Multi-Objective BDD Optimization with Evolutionary Algorithms

Saeideh Shirinzadeh[1] Mathias Soeken[1,2] Rolf Drechsler[1,2]

[1] Department of Mathematics and Computer Science, University of Bremen, Germany
[2] Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{saeideh,msoeken,drechsle}@cs.uni-bremen.de

ABSTRACT

Binary Decision Diagrams (BDDs) are widely used in electronic design automation and formal verification. BDDs are a canonical representation of Boolean functions with respect to a variable ordering. Finding a variable ordering resulting in a small number of nodes and paths is a primary goal in BDD optimization. There are several approaches minimizing the number of nodes or paths in BDDs, but yet no method has been proposed to minimize both objectives at the same time.

In this paper, BDD optimization is carried out as a bi-objective problem using two aforementioned criteria. For this purpose, we have exploited NSGA-II which has been proven to fit problems with a small number of objectives. Furthermore, the algorithm is facilitated with an objective priority scheme that allows to incorporate preference to one of the objectives.

Experimental results show that our multi-objective BDD optimization algorithm has achieved a good trade-off between the number of nodes and the number of paths. Comparison of the results obtained by applying priority to the number of nodes or paths with node and path minimization techniques demonstrates that the proposed algorithm can find the minimum of the preferred objective in most cases as well as lowering the other objective simultaneously.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem solving, Control methods and Search—*Heuristic methods*; J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

Keywords

Multi-objective optimization; genetic algorithm; BDD optimization

GECCO '15, July 11 - 15, 2015, Madrid, Spain

DOI: http://dx.doi.org/10.1145/2739480.2754718

1. INTRODUCTION

Binary Decision Diagrams (BDDs) are graph based data structures that allow a compact and canonical representation of Boolean functions [5, 6]. BDDs are extensively used in VLSI CAD for synthesis and verification [11]. Moreover, *Artificial Intelligence* (AI) benefits from BDDs, e.g., in software model checking [13], sparse-memory applications [19], and enhancing search methods [14].

All these applications exploit BDD optimization with respect to the number of nodes in the graph. Many VLSI CAD applications map BDDs directly to target circuits, and therefore, a smaller BDD results in smaller chip area [10]. Also the number of one-paths, i.e., paths to logical 1, is influential in several applications. As an example, in formal verification using SAT-solving, the number of required steps to solve a SAT problem can be measured by the number of paths in BDDs [24]. In logical synthesis it has been shown that a reduction in the number of paths enhances the minimization of *Disjoint-Sum-of-Product* representations which can be directly extracted from BDDs [30, 22].

Many approaches have been developed for BDD optimization that aim at minimizing the number of nodes. Exact minimization techniques can be classified as a group of these approaches. Although, exact methods can guarantee to find the optimal BDD, they suffer from a high order of complexity and run-time [11]. Hence, there is a strong motivation to employ heuristics regardless that the optimum is not guaranteed. Dynamic reordering based approaches have been widely used for this purpose. They tackle the BDD minimization problem by determining a promising variable ordering in a reasonable amount of time [25]. Simulation-based approaches such as evolutionary algorithms have also been proposed to get a higher degree of minimization. Few techniques using dynamic reordering and evolutionary computation have been developed for minimizing the number of one-paths in BDDs [15, 17]. In this paper the BDD optimization problem is conducted with respect to both criteria addressing significant cost metrics. The proposed *Multi-Objective BDD optimization* (MOB) algorithm minimizes the number of nodes and one-paths at the same time without any loss of quality compared to a single objective method. The proposed approach is also capable of handling specific applications' preferences where either nodes or one-paths are of higher importance.

2. BACKGROUND

This section describes concepts of multi-objective optimization and BDDs and briefly introduces state-of-the-art BDD optimization approaches.

2.1 Basic Concepts

2.1.1 Multi-Objective Optimization

An arbitrary optimization problem with m objectives can be defined as

$$\min \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))^{\mathrm{T}},$$

where solution $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ is described as a decision vector from the decision space $\Omega \subseteq \mathbb{R}^n$, and $\mathbf{f} : \Omega \to \Lambda$ is a set of m objective functions which evaluates a specific solution by mapping it to objective space $\Lambda \subseteq \mathbb{R}^m$.

Let $\mathbf{x}, \mathbf{y} \in \Omega$, then the *Pareto-dominance*, denoted by $\prec$, can be defined as

$$\mathbf{x} \prec \mathbf{y} :\Leftrightarrow \exists j \in \{1, 2, \ldots, m\} : \\ f_j(\mathbf{x}) < f_j(\mathbf{y}) \wedge \forall i \neq j : \ f_i(\mathbf{x}) \leq f_i(\mathbf{y}).$$

According to the definition above, an optimal solution, also called *non-dominated* is a solution that is not dominated by any other solution. The set of optimal solutions is the so-called *Pareto set*, denoted by $\omega \subseteq \Omega$. It must fulfill the property

$$\forall p \in \omega : \nexists q \in \Omega : q \prec p.$$

The *Pareto front* λ is defined as the image of the Pareto set in the objective space, i.e, $\lambda = f(\omega) \subseteq \Lambda$. Finding a well converged set of solutions to the Pareto set is the goal of a *Multi-Objective Evolutionary Algorithm* (MOEA). The final output of a MOEA is a set of non-dominated solutions, so-called *Pareto set approximation*.

