An Adaptive Prioritized ε-Preferred Evolutionary Algorithm for Approximate BDD Optimization

Saeideh Shirinzadeh[1] Mathias Soeken[2] Daniel Große[1,3] Rolf Drechsler[1,3]
[1]Department of Mathematics and Computer Science, University of Bremen, Germany
[2]Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
[3]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{s.shirinzadeh,grosse,drechsler}@uni-bremen.de,mathias.soeken@epfl.ch

ABSTRACT

Approximate computing is an emerging methodology that allows to increase efficiency in a range of resilient applications for an affordable loss of precision or quality. In this paper, we exploit approximation in a multi-criteria optimization approach for the widely used data structure *Binary Decision Diagram* (BDD) to achieve higher efficiency besides lowering the inaccuracy. For this purpose, we utilize an ε-preferred evolutionary algorithm giving a higher priority to minimize BDD sizes as well as maintaining certain error constraints. In particular, we propose an adaptive ε-setting method which adds an automated factor to the algorithm based on the behavior of the function under approximation. This improves the performances of the algorithm by correcting the effect of the user set error constraints which can restrict the dimensions of the search and can lead to immature convergence.

In comparison with the non-optimized BDDs, the proposed algorithm achieves a high gain of 68.02% at a low cost of 2.12% inaccuracy for the whole benchmark set. The experimental results also reveal a considerable improvement of 25.19% in the average value of error rate besides reduction in BDD sizes compared to the manual ε-setting approach.

CCS CONCEPTS

•Mathematics of computing →Evolutionary algorithms;
•Applied computing →Computer-aided design;

KEYWORDS

Approximate BDD optimization; prioritized evolutionary algorithm; adaptive algorithm

ACM Reference format:
Saeideh Shirinzadeh, Mathias Soeken, Daniel Große, and Rolf Drechsler. 2017. An Adaptive Prioritized ε-Preferred Evolutionary Algorithm for Approximate BDD Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference 2017, Berlin, Germany, July 15–19, 2017 (GECCO '17),* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071281

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071281

1 INTRODUCTION

Approximate computing has gained high attention due to allowing to lower costs by causing a loss of quality in resilient applications. A wide variety of modern applications such as media processing, recognition, and data mining tolerate acceptable error rates that can be exploited by approximate computing circuits and systems to increase efficiency in time or energy [5], [14].

Approximate synthesis of multi-level logic circuits was proposed in [20] to minimize area of the resulting circuits for a given error threshold. Another approach has been proposed in [26] that introduces a systematic methodology to formulate the problem of approximate computing and mapping it into an equivalent traditional synthesis problem. *Binary Decision Diagrams* (BDD) is a graph based data structure for efficient representation of Boolean functions, which can be also exploited for approximate computing in the area of electronic design automation. BDDs are canonical w.r.t. a variable ordering, which determines the number of nodes in BDDs, the so-called BDD size [11]. BDD size is the main cost metric for most of the applications utilizing BDDs such as logic synthesis and formal verification [4]. Nonetheless, a BDD can easily explode in size when the number of input variables of the manipulated Boolean function increases.

Improving the variable ordering of a BDD is known to be NP-complete [2]. So far, many BDD optimization methods, using either exact or heuristic approaches, have been proposed to find efficient BDD representations [11]. Using approximation provides another alternative to variable reordering to lower the number of nodes at a cost of inaccuracy. An approach for functional approximation of BDDs was proposed in [23], which eliminates a number of nodes by applying some operators to manually selected BDD levels denoted by variable indices. A similar BDD approximation methodology was incorporated with the classical variable reordering in [22] to attain BDDs with smaller sizes and lower error overheads. For this purpose, a prioritized ε-preferred evolutionary algorithm was suggested in [22] as an appropriate approach to satisfy size minimization and error constraints.

