Minimizing the Number of One-Paths in BDDs by an Evolutionary Algorithm

Mario Hilgemeier Nicole Drechsler Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{mh,nd,rd}@informatik.uni-bremen.de

Abstract- Ordered binary decision diagrams (BDDs) are used in VLSI CAD, especially for the canonical representation of Boolean functions. In the last decade, the method of choice for optimizing this data structure was minimizing the number of nodes in the associated graph. However, recent works have shown that the number of paths is also important.

In this work, minimizing the number of one-paths of BDDs is accomplished by an evolutionary algorithm (EA) acting on the permutation of variables. The optimal operator weights for the EA were determined by a parameter study. Experimental results demonstrate the efficiency of our approach.

1 Introduction

Ordered binary decision diagrams (BDDs) are known to be canonical representations of Boolean functions by a directed graph [3, 4]. One-paths are those routes through the BDD that start at a root and terminate in a logical 1.

In most cases, the minimal number of nodes is the optimization objective. However, one-paths are important for certain applications in the field of VLSI CAD. These applications include:

- Formal verification using SAT-solving: The number of steps necessary to solve a SAT problem is directly related to the number of paths in the corresponding BDD [14].
- Logic synthesis: The number of paths influences the minimization process [12, 17]. A minimized disjoint-sum-of-product representation can directly be extracted from the BDD [9] and leads to smaller circuits.

Currently applied typical heuristic methods for minimizing the number of one-paths like modified sifting (MS) [8] run very fast on certain Boolean functions. But MS can be surpassed by evolutionary algorithms if one is willing to invest more computing time.

Evolutionary algorithms (EAs) excel in problem areas where no straightforward solution method with low computational complexity is known. EAs use a population of solutions that can evolve by producing new offspring using mutation and genetic recombination. The best individuals of that population survive and can have new offspring. Thus a convergence towards good solutions is attained. For example, EAs were successful in VLSICAD [5] and embedded system design [6].

The one-path minimization problem in a BDD is a computationally hard problem that can be attacked by an EA. That EA searches for an optimal permutation of the variable order in the BDD. This work shows that an EA can substantially reduce the number of one-paths in BDDs.

The paper is structured as follows: Section 2 explains BDDs and the MS algorithm. The EA is described in detail in Section 3. In Section 4, three promising operator combinations of a standard EA are selected. The most successful of these three is then compared to the results of MS. Section 5 concludes the paper and summarizes the results.

2 Preliminaries

This section introduces some basics concerning BDDs, one-paths, and reviews a state-of-the-art heuristic to minimize the number of one-paths that will be used later for comparison with the EA.

2.1 Binary Decision Diagrams

BDDs [3] are directed acyclic graphs where a Shannon decomposition is carried out in each node. Each node has a Boolean variable attached to it from the corresponding logical expression that describes the Boolean function $f : B^n \rightarrow B^m$. Depending on the value of the variable, the path is chosen; in Figures 1 and 2, a dashed line represents a logical 0 value and a logical 1 is shown as a straight line. In these examples, $n = 5$ and $m = 1$. Note that there are m root nodes. In the case of $m > 1$, one speaks of a *shared* BDD because $f = (f_1, ..., f_m)$ is composed of m functions $f_i : B^n \rightarrow B, \; 1 < i \leq m.$

If a BDD contains no isomorphic sub-graphs and no vertices with both outgoing edges pointing to the same node, that BDD is called *reduced*. Following an *ordered* BDD down from its root node, each Boolean variable value is met at most once and in the same order along each path.

Typically, a variable is ascribed to more than one node. In this paper, we restrict "BDD" to mean reduced ordered binary decision diagrams. It follows that the nodes belonging to the same variable can be arranged in horizontal levels.

The BDDs terminate either in logical 0 or 1. One-paths are those routes through the BDD that start at the root and terminate in the node representing logical 1.

The number of nodes necessary to represent the BDD for the Boolean function is dependent on the variable (i.e. level) order. The following example will show the effects of variable ordering on the number of nodes of a BDD and especially the number of one-paths. The number of one-paths is denoted by P_1, the number of zero-paths by P_0.