2.1.2 Binary Decision Diagrams

Binary Decision Diagrams (BDD) can provide a compact graph representation for most of the practical relevant Boolean functions. The main reason of the compactness of BDDs is that such functions often have common *subfunctions* where for a given function $f(x_1, x_2, \ldots, x_n)$ its subfunctions are $f(0, x_2, \ldots, x_n)$ and $f(1, x_2, \ldots, x_n)$ as well as their subfunctions and so on. A function $f(x_1, \ldots, x_n)$ is called a *bead* if it depends on its first variable, or in other words, if its truth table representation is not of the form $\alpha\alpha$ for any bitstring α of length 2^{n-1} [20]. The nodes of a BDD for a Boolean function f are all subfunctions of f which are beads. The size of a BDD can further be reduced with complemented edges such that a subfunction and its complement can be represented by the same node [3].

It can be easily seen that the set of subfunctions changes when the order of input variables is changed. In fact the *variable ordering* of a BDD can have a significant impact on the number of nodes in a BDD, referred to as N, and also the number of *one-paths*, referred to as P, which are all paths from the start vertex to the 1-terminal that have an even number of complemented edges (including the complement of the start vertex). As an example, the function

$$\bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2x_4 \vee x_1x_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3x_4$$

from [15] and depicted in Fig. 1 has its minimal number of nodes, $N = 5$, for the variable ordering $x_4 < x_3 < x_3 < x_1$. The minimal number of one-paths, $P = 4$, is obtained for the variable ordering $x_2 < x_1 < x_4 < x_3$.

Improving the variable ordering to find a BDD with at most N nodes is known to be NP-complete [2] and several heuristics have been proposed that are illustrated in the next section.

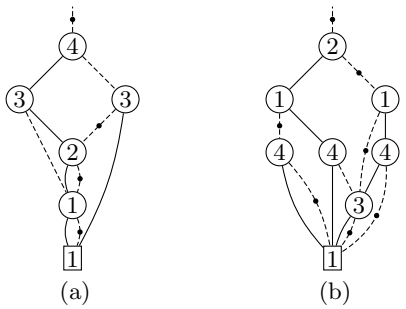

Figure 1: BDDs for the function $\bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2x_4 \vee x_1x_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3x_4$

2.2 Related Work

BDD optimization techniques aim to find an optimal variable ordering which leads to the minimum BDD with respect to the optimization criterion. There are several approaches for exact minimization guaranteeing to find the minimal BDD [12, 16, 18]. Nevertheless, high complexity of the exact methods is a reason for heuristic methods to be of higher interest. Sifting [25] is a well known node minimization algorithm based on a hill-climbing framework. The technique is based on a rule that swaps two adjacent variables in a BDD without changing the function. Sifting uses this feature by moving each variable downwards and upwards in the variable order. The BDD size resulting from reordering every variable is recorded during this procedure and finally the variable is moved back and fixed to the position where the BDD with the minimum number of nodes was found. Results obtained by sifting can be far from the optimum when the number of inputs increases. Employing approaches such as *Simulated Annealing* (SA) and *Evolutionary Algorithms* (EAs) for BDD node minimization, the number of nodes can even decrease to half of that achieved by sifting for large circuits [1, 9]. Since SA has been used for comparison with MOB in our experimental studies, its framework is briefly discussed in the following. At first, a variable ordering is generated randomly and the nodes of the resulting BDD are counted. Before a termination criterion is satisfied, a randomly chosen ordering is accepted instead of the current variable permutation if it leads to a BDD with a lower number of nodes. Otherwise, the new variable ordering might be accepted in a probabilistic manner. In fact, SA allows to accept a variable ordering with a certain probability in order to make the algorithm capable of escaping a local minimum.

The majority of BDD optimization approaches are based on node minimization, however it has been proven that the number of BDD paths are also important in some applications such as SAT-solving or synthesis [11]. *Modified Sifting* (MS) [15] is a BDD optimization method aiming at minimizing the number of one-paths. The framework of MS is the same as sifting with the objective being set to the number of one-paths. Similar to sifting, MS examines just a few possibilities of the whole search space that makes it faster compared to simulation based methods such as SA and EA. Consequently, the BDDs found by MS might have a considerable higher number of one-paths than the optimum when the number of

variables increases. An EA for BDD path minimization was proposed in [17]. EA has a better performance than MS for the same reasons mentioned above. EA applies roulette wheel mating selection for creating an offspring with a cardinality of half of the current population. In each iteration, after employing variation operators the offspring is produced and then the new population is created by combining the offspring with the best half of the individuals in the current population.