Although the approach presented in [22] addresses the main aspects of approximate BDD optimization, it's entirely manual nature of applying error constraints results in immature convergence to higher deviation from the exact BDD. In this work, we propose an adaptive ε-selection method to prevent this drawback. The proposed approach adds an automated element, which is adopted based on the behavior of the function under approximation, to the user defined

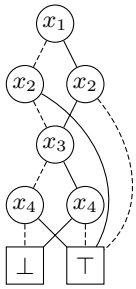

Figure 1: BDD for the function $(x_1 \oplus x_2) \vee (x_3 \oplus x_4)$

error constraint required by the evolutionary algorithm. This compensates the lack of sufficient knowledge of the user in setting error constraints which can dramatically limit the search directions and lead to lower quality results. Using the proposed adaptive algorithm improves the BDD sizes and especially lowers the error overheads which is also confirmed by the experiments showing a reduction of 25.19% in the error rate compared to the manual approach.

The remainder of this paper is organized as follows. In Section 2 we present the preliminaries on BDD approximation as well as discussing the existing BDD approaches for BDD minimization briefly. Section 3 explains the requirements of the BDD approximation problem. In Section 4, we present our proposed evolutionary approach. Experimental results are presented in Section 5 and Section 6 concludes the paper.

2 BACKGROUND

2.1 Preliminaries

2.1.1 Binary Decision Diagrams. A BDD (e.g., [10]) is a graph-based representation of a function that is based on the Shannon decomposition $f = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i}$, where f_{x_i} and $f_{\bar{x}_i}$ designate function f when the variable x_i is equal to 1 or 0, respectively. Applying this decomposition recursively allows dividing the function into many smaller sub-functions. Solid and dashed lines refer to high and low successors, respectively (see Fig. 1). BDDs make use of the fact that for many functions of practical interest, smaller sub-functions occur repeatedly and need to be represented only once. Combined with an efficient recursive algorithm that makes use of caching techniques and hash tables to implement elementary operations, BDDs are a powerful data structure for many practical applications. BDDs are ordered in the sense that the Shannon decomposition is applied w.r.t. some given variable ordering which also has an effect on the BDD's number of nodes. Improving the variable ordering for BDDs is NP-complete [2] and many heuristics have been presented that aim at finding a good ordering. Throughout this paper, we consider the initial BDDs, i.e. the input BDDs before being approximated and optimized, with a fixed variable ordering and we assume that this is the natural one $x_1 < x_2 < \cdots < x_n$.

Given a Boolean function $f(x) = f(x_1, \ldots, x_n)$, $|f|$ refers to the number of binary vectors $x = x_1 \ldots x_n$ such that $f(x) = 1$ (ON-set). We define $B(f)$ to be the size of the BDD representation for f and a given variable ordering, which is the total number of nodes, including the sinks.

2.1.2 BDD approximation. BDD approximation aims at minimizing a BDD representing a given Boolean function f by approximating it with a Boolean function $\hat{f}$ and respecting a given threshold based on an error metric. The associated exact decision problem is called *Approximate BDD Minimization* (ABM, [23]) and tries to find an approximated function $\hat{f}$ for a given function f, under constraints including an error metric e, a threshold t, and a size bound b, to satisfy the two following statements $e(f, \hat{f}) \leq t$ and $B(\hat{f}) \leq b$.

In this paper, we use worst-case error and error rate as error metrics e. The *worst-case error*

$$\mathrm{wc}(f, \hat{f}) = \max_{x \in \mathbb{B}^n} |\operatorname{int}(f(x)) - \operatorname{int}(\hat{f}(x))| \tag{1}$$

where 'int' represents the integer value of a bit vector, is the maximum difference between the value of the approximated output and the corresponding exact version for all input combinations. The *error rate*

$$\mathrm{er}(f, \hat{f}) = \frac{\sum_{x \in \mathbb{B}^n} [f(x) \neq \hat{f}(x)]}{2^n} \tag{2}$$

is the ratio of the number of inequalities between the approximated output and the initial function to the total number of input assignments. The product of error rate and the total number of input assignments is equal to the Hamming distance when applied to single-output functions. The first metric considers the integer representation of the function values instead of the binary representation. This is useful since in most applications a change in more significant bits has a much higher effect than changes in less significant bits.