Example 1. *Consider the BDDs in Figures 1 and 2 [7]. Both BDDs represent the same Boolean function*

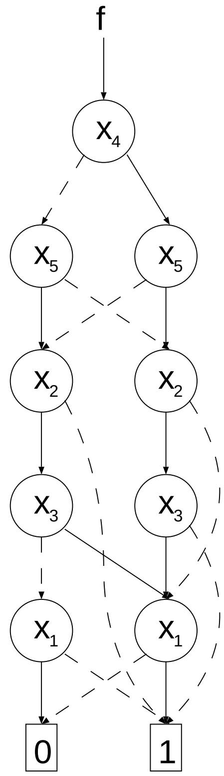

Figure 1: BDD of f with minimized nodes

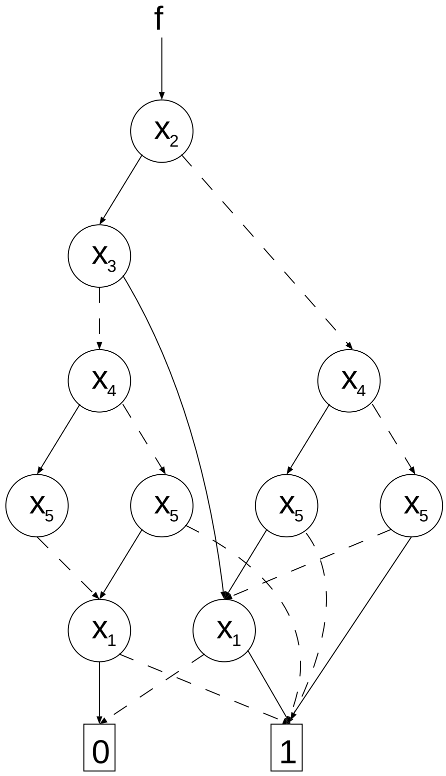

Figure 2: BDD of f with minimized paths

$f(x_1, x_2, x_3, x_4, x_5)$ which is defined by
$f = x_1\overline{(x_4 \oplus x_5)} + \overline{x_2}(x_4 \oplus x_5) + x_1x_3 + \overline{x_1}x_2\overline{x_3}$.

Figure 1 shows 11 nodes. This BDD corresponds to the variable order $(x_4, x_5, x_2, x_3, x_1)$; it has $P_0 = 8$ and $P_1 = 12$. The logically equivalent BDD in Figure 2 has the variable order $(x_2, x_3, x_4, x_5, x_1)$. It has more nodes (12), but only 9 one-paths and 5 zero-paths.

In the above example it has been demonstrated that the number of one-paths does not have to be minimal when the number of nodes is minimized and vice versa.

2.2 Modified Sifting

Sifting [15] is a well-known algorithm for minimizing the number of nodes in a BDD. It works the following way: while the other variables remain in their positions, a single variable is shifted to all possible positions. Then the best of these positions - namely the one with the minimal number of nodes - is accepted for that variable. This is done once for all variables.

Modified sifting (MS) [8] works like the sifting algorithm for minimizing the number of nodes, but in this case the acceptance criterion is that P_1 has to be minimal.

3 Evolutionary Algorithm

In this section, the employed standard EA with sexual reproduction, recombination and mutation and its associated operators is introduced. This standard EA is only modified with respect to the probabilities of these operators.

The search for the minimal number of one-paths for the BDD representations of a given Boolean function is analogous to the search for a permutation of the node indices.

We will first give details of coding, fitness evaluation, and operators. Then the evolutionary algorithm is sketched.

3.1 Coding and Fitness Evaluation

An individual of the population is described by a permutation of the variable indices of the BDD. For each individual, this permutation is stored in an index list. This index list constitutes the chromosome of the individual.

During the minimum search for a single Boolean function, P_1 is the fitness function for each individual.

3.2 Operators

We now briefly describe the operators that will be used by the EA.

The reproduction includes crossover operators that are specially designed for recombination of permutations. One-point crossover does not work for permutations because this leads to duplicates and omissions (except in the case where both permutations contain the same set of variable indices on corresponding sides of the cut). Therefore crossover operators employ “repair mechanisms” that make sure that the permutation remains valid or use algorithms that always produce valid permutations. Typically, these operators produce two children from two parents.

After reproduction, a single mutation operator might be applied according to the given probabilities. A mutation operator acts on a single child. That is, the two children from a reproduction may be mutated by different operators.

In other words, a new child might have been subject to one crossover, to one mutation, or both.