All of the BDD optimization techniques explained above optimize BDDs with respect to one of the cost metrics. An approach introduced in [21] employs sifting and MS separately and then stores the number of nodes and one-paths returned by both methods. Thereafter, a comparison operator checks the node counts found by two approaches. If they are equal, the optimized BDD is characterized by the BDD found by MS. Otherwise, the same scenario is repeated for equal one-path counts but the BDD found by sifting is selected this time. In the case that both objectives in the BDDs found by two methods are different, the user is supposed to choose between the resulting BDDs. Indeed, the algorithm introduced in [21] chooses between results obtained by sifting and MS instead of a multi-objective optimization.

3. MULTI-OBJECTIVE BDD OPTIMIZATION

3.1 Overview

The proposed multi-objective BDD minimization approach is a non-dominated sorting based algorithm structurally similar to NSGA-II [7] with variation operators specifically designed for escaping invalid variable permutations. NSGA-II has been shown excellent performance on problems with two or three objectives. However, as a Pareto-dominance algorithm it fails to rank solutions in the presence of many objectives. Since BDD optimization is characterized as a bi-objective problem in this paper, NSGA-II would be sufficient and more reasonable than paying high penalties when using many objective optimization approaches, e.g., [28].

In order to keep the paper self-contained, a brief explanation of the relation employed by NSGA-II to rank a population is given. According to non-dominated sorting, all individuals should be assigned to a level of dominance that is equal to their overall fitness values used for ranking the population. To find all members of the first non-dominated front, each individual should be compared with every other individual in the population to find if it is non-dominated. This procedure can be completed by exploiting a fast non-dominated sorting scheme to find all levels of dominance. Fast non-dominated sorting decreases computational complexity by counting the number of individuals which dominate or are dominated by each individual in the population. To preserve a well-distributed optimal set, density of individuals is also considered besides non-domination rank in the selection process. The density estimation metric so-called crowding distance is calculated to discriminate between individuals belonging to the same non-dominated front. Thus, between two individuals with different non-dominated ranks, the individual with lower rank is preferred. Otherwise, the individual with greater value of crowding distance is preferred in the case of equal ranks.

The general framework of the MOB algorithm is described in Algorithm 1. First, a random parent population P_0 of size N is initialized. Each individual of the population is characterized by a permutation of the variable indices of the BDD. Objective values are assigned to the population by counting the number of nodes and one-paths. Then, the population is sorted based on non-domination. Steps 6–18 are iterated until the stopping criterion is satisfied. After applying binary tournament, recombination, and mutation operators an offspring Q_t of size equal to the parent population is created in step 6. To ensure elitism, Q_t is combined with the current population P_t resulting in a new population R_t. In step 9, non-dominated sorting is employed to classify R_t into different non-domination levels (F_1, F_2, and so on). Thereafter, individuals are added to P_{t+1} starting from the first non-dominated front F_1. This procedure is continued to fill population P_{t+1} with subsequent non-dominated fronts until their sizes are smaller than the free available slots in P_{t+1} (step 17), formally expressed as

$$P_{t+1} \leftarrow P_{t+1} \cup F_i[1 : (N\text{–} \mid P_{t+1} \mid)].$$

Otherwise, the lesser crowded individuals in the current front F_i are chosen after sorting the front according to the crowding distance.

Algorithm 1 Framework of the general MOB

```
1: P_0 ← InitializePopulation
2: CountNodesOne-paths(P_0)
3: {F_1, F_2, ...} ← Non-dominated-sort(P_0)
4: t ← 0
5: while the stopping criterion is not met do
6:     Q_t ← MakeOffspringPopulation(P_{t+1})
7:     CountNodesOne-paths(Q_t)
8:     R_t ← P_t ∪ Q_t
9:     {F_1, F_2, ...} ← Non-dominated-sort(R_t)
10:    P_{t+1} ← ∅
11:    i ← 1
12:    while |P_{t+1}| + |F_i| ≤ N do
13:        P_{t+1} ← P_{t+1} ∪ F_i
14:        i ← i + 1
15:    end while
16:    CrowdingDistanceSort(F_i)
17:    P_{t+1} ← P_{t+1} ∪ F_i[1 : (N - |P_{t+1}|)]
18:    t ← t + 1
19: end while
```

3.2 Variation Operators

In this section, the employed variation operators are introduced. The offspring population is produced by recombination and mutation operators. Recombination includes specific crossover operators which avoid omissions and duplication in variable indices. Indeed, the employed crossover operators ensure that the permutations remain valid. Three crossover operators are used in the proposed BDD optimization algorithm including partially matched crossover, inversion and reproduction. Partially matched crossover PMX [23] is performed by randomly choosing two crossover points which break two parents in three sections. Two children are produced by combination of sections from both parents such that each previously used variable index is substituted for a new one to guarantee valid permutations. Inversion introduced in [17] breaks each parent into three sections by choosing two random positions. Then, one child is produced from each parent by inverting the order of indices in a randomly chosen part. The last employed crossover operator only copies each

parent with no genetic change. That means the created children are identical to their parents.

