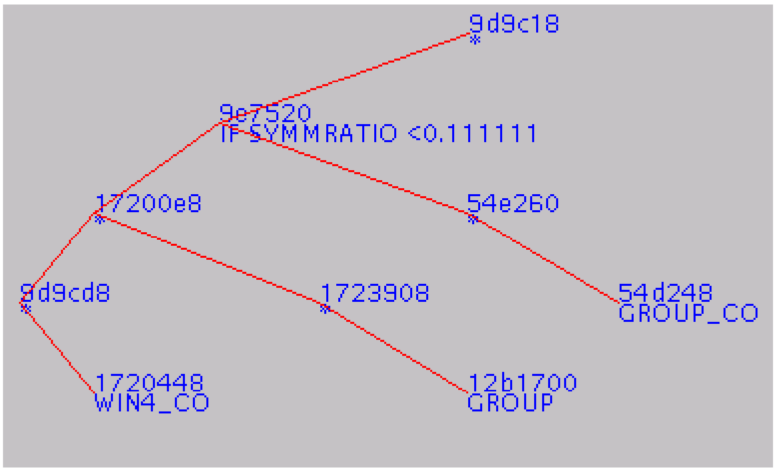
Due to space limitation, we cannot discuss all elements of the final solution in detail, but we will focus on some high-lights in the following.

An element very similar to the first one in the final set is:



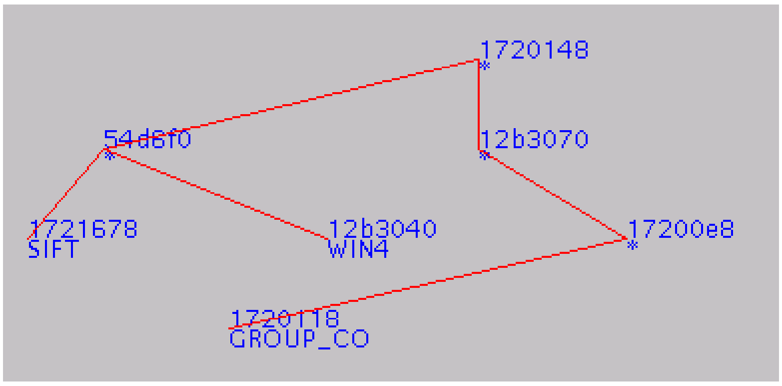

This element is nearly the same as the first one, but has the WHILE operator as top node. Thus, the sub-tree is called repeatedly and by this the quality can be further improved. Here it should be mentioned that the use of WHILE in the root node might be "dangerous", since on bigger benchmarks the run times can become very large. The WHILE implies that the complete sub-tree is repeatedly called as long as an improvement can be obtained. If the scenario is considered that the heuristics in the sub-trees always improve on the result of the previous iteration, but only very slightly, run times can explode. (In practice this has never been observed in our experiments, but this aspect should be considered when designing heuristics for application in VLSI CAD.)

The solutions can also become more complex:




Then it is up to the user to decide for the specific application whether he is willing to invest more run time to get higher quality results. But as can be seen, all operators that are "GP specific", i.e. the powerful operators that cannot be described based on GAs really occur in the final solution (see examples above). The heuristics become very compact and by this also easy to analyze and easy to understand for the user.

The best solution with respect to quality measured in the number of BDD nodes found in this test run was:




The sum over all nodes of the benchmarks that were unknown during the learning phase was 1805.

In summary, the GP approach clearly outperforms the (highly optimized) GA that has been developed over several years. The underlying data structure gives more flexibility and allows to define more powerful operators resulting in compact heuristics that obtain high quality results.

## 5 Conclusions

In this paper we presented an approach to learning heuristics based on GPs. It became possible to generate flexible high-quality heuristics that have very high performance in comparison to direct application of evolutionary techniques. Experimental results showed details about the learning phase and demonstrated the influence of the integration of decision procedures. This became possible due to the underlying encoding as trees typically used in GPs, while this turned out to be a limitation of the corresponding GA approach. It is focus of current work to study the multi-objective aspect of this application in more detail and to also apply the technique to further applications in the field of VLSI CAD.

## References

[Bry86] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Comp., 8:677-691, 1986.

[BW96] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. IEEE Trans. on Comp., vol. 45, num. 9, pages 993-1002, 1996.

[BLW95] B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. In Int'l Workshop on Logic Synth., pages 5b:5.1-5.10, 1995.

[DB95] R. Drechsler and B. Becker. Learning heuristics by genetic algorithms. In ASP Design Automation Conf., pages 349-352, 1995.

[DB98] R. Drechsler and B. Becker. Binary Decision Diagrams -- Theory and Implementation, Kluwer Academic Publishers, 1998

[DBG95] R. Drechsler, B. Becker, and N. Göckel. A genetic algorithm for variable ordering of OBDDs. In Int'l Workshop on Logic Synth., pages P5c:5.55-5.64, 1995.

[DDB99] N. Drechsler, R. Drechsler, and B. Becker. A new model for multi-objective optimization in evolutionary algorithms. In Fuzzy'99, LNCS 1625, pages 108-117, 1999.

[DGB96] R. Drechsler, N. Göckel, and B. Becker. Learning heuristics for OBDD minimization by evolutionary algorithms. In PPSN'96, LNCS 1141, pages 730-739, 1996.

[FMK91] M. Fujita, Y. Matsunga, 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.

[Koz92] J. Koza. Genetic Programming - On the Programming of Computers by means of Natural Selection. MIT Press, 1992

[Koz94] J. Koza. Genetic Programming II – Automatic Discovery of Reusable Programs. MIT Press, 1994

[PS95] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In Int'l Conf. on CAD, pages 74-77, 1995.

[RBKM92] D.E. Ross, K.M. Butler, R. Kapur, and M.R. Mercer. Functional approaches to generating orderings for efficient symbolic representations. In Design Automation Conf., pages 624-627, 1992.

[Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In Int'l Conf. on CAD, pages 42-47, 1993.