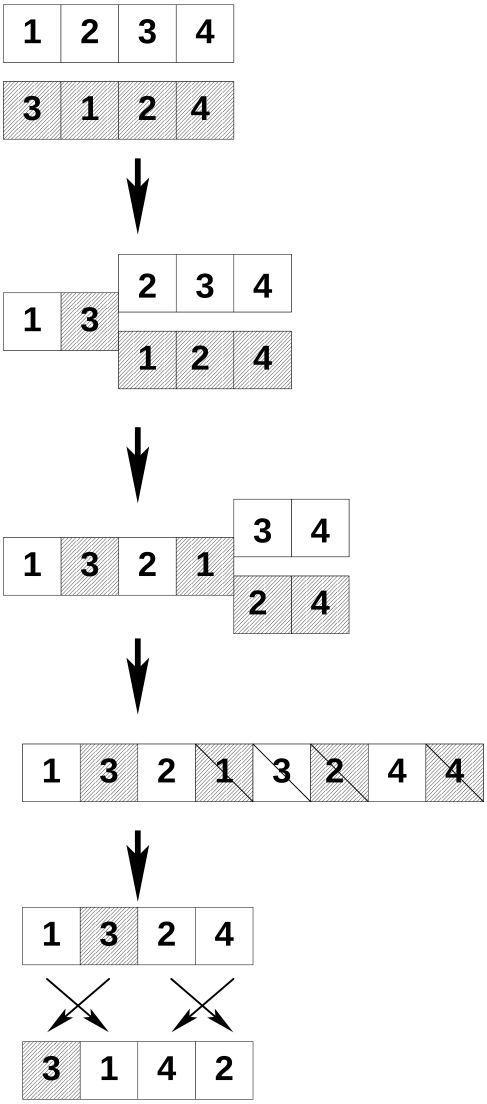

Figure 3: Example for the MERGE operator

Reproduction and mutation are invoked according to the given probabilities. Reproduction operators used are PMX, CX, MERGE, INV and REP. Mutation operators include MUTP1, MUTP2, and MUTPD. These operators will be described now.

- PMX (partially matched crossover) [13] respects the absolute position of each variable index. PMX sets two "crossover points" in the permutation between which the index order remains the same as in the parent. For the other positions in the permutation, a substitution mechanism guarantees that each variable index appears exactly once in each permutation.

- CX (cycle crossover) [11] makes use of cycles in the permutation to avoid double or missing variable indices after the crossover.

- MERGE produces the first child in the following way. Alternating between the parents, MERGE takes one variable index from each parent (in the order they appear in the parents) until the permutation length is reached. After doing this, MERGE checks from left to right if an index has been used already. If this is the case, that index number is removed.

  The second child is produced by exchanging the indices of even and odd positions in the child permutation (see Example 2 below).

- The inversion operator INV inverts the order of the index list for a randomly chosen part of the chromosome. It produces one child from each parent.

- The reproduction operator REP reads the index list forward. This means that no genetic change is introduced by this operator. It only copies individuals. It produces two children that are identical to their parents.

- The mutation operator MUTP1 selects two random positions in the list, and exchanges their contents.

- MUTP2 is MUTP1 applied twice to the same individual.

- MUTPD exchanges randomly chosen adjacent positions in the permutation.

All operators produce only valid offspring. As an example for the MERGE operator, consider Figure 3.

Example 2. *Let parent 1 be the permutation (**1, 2, 3, 4**) and parent 2 be the permutation (3, 1, 2, 4) (see Figure 3). Now imagine these as the two parts of an open zipper. When the zipper is closed, the two halves merge, yielding (**1**, 3, **2**, 1, **3**, 2, **4**, 4).*

*Working from left to right, the redundant variable indices are cancelled (boxes diagonally striked out in Figure 3). So the first child becomes (**1**, 3, **2**, 4).*

Now the second child is generated by exchanging even and odd positions (crossed arrows in Figure 3). Numbers 1 and 3 as well as 2 and 4 are exchanged, yielding (3, 1, 4, 2).

3.3 Algorithm

The flow of the EA is sketched in Figure 4. At the beginning, individuals are initialized with random permutations of the variable indices. The population size is three times the number of inputs of the Boolean function being minimized, but not greater than 120. Each individual is assigned