After recombination, three mutation operators all described in [17] are employed such that one of them might affect a child according to given probabilities. The first operator exchanges two random variable indices of the individual. The second operator applies the first mutation technique twice on the same permutation. In the third mutation operator a position is chosen randomly and its content is exchanged with an adjacent index.

3.3 Priority

In many real world applications, one or more objectives are of higher level of significance. In such optimization problems, although it is desirable to minimize all objectives, it is preferred to escape penalties for objectives with higher priorities as a cost of improving lesser important criteria. Hence, the proposed MOB optimization approach is equipped with the ability to handle priority in order to meet requirements of specific applications. For instance, the number of nodes is more important than the number of one-paths in formal verification. On the other hand, the number of paths are regarded as an influential objective in SAT-solving.

Several techniques have been developed to model priorities in a multi-criteria optimization problem. Weighted sum is a widely used classical method that is able to handle objective priorities. It scalarizes the set of objectives into a fitness value by multiplying each objective with a user supplied weight. For a solution $\mathbf{x} \in \Omega$, the fitness function f($\mathbf{x}$) is given by

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i.x_i.$$

It is obvious that larger coefficients, denoted by ω_i, represent higher objective priorities in the fitness value. Another approach so-called *priority preferred* is proposed in [26]. It incorporates a lexicographic ordering of objectives with relation *preferred* that is a refinement of Pareto-dominance [8]. In this work, the model explained in [26] is adapted to non-domination instead of *preferred*. Let $p = \{p_1, p_2, \ldots, p_m\}$ be a priority vector determining priorities assigned to the objectives for an m-objective problem. Each component p_i, $i \in \{1, 2, \ldots, m\}$ can adopt values from the set {1, 2, ...,n}, $n \leq m$ (n is equal to m in case that all objectives have different priorities). Considering a minimization problem, we assume that a lower value of p_i means objective i is of higher priority. Given two solutions $\mathbf{x}$, $\mathbf{y} \in \Omega$, $\mathbf{x}|_j$ and $\mathbf{y}|_j$ represent subvectors of $\mathbf{x}$ and $\mathbf{y}$ only including objective functions with priority of j, priority-dominance is defined as

$$\mathbf{x} \prec_p \mathbf{y} :\Leftrightarrow \exists j \in \{1, 2, \ldots, n\} : \mathbf{x}|_j \prec \mathbf{y}|_j \wedge \forall k < j : \mathbf{y}|_k \not\prec \mathbf{x}|_k.$$

More informally, the relation defined above employs Pareto-dominance to compare objective functions with equal priorities. In other words, $\mathbf{x}$ priority-dominates $\mathbf{y}$ if there is a subvector of objective functions with identical priority in $\mathbf{x}$ that dominates the corresponding subvector in $\mathbf{y}$, and at the same time $\mathbf{x}|_j$ is not dominated by any subvector of priority value higher than j in $\mathbf{y}$.

Priority-dominance can be defined more simply in the bi-objective BDD optimization problem. In this case, the priority vector consists of two values, i.e., 1 for objective with higher significance and 2 for the other objective. The priority subvector for each individual is equal to the corresponding objective function representing the number of nodes or one-paths. The procedure to compare individuals according to a given priority vector is described in Algorithm 2. In steps 6–13, an individual $\mathbf{x}$ is compared with $\mathbf{y}$ based on priority-dominance. For this purpose, relation dominance is applied to the corresponding objective functions with high priority in $\mathbf{x}$ and $\mathbf{y}$. Obviously, if these values are different $\mathbf{x}$ priority-dominates or is priority-dominated by $\mathbf{y}$. Otherwise, the other objective is taken into account to discriminate between individuals. In order to determine the non-domination front for each individual, two entities are also calculated in parallel with the comparisons. S_x represents the set of solutions which are dominated by individual $\mathbf{x}$, and n_x is the number of individuals dominating $\mathbf{x}$. Thereafter, these entities can be used for fast non-domination sorting as described in [7].

Algorithm 2 Priority-dominance

```
1: GetPriorityVector(p)▷ To determine priority subvectors
2: for all x ∈ Ω do
3:     S_x ← ∅            ▷ Set of individuals dominated by x
4:     n_x ← 0   ▷ The number of individuals dominating x
5:     for all y ∈ Ω do
6:         i ← 1
7:         while i ≤ 2 do
8:             if x|_i ≺ y|_i then
9:                 S_x ← S_x ∪ {y}
10:            else if y|_i ≺ x|_i then
11:                n_x ← n_x + 1
12:            else
13:                i ← i + 1
14:            end if
15:        end while
16:    end for
17: end for
```

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

In order to evaluate the performance of MOB and compare it with other discussed existing approaches, several experiments are carried out on a benchmark set including 25 Boolean functions. The functions are taken from ISCAS89 [4] and LGSynth91 [29] benchmark sets with a range of input variables from 7 to 54. MOB has been run 30 times independently for each function. In all experiments, the population size is set to 100 and the algorithm terminates after 1000 iterations. Similarly to EA, the probabilities of crossover operators are set to 0.98, 0.01, and 0.01 for PMX, inversion and reproduction, respectively. Three mutation operators have an overall probability of $1/n$, where n is the number of variables of the BDD. The CUDD package [27] is used for BDD representation and comparison of results with node minimization approaches.