In order to approximate a BDD we use the operators which can efficiently be implemented using the APPLY algorithm [18]. These operators should (i) change the function and (ii) reduce the size of the BDD. The functional change should at best not be too drastically, i.e., the effect on the error metric is kept small. In this paper, we use co-factoring on some variable x_i, i.e.,

$$\hat{f} \leftarrow f_{x_i} \quad \text{or} \quad \hat{f} \leftarrow f_{\bar{x}_i}, \tag{3}$$

as the simplest operator, however, with quite a significant effect on the error metric. A more complex algorithm is called *heavy branch subsetting* [17] which we apply partially to the level of x_i and all lower levels (towards the terminals), denoted $\lfloor f \rfloor_{x_i}$ and $\lceil f \rceil_{x_i}$ and called *rounding up* and *rounding down*. For each node that appears at level x_i or lower (in other words for each node labeled x_j with $j \geq i$), the lighter child, i.e., the child with the smaller ON-set, is replaced by a terminal node. The terminal node is $\bot$ when rounding down, and $\top$ when rounding up.

2.2 Related work

In many VLSI CAD applications BDDs are directly mapped to target circuits such that each BDD node is simply replaced by a multiplexer in the real implementation [11]. Hence, the smaller size of BDDs means the smaller chip area which makes number of nodes the main cost metric. There has been few approaches presenting BDD optimization techniques w.r.t. different criteria such as the number of paths [13,

15], expected or average path length [11]. In [21], a multi-objective BDD optimization method was proposed to reduce both of the number of nodes and paths in BDDs. Nonetheless, the majority of approaches presented so far optimize BDDs w.r.t. the number of nodes.

There are several methods that use exhaustive optimization techniques to reduce the number of nodes in BDDs [11, 12, 16]. These exact methods guarantee to find the optimum BDD, but their complexity is a serious drawback which makes the use of heuristic approaches more practical. Sifting [18] is a well-known heuristic for BDD node minimization which algorithm is based on a hill-climbing framework. Sifting swaps two adjacent variables in a BDD without changing the function. The variables are moved upward and downward in the variable ordering representing the BDD. The resulting BDD sizes are recorded during the variable movements. At the end of this procedure each variable is moved back and fixed at the position where the BDD with the minimum number of nodes was found. Sifting is recognized as a fast and useful technique, but it fails to find or even approach the optimum BDD when the number of input variables increases. Other approaches such as simulated annealing [1] and genetic algorithms [9] have been also proposed for BDD minimization, which can achieve BDD sizes up to half of that found by sifting.

As approximation is becoming an emerging computing paradigm for exploiting the intrinsic error resilience in many of modern applications, the need for approximate computing design methods increases. In [23], this requirement was addressed for applications using BDDs by introducing a minimization approach for approximate computing. The paper proposes approximate operators for eliminating a number of nodes in BDDs as well as several algorithms to compute different error metrics.

In [22], the fundamentals of BDD approximation proposed in [23] were incorporated with variable ordering to increase gain by simultaneously optimizing and approximating BDDs. The approach uses a *prioritized ε-preferred evolutionary algorithm* to search for BDDs represented by variable permutation and pairs them with their corresponding approximation operators which lead to smaller sizes and deviations from the exact function. A higher priority is given to BDD sizes to ensure that the minimum approximated BDD is attainable under certain error constraints. At the same time, setting ε values below the constraints allows to choose BDDs with lower error values in case of equal sizes.

3 ALGORITHM REQUIREMENTS

In this section we briefly review the requirements of an algorithm for the approximate BDD optimization problem.

The most simple model for approximate BDD optimization considering the two error metrics described in Section 2.1.2 is a three-objective problem with the first objective set to the BDD size. This model might need to avoid occurrence of non-comparable solutions by adding density information due to the presence of three objectives. The model is also not a true projection of the problem because of ignoring the main purpose of approximate BDD computing which is finding the BDD with the minimum size. Moreover, it does not guarantee to maintain the validity of the approximated BDDs.

By incorporation of approximation and optimization for BDDs, we accept to pay some costs as errors to find smaller BDDs which might not exist using only variable ordering techniques. Thus, the error metrics can not be considered as important as the number of BDD nodes. This means that the optimization algorithm should be capable of handling higher priority of the first criterion. A more comprehensive priority scheme that allows different priority scenarios for all objectives might be also of interest for some Boolean functions when one of the error metrics highly affects the precision and validity of approximation. For example, for a BDD representing an adder a high value of error-rate can be tolerated if the integer values of the real and the approximated functions are slightly different even for a large number of assignments. While a high value of worst-case error can mean a failure since it shows that the output of the approximated function is unacceptable for at least one input assignment.