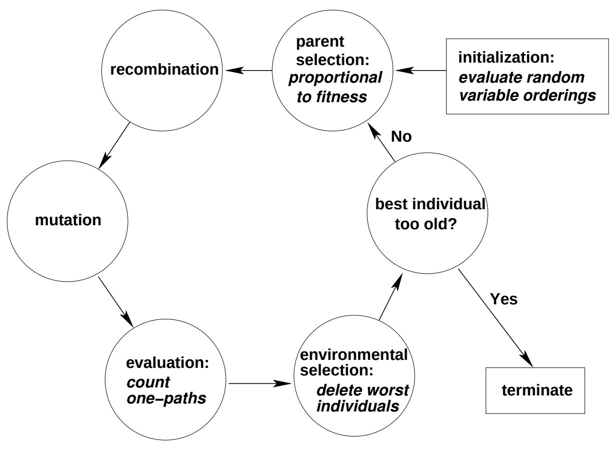

Figure 4: Sketch of the EA

Table 1: Operator weightings with their EA results

operator	weighting # 1	2	3	4	5	6	7	8
PMX	0.9510	0.9216	0.9510	0.9216	0.98	1.00	0.98	0.98
CX	0.0000	0.0196	0.0000	0.0196	0.00	0.00	0.00	0.00
MERGE	0.0000	0.0196	0.0000	0.0196	0.00	0.00	0.00	0.00
INV	0.0245	0.0196	0.0245	0.0196	0.01	0.00	0.01	0.01
REP	0.0245	0.0196	0.0245	0.0196	0.01	0.00	0.01	0.01
MUTP1	0.05	0.33	0.33	0.05	0.05	0.05	0.05	0.02
MUTP2	0.05	0.33	0.33	0.05	0.05	0.05	0.05	0.02
MUTPD	0.00	0.33	0.33	0.00	0.00	0.00	0.05	0.00
result								
Σ nodes	11968	12481	13861	14344	13284	12663	11985	15283
Σ P_1	**29235**	31139	29406	29258	**29218**	29492	29340	29785
Σ sec	1757.95	1781.84	1748.23	1856.52	1830.61	1720.19	1720.70	1756.22
Σ gener.	9494	9810	9389	10313	10156	9266	9294	9712

operator	weighting # 9	10	11	12	13	14	15	16
PMX	0.98	0.98	0.98	0.98	0.00	0.00	0.36	0.00
CX	0.00	0.00	0.00	0.00	0.98	0.00	0.32	0.48
MERGE	0.00	0.00	0.00	0.00	0.00	0.98	0.32	0.48
INV	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
REP	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
MUTP1	0.07	0.10	0.06	0.08	0.07	0.07	0.07	0.07
MUTP2	0.07	0.10	0.06	0.08	0.07	0.07	0.07	0.07
MUTPD	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
result								
Σ nodes	12972	13299	12834	12542	16561	12863	11394	13240
Σ P_1	**29191**	29392	29269	31569	31964	31981	31392	29893
Σ sec	1706.35	1732.86	1669.44	1922.72	2041.20	1763.46	1686.11	1717.26
Σ gener.	9580	9556	9119	10105	10132	11186	9797	9550

a fitness according to the number of one-paths of the corresponding BDD.

After initialization the evolutionary loop is started. Fitness-proportional selection [11], i.e. roulette wheel selection, determines which parents will produce children. In each generation, the number of children is half the population size.

Recombination and mutation operators are chosen randomly according to the specified set of operator probabilities. The fitness evaluation consists of the counting of the one-paths of the children. To keep the population constant, the worst third of this bigger population (parents + children) is discarded during the environmental selection before the next generation.

The termination condition means that the best individual has survived for a long time and no improvements were found during its lifetime.

4 Experiments

To tune and test the EA, a number of experiments was conducted with various benchmarks functions. These benchmark functions were chosen as a representative set of easy and hard problems. Moreover, the benchmarks are mostly not too expensive, computationally, so that the set of benchmarks can be run in reasonable time.

4.1 Qualities

In this section we give some arguments why we deem the chosen set of benchmark functions representative.

The sizes of the 38 benchmark functions vary over a wide range. They have between 5 and 54 inputs and between 1 and 41 outputs (for a detailed list see Table 3). From the MS algorithm, the values for P_1 are known to vary over more than three orders of magnitude.

4.2 Computing Environment

The tests for the EA as well as the reference runs for the MS algorithm were done on a Linux machine equipped with an AMD Athlon XP2200+ processor (1.8 GHz, 256kB cache) and 512 MB RAM.

For the BDD representation, CUDD [16] was used. The EA is based on the C++ library for evolutionary algorithms GAME [10] (version 2.3). Test control programs were written in Python.

4.3 Parameter Selection

In order to assess the usefulness of the EA operators (see Section 3.2) for our set of benchmarks, a good, balanced set of operators is sought. Such a set of operator probabilities is called a weighting. To find a good operator set, sixteen different weightings were tested, e.g. with PMX, CX, and MERGE of nearly equal probability, etc.

A simple measure of goodness for an operator weighting is the sum over all P_1 values. In this way, benchmarks with larger absolute improvements of P_1 influence the sum stronger. We considered this a desirable effect because these more complex functions are the intended application area.

The final result of this parameter exploration is shown in Table 1. The three best results for the sum of P_1 are shown in boldface. The most promising operator weighting is #9.

This operator weighting of the EA was used for comparison with MS. Since running the EA with a specific parameter set over all 38 benchmarks needed about half an hour with the given hardware configuration, each weighting was tested only once during the selection process.

4.4 EA Performance

The performance of the selected EA was tested by running it 25 times, each time over all the benchmarks.

The different value spreads for nodes and P_1 are shown graphically in Figure 5. Each benchmark is represented by a dot. For the 25 runs of each benchmark with the finally selected operator weightings, median (dot) and value spread (bars) are shown. The horizontal bars show the range of the resulting node numbers of the BDD of each benchmark. The vertical bars correspond to the range of results in P_1. Note that in this plot of one-paths versus nodes both axes are logarithmically scaled.

Figure 5 shows the wide range of problem sizes and the rough correlation between the length of one-paths and the number of nodes.