4.2 Performance Assessment

To evaluate the performance of the general MOB algorithm, statistical studies have been performed on the final populations for all thirty runs of each function. Complete statistics representing the average values, the smallest and

Table 1: Descriptive statistics for MOB

Function	Inputs	Outputs	Min		Mean		Median		Max	
			N	P	N	P	N	P	N	P
s1196	32	31	611	2581	677.37	2922.21	675	2857	1014	4333
s1238	32	31	627	2518	707.34	2827.9	679	2806	1057	4479
s1488	14	25	373	352	396.17	365.45	402	357	464	517
s208	18	9	50	53	59.52	62.67	60	61	63	202
s27	7	4	10	16	10	16	10	16	10	16
s298	17	20	74	70	74.51	73.4	74	74	77	74
s344	24	26	104	330	104.73	331.45	104	330	110	346
s382	24	27	121	230	138.73	255.82	136	247.5	208	332
s386	13	3	109	57	114.16	64.89	113	65	133	82
s400	24	27	121	230	142.147	252.05	135	247	188	332
s444	24	27	119	230	134.95	261.88	132	258	188	340
s510	25	13	146	153	149.49	166.78	150	159	156	183
s526	24	27	116	157	128.74	164.15	129	161	147	197
s641	54	42	461	1512	533.51	1594.76	527	1593	604	1793
s713	54	42	435	1523	527.23	1647.11	517	1627	631	1936
s820	23	24	221	146	224.94	157.13	224	151	240	180
s832	23	24	220	146	224.75	158.37	225	152	235	179
alu4	14	8	564	1372	573.19	1428.55	572	1372	644	1599
clip	9	5	75	214	75	214	75	214	75	214
misex1	8	7	35	34	35.4	35.24	35	36	37	36
sao2	10	4	81	97	84.43	102.15	85	102	96	111
t481	16	1	21	841	21.94	903.31	21	841	32	1009
cordic	23	2	43	9440	49.41	20260.62	47	25797	82	34393
misex3	14	14	480	1973	545.74	2120.83	536	2056	751	3046
seq	41	35	1208	1766	1275.76	1901.32	1270	1883	1461	2301
$\sum$			6425	26041	7009.16	38288.04	6933	43462.5	8703	58230

the greatest observations of results are shown in Table 1. It is worth mentioning that for a function with four or more number of variables, a BDD minimal in both metrics of nodes and paths does not exist necessarily [15]. Thus, the values of nodes and one-paths achieved by MOB might represent the real optimum for one or both objectives or even none of them. Indeed, with no priority to nodes or one-paths, results shown in Table 1 are supposed to express a good trade-off between objectives.

In order to analyze the descriptive statistics of results obtained by the general bi-objective MOB, we assume the minimum found for each objective as a metric roughly estimating the optimal value. Statistics of the number of nodes (N) and one-paths (P) shown in Table 1 demonstrate that the average values represented by mean and median are closer to the minimum of the number of nodes and paths rather than the maximum values found in the whole runs. For instance, for function s1196 mean values for the number of nodes and one-paths are only 10.86% and 13.22% greater than their minimum values, respectively. The worst observations of the number of nodes and one-paths are 70.37% and 67.88% greater than minimums found for the corresponding objectives. This property in the first three statistics is especially apparent for the functions resulting in larger BDDs in size and number of paths. Considering the effect of maximum number of nodes and one-paths on worsening their average values, the discussion above reflects the fact that the final populations are of high quality in comparison with the best ever found values. In general, the number of nodes and paths of the BDDs found by MOB are mostly spread over values expressing an optimal BDD which indicates the quality of results.

4.3 Comparison of Results

As discussed before, the number of nodes or one-paths of BDDs resulting by the general MOB algorithm does not represent the optimum value for each objective necessarily. Instead, the minimum BDD, either in the number of nodes or paths, is accessible for a single-objective method. Therefore, in order to make a fair comparison the results of MOB with priority to the number of nodes or one-paths are compared with the introduced node or path minimization methods, respectively. For this purpose, the BDDs possessing the smallest values for the preferred objective in all 30 runs are chosen as the results of the prioritized MOBs shown in Table 1 and Table 2.