The complete statistics for EA (Table 2) shows that the value range of found P_1 minima is quite small for most functions. However, the value spread becomes larger when the P_1 values increase. Examples for this are the first two (*s1196, s1238*) and the last three functions (*cordic, misex3, seq*). Table 2 serves as a value table for Figure 5 where this effect shows: the functions with smaller P_1 rarely have visible P_1 error bars. In this logarithmic diagram only functions with large P_1 have visible error bars which means that their percentual value spread relative to the median is greater.

Table 2: Descriptive statistics for the EA

	P_1			
function	**min.**	**mean**	**median**	**max.**
s1196	2508	2626.92	2673	2783
s1238	2508	2621.40	2650	2757
s1488	352	352.00	352	352
s1494	352	352.12	352	353
s208	53	53.04	53	54
s27	16	16.00	16	16
s298	70	70.00	70	70
s344	330	331.80	330	348
s349	330	332.16	330	351
s382	230	231.32	230	239
s386	57	57.56	57	64
s400	230	230.64	230	236
s444	230	232.32	231	239
s510	153	154.88	154	163
s526	156	159.88	159	172
s526n	156	158.60	158	168
s641	1444	1498.36	1497	1580
s713	1447	1497.04	1488	1596
s820	146	146.04	146	147
s832	146	146.12	146	149
alu4	1372	1372.00	1372	1372
b12	60	60.04	60	61
clip	214	214.00	214	214
inc	27	27.00	27	27
majority	5	5.00	5	5
misex1	34	34.00	34	34
misex2	29	29.56	30	30
rd53	35	35.00	35	35
rd73	147	147.00	147	147
rd84	294	294.00	294	294
sao2	97	97.12	97	98
t481	841	867.88	841	1009
xor5	16	16.00	16	16
5xp1	79	79.00	79	79
9sym	148	148.00	148	148
cordic	8332	11349.12	11044	13218
misex3	1976	1991.28	1987	2031
seq	1744	1753.20	1752	1769
Σ	26987	29787.40	29471	31831

4.5 Comparison to Modified Sifting

We now compare our EA to the state-of-the-art heuristic for one-path minimization, MS. First, we will have a look at a graphic representation of the differences. After that, the details will be discussed using Table 3.

In Figure 6, the differences between our EA and MS are shown. For each best benchmark result of these two algorithms, the difference of nodes (abscissa value) and the negative difference of P_1 (ordinate value) was computed and plotted. Note that the ordinate is logarithmically scaled (about four orders of magnitude).

Because of the logarithmic scale, functions with one-path differences of zero are not shown. It is worth mentioning in this context that in no case the EA was worse than MS.

Many values cluster near a zero node difference while being better in the one-path difference. For these benchmarks, P_1 can be improved at little or no increase in node number. It is noteworthy that most of the time (i.e. all dots left of zero node difference), an improvement in P_1 goes with a *decrease* in node number (contrary to Example 1). For greater differences in one-paths the spread in node difference becomes larger.

For a closer inspection of benchmark results we now use Table 3. For each function, the results of the two different algorithms are shown in one row.

The upper, larger part of Table 3 contains the ISCAS89 benchmarks functions [2, 1] considered in [8]. Separated

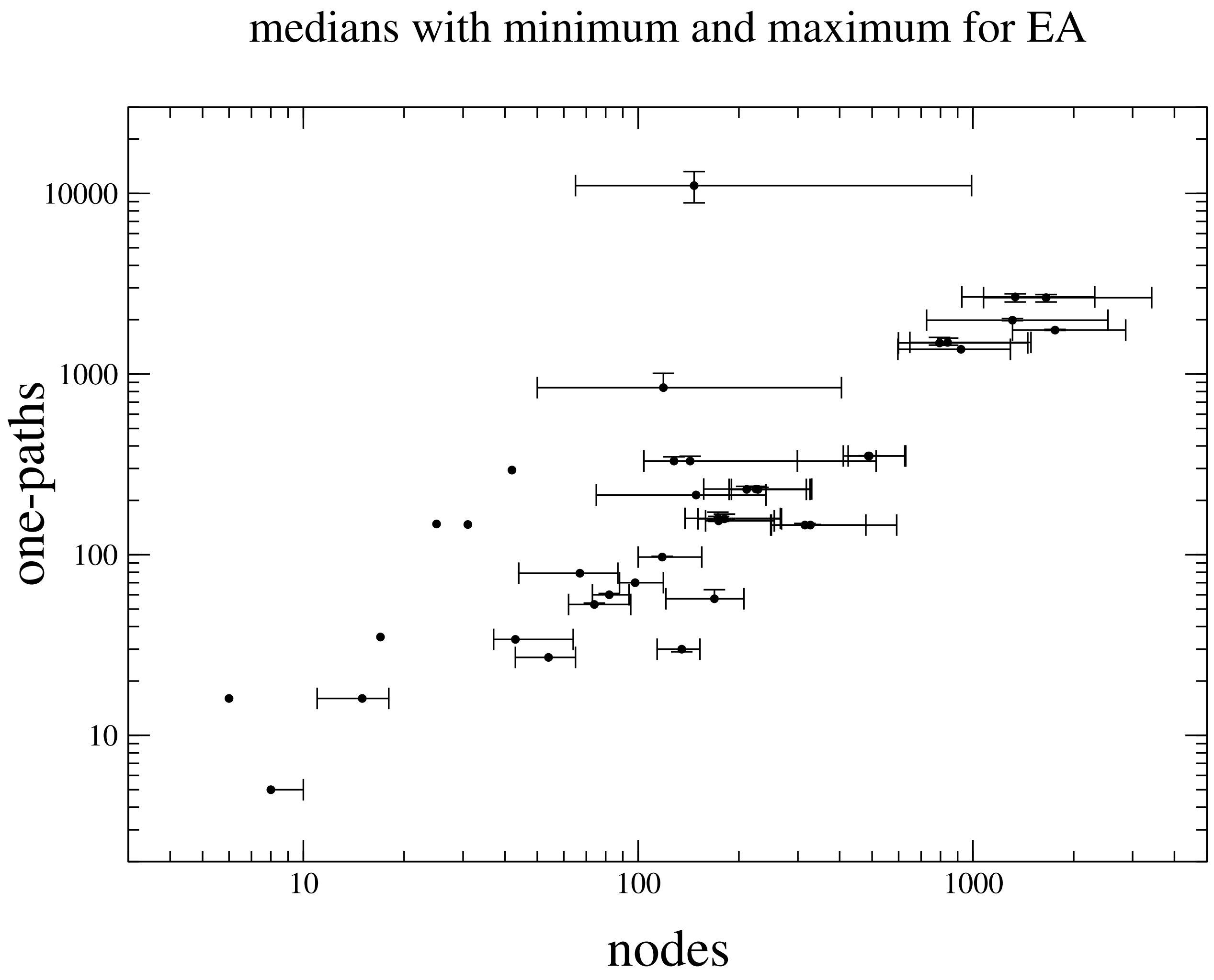

Figure 5: Qualities of benchmark set. Plot of P_1 versus nodes.

by a horizontal line and below these are functions that have been used in a recent paper [9]. The lower, smaller part of Table 3 gives three benchmarks that were deemed especially difficult. Each of the two parts of Table 3 has a statistic section at its bottom.

For comparison, the one-path minimization results from the MS algorithm [8] are shown in the three columns labeled *nodes*, P_1, and *sec* (CPU run-time in seconds).

The results of our EA shown in Table 3 are the minima of the 25 benchmark runs. The run-time given is that of one single run. If there were several runs with minimal P_1, the run with the smallest number of nodes was chosen. For 20 functions, the best EA results were better than MS in node number; in this cases, the node number was printed in boldface. It is remarkable that for these runs the number of generations is often also small: a good permutation was found early.

Normally, the EA runs much longer than MS, as was expected. But it should be mentioned that functions exist where this is not the case. The *seq* function is such a case which proved difficult for MS. Here the EA is superior not only in P_1 but also in run-time.

In the 38 benchmarks considered, the EA improves on the one-path minima found by MS in 25 cases (66 %). In all other cases EA was as good as MS, concerning minimal P_1. In 17 cases (45 %) the EA additionally improved on the number of nodes in the BDD.

Improvements by the EA in comparison to MS amounted to up to 72 % with a mean improvement of 8.4 %.

The EA is a reliable algorithm for the P_1 minimization problem. This is exemplified by the fact that the median P_1 values for the EA (see Table 2) are never worse than the results of MS in Table 3. This means that about every second run of this EA is better than MS for a given benchmark; one has to run only a handful of EA attempts to get an improvement over MS. And the best P_1 results of our EA are always better or equal to MS (boldface in Table 3).

5 Conclusions

In this paper, the number of one-paths of various Boolean functions was minimized by an EA. The most promising combination from a set of different operator weightings for that EA was chosen. This most successful parameter setting predominantly used partially matched crossover (PMX) as recombination operator. The EA with this parameter setting always found the P_1 minima of the MS method, most of the time improving on these.