In Table 2, the BDDs found by node minimization techniques discussed in Section 2.2, sifting and SA, are compared to the optimum BDDs obtained by MOB with priority to the number of nodes. It can be easily concluded from the table that the number of BDD nodes found by sifting for the 25 benchmark circuits are far away from the values returned by the two other approaches except for a few cases. However, sifting obtains smaller one-path counts than the compared methods for a couple of functions that is an unintended result of lower capability of finding the minimized BDDs. On the other hand, SA shows high performance by achieving BDDs with the minimum number of nodes for most of the functions. Considering all results shown in Table 2, MOB with priority to the number of nodes shows obviously higher quality than

Table 2: Comparison of results with priority to the number of nodes with existing node minimization approaches

Function	Sifting [25]		SA [1]		MOB	
	N	*P*	*N*	*P*	*N*	*P*
s1196	642	**3511**	600	4079	**599**	3907
s1238	642	**3511**	600	4079	**599**	3909
s1488	391	551	**369**	582	373	**407**
s208	61	**79**	**41**	1867	**41**	1867
s27	**10**	17	**10**	17	**10**	**16**
s298	78	**73**	**74**	74	**74**	74
s344	**104**	**330**	**104**	**330**	**104**	**330**
s382	121	**315**	**119**	332	**119**	332
s386	123	70	**110**	65	**110**	**58**
s400	121	315	**119**	332	121	**292**
s444	161	447	**119**	**332**	**119**	**332**
s510	165	206	148	192	**146**	**181**
s526	141	368	**113**	207	115	**189**
s641	629	2164	408	2883	**385**	**2671**
s713	629	2164	408	2883	**389**	**2617**
s820	258	184	**221**	180	**221**	**177**
s832	258	184	**221**	180	**221**	**151**
alu4	804	2876	**564**	**1430**	**564**	**1430**
clip	87	326	**75**	275	**75**	**214**
misex1	**35**	39	**35**	39	**35**	**36**
sao2	86	**97**	**81**	130	**81**	111
t481	**21**	1009	**21**	1009	**21**	**841**
cordic	43	38482	43	38482	**42**	**27329**
misex3	602	2654	**478**	3485	**478**	**3408**
seq	2163	62587	**1193**	2265	**1193**	**2259**
$\sum$	8325	122559	6274	65729	**6235**	**53228**

Table 3: Comparison of results with priority to the number of one-paths with existing one-path minimization approaches

Function	MS [15]		EA [17]		MOB	
	N	*P*	*N*	*P*	*N*	*P*
s1196	1523	2874	1142	**2508**	**1087**	2509
s1238	1523	1874	1105	**2508**	**1064**	2509
s1488	500	369	410	**352**	**408**	**352**
s208	**62**	**53**	**62**	**53**	**62**	**53**
s27	13	**16**	11	**16**	**10**	**16**
s298	91	**70**	88	**70**	**76**	**70**
s344	**104**	**330**	**104**	**330**	**104**	**330**
s382	**152**	238	188	**230**	188	**230**
s386	158	61	121	**57**	**113**	**57**
s400	152	238	190	**230**	**188**	**230**
s444	**154**	243	188	**230**	188	**230**
s510	184	170	159	**153**	**155**	**153**
s526	153	162	**138**	**156**	**138**	157
s641	768	1700	911	**1444**	**611**	1447
s713	768	1700	1458	**1447**	**616**	1452
s820	310	155	250	**146**	**230**	**146**
s832	310	155	249	**146**	**230**	**146**
alu4	621	1545	597	**1372**	**572**	**1372**
clip	127	262	**75**	**214**	**75**	**214**
misex1	42	**34**	37	**34**	**36**	**34**
sao2	102	99	100	**97**	**86**	**97**
t481	35	1009	50	**841**	**21**	**841**
cordic	**49**	30093	171	**8332**	73	9440
misex3	946	2303	727	1976	**577**	**1973**
seq	1398	1841	1499	**1744**	**1391**	**1744**
$\sum$	10245	47594	10030	**24684**	**8299**	25802

the other two methods with respect to both metrics. The sum of results for the whole benchmark set obtained by MOB reveals a decrease of 2090 and 39 in the number of nodes in comparison with sifting and SA. Moreover, MOB with priority to the number of nodes finds BDDs with noticeably lower one-path counts for the majority of the benchmark functions. To make a more precise comparison, the sum of the number of one-paths in MOB is 97.36% and 19.02% smaller than the corresponding values in sifting and SA, respectively. Indeed, MOB with priority to the number of nodes finds better minimized BDDs than the well known node minimization techniques besides achieving a large reduction in the number of one-paths.

Since the number of nodes are assumed to be of higher importance in this experiment, finding minimal BDDs in size is the principal goal of the MOB with priority to the number of nodes. Nonetheless, there are three functions for which MOB fails to find the BDD representations with minimum number of nodes obtained by SA. For functions *s1488*, *s400*, and *s526*, BDDs found by MOB have in total 8 more nodes than the three BDDs resulted by SA. On the other hand, the sum of one-path counts for the mentioned functions shows a decrease of 233 compared to the total number of one-paths in the corresponding BDDs found by SA. Considering the fact that MOB with priority to the number of nodes is a multi-objective algorithm, finding BDDs with a small increase in the node counts as a fair cost of reducing the number of paths is acceptable.