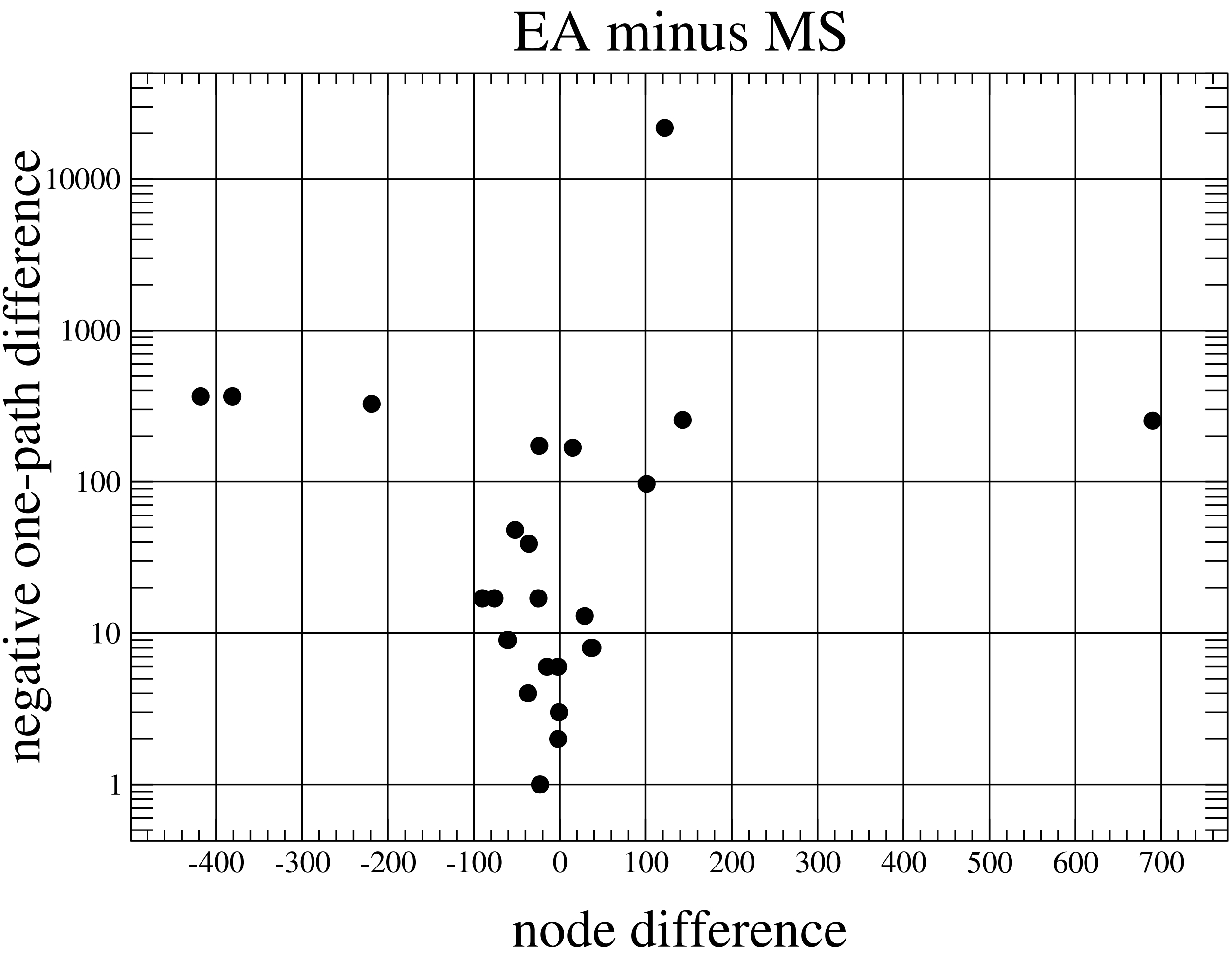

Figure 6: Differences between EA and MS. Plot of negative P_1 difference versus node difference.

6 Acknowledgements

We like to thank Görschwin Fey for providing the benchmark data of the MS algorithm. We also thank the referees for their helpful comments.

Bibliography

[1] ISCAS'89 benchmark information. *www.cbl.ncsu.edu/CBL_Docs/iscas89.html*, 1997.

[2] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.

[3] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[4] R. Bryant. Binary decision diagrams and beyond: Enabling techniques for formal verification. In *Int'l Conf. on CAD*, pages 236–243, 1995.

[5] R. Drechsler. *Evolutionary Algorithms for VLSI CAD*. Kluwer Academic Publisher, 1998.

[6] R. Drechsler and N. Drechsler. *Evolutionary Algorithms for Embedded System Design*. Kluwer Academic Publisher, 2002.

[7] E. Dubrova and D. Miller. On disjoint covers and ROBDD size. In *Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 162–164, 1999.

[8] G. Fey and R. Drechsler. Minimizing the number of paths in BDDs. In *Symposium on Integrated Circuits and System Design*, pages 359–364, 2002.

[9] G. Fey and R. Drechsler. A hybrid approach combining symbolic and structured techniques for disjoint sop minimization. In *Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI)*, pages 54–60, 2003.

[10] N. Göckel, R. Drechsler, and B. Becker. GAME: A software environment for using genetic algorithms in circuit design. In *Applications of Computer Systems*, pages 240–247, 1997.

[11] D. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addision-Wesley Publisher Company, Inc., 1989.

[12] A. Mishchenko and M. Perkowski. Fast heuristic minimization of exclusive-sums-of-products. In *Int'l Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 242–250, 2001.

[13] I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Int'l Conference on Genetic Algorithms*, pages 224–230, 1987.

[14] S. Reda, R. Drechsler, and A. Orailoglu. On the relation between SAT and BDDs for equivalence checking. In *Int'l Symp. on Quality Electronic Design*, pages 394–399, 2002.

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

[16] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.

[17] Y. Ye and K. Roy. Efficient synthesis of AND/EXOR networks. In *ASP Design Automation Conf.*, pages 539–544, 1997.

Table 3: Best results of EA compared to MS

function	#in	#out	MS nodes	MS P_1	MS sec	EA nodes	EA P_1	EA sec	EA genr.	Δ %
s1196	32	31	1523	2874	0.30	**1142**	**2508**	104.14	555	12.73
s1238	32	31	1523	2874	0.28	**1105**	**2508**	129.56	896	12.73
s1488	14	25	500	369	0.04	**410**	**352**	39.80	254	4.61
s1494	14	25	500	369	0.04	**424**	**352**	19.89	175	4.61
s208	18	9	**62**	**53**	0.02	**62**	**53**	25.02	235	0.00
s27	7	4	13	**16**	< 0.005	**11**	**16**	6.37	61	0.00
s298	17	20	91	**70**	< 0.005	**88**	**70**	17.54	165	0.00
s344	24	26	**104**	**330**	0.01	**104**	**330**	45.47	423	0.00
s349	24	26	**104**	**330**	0.01	**104**	**330**	37.10	345	0.00
s382	24	27	**152**	238	< 0.005	188	**230**	40.98	378	3.36
s386	13	13	158	61	0.01	**121**	**57**	18.08	171	6.56
s400	24	27	**152**	238	0.01	190	**230**	29.80	276	3.36
s444	24	27	**154**	243	0.01	183	**230**	34.17	313	5.35
s510	25	13	184	170	29.18	**159**	**153**	26.93	244	10.00
s526	24	27	153	162	0.01	**138**	**156**	40.59	375	3.70
s526n	24	27	153	162	0.01	**151**	**156**	28.20	260	3.70
s641	54	42	**768**	1700	0.11	911	**1444**	216.99	1707	15.06
s713	54	42	**768**	1700	0.11	1458	**1447**	189.99	1498	14.88
s820	23	24	310	155	0.07	**250**	**146**	22.06	195	5.81
s832	23	24	310	155	0.06	**249**	**146**	30.21	269	5.81
alu4	14	8	621	1545	0.17	**597**	**1372**	40.02	322	11.20
b12	15	9	**67**	**60**	< 0.005	73	**60**	17.66	167	0.00
clip	9	5	127	262	0.01	**75**	**214**	19.80	188	18.32
inc	7	9	79	66	0.01	**43**	**27**	8.86	86	59.09
majority	5	1	**8**	**5**	< 0.005	**8**	**5**	3.57	35	0.00
misex1	8	7	42	**34**	< 0.005	**37**	**34**	8.32	80	0.00
misex2	25	18	147	30	0.01	**124**	**29**	12.21	112	3.33
rd53	5	3	**17**	**35**	< 0.005	**17**	**35**	7.94	78	0.00
rd73	7	3	**31**	**147**	< 0.005	**31**	**147**	11.31	109	0.00
rd84	8	4	**42**	**294**	< 0.005	**42**	**294**	12.84	124	0.00
sao2	10	4	102	99	0.01	**100**	**97**	13.42	127	2.02
t481	16	1	**35**	1009	< 0.005	50	**841**	17.33	162	16.65
xor5	5	1	**6**	**16**	< 0.005	**6**	**16**	6.21	61	0.00
5xp1	7	10	45	82	< 0.005	**44**	**79**	10.62	102	3.66
9sym	9	1	**25**	**148**	< 0.005	**25**	**148**	11.28	109	0.00
Σ			9076	16101	30.56	8738	**14312**	1304.28	10657	
median										3.66
mean										6.47
cordic	23	2	**49**	30093	0.01	171	**8332**	84.60	747	72.31
misex3	14	14	946	2303	0.39	**727**	**1976**	43.42	325	14.20
seq	41	35	**1398**	1841	4469.27	1499	**1744**	305.67	991	5.27
Σ			2393	34237	4469.67	2379	**12052**	433.69	2063	
mean										30.59