Table 3 shows the comparison of results obtained by MOB with priority to the number of one-paths with BDDs found by MS and EA. It can be clearly seen that results obtained by MS are dominated by EA and MOB. Therefore, the quality of BDDs optimized by EA and MOB are compared in the following. According to the table, MOB has found the minimum number of one-paths that were also found by EA for the majority of benchmark circuits. However, for a few functions BDDs obtained by MOB have higher one-path counts than the corresponding BDDs by EA. For functions *s1196*, *s1238*, *s641*, and *s713* the BDDs found by MOB show few increases in the numbers of one-paths while the node counts are considerably lowered. Only the function *cordic* shows a high difference between the numbers of one-paths resulted by EA and MOB. The BDD representing *cordic* found by MOB is 13% larger than the BDD resulted by EA with respect to the number of one-paths. It is also worth mentioning that the number of nodes in the BDD found by MOB for the function *cordic* is decreased to less than half of the nodes in the BDD found by EA. Although MOB has almost failed to find the minimal BDD wih respect to the significant objective for *cordic*, it has compensated this failure with a fair reduction in the other objective. For function *misex3*, MOB has even found a BDD noticeably smaller in size with a lower number of one-paths compared to the BDD obtained by EA.

The total number of one-paths obtained by MOB is almost equal to half of the one-paths achieved by MS with a reduction of 18.99% in the number of nodes. In comparison with EA, MOB shows an increase of 4.52% in the one-path counts that is mainly caused by result of *cordic*. Also, MOB has 17.26% less BDD nodes compared to EA for the whole

benchmark set. Although, the results obtained by MOB are slightly worsened in the number of one-paths compared to EA, MOB has noticeably lowered the number of BDD nodes. In general, it is fair to say that MOB with priority to the number of one-paths shows a high performance with respect to both objectives in comparison with existing node minimization techniques.

In order to have a thorough assessment on the experimental results, multiplication of the number of nodes and one-paths is considered as a metric representing the quality of BDDs. In fact, a lower *Node-One-Path-product* (NOP) means a better trade-off between both optimization criteria. The values of NOPs obtained by all types of MOB and the other methods are shown in Table 4. According to the table, the general MOB with no priority has achieved the minimum sum of all NOP values among all other approaches that reveals its high performance as a multi-objective technique. The sum of NOP values of the general MOB is decreased to 92.78%, 27.02%, 43.84%, and 32.28% of the corresponding values in sifting, SA, MS, and EA, respectively. Moreover, the two other versions of MOB have also considerable lower NOP sums than that of node or path minimization methods. In other words, priority based MOB types find a good trade-off that is usually the minimal BDD with respect to the objective of higher importance besides reducing the value of the other objective. To evaluate the effect of priority on MOB, sum of results with priority to nodes and one-paths are respectively compared to the sum of median values of nodes and one-paths obtained by the general MOB. The integration of results shown in Tables 1, 2, and 3 describes that after applying priority to the number of nodes and one-paths, MOB has achieved 10.06% and 40.63% less BDD nodes and one-paths than the general MOB. Since one-path counts are spread over a larger range than nodes, results of MOB with priority to the number of one-paths show higher level of BDD abstraction in respect to one-paths. In general, incorporating priority with MOB has improved the performance of the algorithm with respect to the user preferred objective in addition to preserving favorable multi-objective features.

5. CONCLUSION

Since many applications in electronic design automation and formal verification successfully use BDDs, their optimization is of high interest. The existing BDD optimization approaches either target the number of nodes or the number of paths. In this paper, we have proposed a bi-objective evolutionary algorithm to find minimal BDDs in both cost metrics. The proposed approach also benefits from an objective priority facility allowing the user to give higher importance to one of the optimization criteria according to the application's requirements. Experimental results show that the multi-objective BDD optimization approach is able to find more compact BDDs considering both objectives in comparison with the existing BDD optimization methods.

Acknowledgments

This work was supported by the German Research Foundation (DFG) (DR 287/23-1) within a Reinhart Koselleck project and by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

6. REFERENCES

[1] B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. In *Int'l Workshop on Logic Synth*, 1995.

[2] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[3] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th annual Design Automation Conference (DAC)*, pages 40–45, 1990.

[4] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems*, pages 1929–1934, 1989.

[5] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[6] R. E. Bryant. Binary decision diagrams and beyond: Enabling technologies for formal verification. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 236–243, 1995.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[8] N. Drechsler, R. Drechsler, and B. Becker. Multi-objective optimisation based on relation *Favour*. In *Proceedings of the first International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, volume 1993, pages 154–166, 2001.

[9] R. Drechsler, B. Becker, and N. Göckel. A genetic algorithm for variable ordering of OBDDs. In *IEE Proceedings of Computers and Digital Techniques*, volume 143(6), pages 364–368, 1996.

[10] R. Drechsler, J. Shi, and G. Fey. Synthesis of fully testable circuits from bdds. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 23(3):440–443, 2004.

[11] R. Ebendt, G. Fey, and R. Drechsler. Advanced BDD optimization. Springer, 2005.

[12] R. Ebendt, W. Günther, and R. Drechsler. An improved branch and bound algorithm for exact BDD minimization. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 22(12):1657–1663, 2003.

[13] S. Edelkamp and T. Mehler. Byte code distance heuristics and trail direction for model checking Java programs. In *Workshop on Model Checking and Artificial Intelligence (MoChArt)*, pages 69–76, 2003.

[14] S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *Lecture Notes in Computer Science*, volume 1504, pages 81–92. Springer, 1998.

[15] G. Fey and R. Drechsler. Minimizing the number of paths in BDDs: Theory and algorithm. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 25(1):4–11, 2006.

[16] S. J. Friedman and K. J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *Proceedings of the 24th ACM/IEEE Design Automation Conference (DAC)*, pages 348–356, 1987.

[17] M. Hilgemeier, N. Drechsler, and R. Drechsler. Minimizing the number of one-paths in BDDs by an

Table 4: Comparison of Node-One-Path-products (NOPs)

Function	Sifting [25]	SA [1]	MS [15]	EA [17]	MOB priority to one-paths	MOB priority to nodes	MOB general
s1196	2254062	2447400	4377102	2864136	2727283	2340293	**1906860**
s1238	2254062	2447400	4377102	2771340	2669576	2341491	**1889888**
s1488	215441	217341	184500	144320	143616	151811	**142065**
s208	4819	76547	**3286**	**3286**	**3286**	76547	**3286**
s27	170	170	208	176	**160**	**160**	**160**
s298	5694	5476	6370	6160	**5320**	5476	5476
s344	**34320**	**34320**	**34320**	**34320**	**34320**	**34320**	**34320**
s382	38115	39508	36176	43240	43240	39508	**33082**
s386	8610	7150	9638	6897	6441	**6380**	**6380**
s400	38115	39508	36176	43700	43240	35332	**32562**
s444	71967	39508	37422	43240	43240	39508	**33082**
s510	33990	28416	31280	24327	23715	26426	**23562**
s526	51888	23391	24786	21528	21666	21735	**20640**
s641	1361156	1176264	1305600	1315484	884117	1028335	**806837**
s713	1361156	1176264	1305600	2109726	894432	1018013	**802382**
s820	48246	39780	48050	36500	33580	39117	**33222**
s832	48246	39780	48050	36354	33580	33371	**32928**
alu4	2312304	806520	959445	819084	**784784**	806520	**784784**
clip	28362	20625	33274	**16050**	**16050**	**16050**	**16050**
misex1	1365	1365	1428	1258	**1224**	1260	1260
sao2	**8342**	10530	10098	9700	**8342**	8991	**8342**
t481	21189	21189	35315	42050	**17661**	**17661**	**17661**
cordic	1654726	1654726	1474557	1424772	689120	1147818	**686774**
misex3	1597708	1665830	2178638	1436552	1138421	1629024	**1116951**
seq	135375681	2702145	2573718	2614256	2425904	2694987	**2305520**
$\sum$	148829734	14721153	19132139	15867306	12648099	12575456	**10744074**

evolutionary algorithm. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1724–1731, 2003.

[18] N. Ishiura, H. Sawada, and S. Yajima. Minimazation of binary decision diagrams based on exchanges of variables. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 472–475, 1991.

[19] R. M. Jensen, E. A. Hansen, S. Richards, and R. Zhou. Memory-efficient symbolic heuristic search. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 304–313, 2006.

[20] D. E. Knuth. *The Art of Computer Programming*, volume 4A. Addison-Wesley, 2011.

[21] K. Lu, Y. Ramin, Y. Edwin, and K. Constantinos. Structural optimization of reduced ordered binary decision diagrams for SLA negotiation in IaaS of cloud computing. In *Proceedings of the 10th International Conference on Service-Oriented Computing*, pages 268–282, 2012.

[22] A. Mishchenko and M. Perkowski. Fast heuristic minimization of exclusive-sums-of-products. In *5th Int'l Workshop on Applications of the Real-Muller Expansion in Circuit Design*, 2001.

[23] I. Oliver, D. Smith, and J. Holland. Study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the second International Conference on Genetic Algorithms*, pages 224–230, 1987.

[24] S. Reda, R. Drechsler, and A. Orailoglu. On the relation between SAT and BDDs for equivalence checking. In *Proceedings of the International Symposium on Quality Electronic Design*, pages 394–399, 2002.

[25] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 42–47, 1993.

[26] F. Schmiedle, N. Drechsler, D. Große, and R. Drechsler. Heuristic learning based on genetic programming. *Genetic Programming and Evolvable Machines*, 3(4):363–388, 2002.

[27] F. Somenzi. CUDD: CU Decision Diagram package release 2.5.0. University of Colorado at Boulder, 2012.

[28] A. Sülflow, N. Drechsler, and R. Drechsler. Robust multi-objective optimization in high dimensional spaces. In *Proceedings of the 4th International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, pages 715–726, 2006.

[29] S. Yang. Logic synthesis and optimization benchmarks user guide: Version 3.0. Microelectronics Center of North Carolina (MCNC), 1991.

[30] Y. Ye and K. Roy. A graph-based synthesis algorithm for AND/XOR networks. In *Proceedings of the 34th Design Automation Conference (DAC)*, pages 107–112, 1997.