Besides the discussion above on the priority of objectives, the algorithm should also guarantee that the final solutions represent valid BDDs according to the user defined standards. Hence, for each error metric a threshold value should be set. It is worth noting that it is sufficient for the algorithm to satisfy the error thresholds. In fact, we ensure not to lose the minimum BDD size at a cost of finding error values smaller than required.

Both of the aspects mentioned above have been addressed in [22] by utilizing the *prioritized ε-preferred* which maintains the higher significance of BDD size by a priority scheme, and ensures the fulfillment of the error constraints by setting threshold values besides the metric ε. The latter contributes on guiding the search towards lower error values if several candidates with the minimum size coexist. Nevertheless, an important factor has been taken into account by the algorithm used in [22].

The values of errors caused by approximation highly depend on the characteristics of the functions represented by BDDs. It is almost impossible for a user to anticipate the behavior of a function under approximation [23]. For some functions, a small operation on the BDD under approximation can cause dramatic inaccuracies without a noticeable improvement in the size, whereas high gain can be achievable at low error costs for other functions. In such a case, approximate BDD optimization may not lead to the best possible error values. If the error constraints are set too loose, the process will terminate without finding the best possible error values. On the other hand, strict constraints may allow only few individuals in the population which cover a small percentage of the whole search space. This can limit the of directions of evolution and prevent to obtain minimal BDDs. Therefore, a proper adaptive error constraint scheme depending on the behavior of the function under approximation can be highly effective on the quality of results.

In this work, we use the same evolutionary algorithm as proposed in [22] but enhance it with an adaptive ε setting method. This alleviates the concerns about error constraints by allowing to loose the manually set threshold values and move the weight of the constraints to the automated ε setting.

4 ALGORITHM FEATURES

In this section, we first explain the prioritized ε-preferred relation with adaptive ε which we employed for comparison of solutions. Then, we present the overview of the proposed multi-objective algorithm for approximate BDD optimization.

4.1 Prioritized ε-preferred

In general, an arbitrary multi-objective problem with m objectives is defined as

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

solution $\mathbf{x} = (x_1,\ x_2,\ \ldots,\ x_n)^{\mathrm{T}}$ in the equation above is 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 solution by mapping it to objective space $\Lambda \subseteq \mathbb{R}^m$. Assuming two arbitrary solutions x, $y \in \Omega$, the Pareto-dominance can be defined as

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

As discussed before, the strict relation of Pareto-dominance is not required for the approximate BDD optimization. In this case, we are only interested in finding BDDs with smaller sizes, and the error metrics does not matter until they satisfy the threshold values. To find the appropriate ranking method for this purpose, we introduce the so-called relation *preferred* proposed in [7]. Preferred is a refinement of Pareto-dominance which respects the number of objective functions for discriminating solutions. For i, $j \in \{1, 2, \ldots, m\}$, relation preferred is defined as

$$\mathbf{x} \prec_{\text{preferred}} \mathbf{y} :\Leftrightarrow \\ |\{i : f_i(\mathbf{x}) < f_i(\mathbf{y})\}| > |\{j : f_j(\mathbf{y}) < f_j(\mathbf{x})\}|.$$

More informally, the equation above means that solution x is preferred to y if the number of objective functions of x which are smaller than their corresponding values of y is larger than the number of objective functions of y smaller than the same components of x.

Relation preferred is not transitive and as a result it can not be a partial order. Consequently, concepts such as non-dominating levels represented for Pareto-dominance [6], where each solution in a level dominates all solutions belonging to the levels with higher fitness values can not be defined for preferred. For a population compared based on preferred, the solutions can be pairwise comparable [7]. To make this more clear we give an example. Let $f(x) = (8, 7, 1)$, $f(y) = (1, 9, 6)$, and $f(z) = (7, 0, 9)$ be solutions mapped into the objective space. According to relation preferred, x is preferred to y and y is preferred to z. For a transitive relation it could be concluded that x has the same relation with z, while here solution z is preferred to x. Indeed, the solutions x, y, and z describe a cycle in comparison with each other.

Solutions inside a cycle are denoted as incomparable and are ranked equally. To sort solutions based on preferred, the population is divided into the so-called *Satisfiability Classes* (SCs, [7]). Each SC represents the fitness value for a set of solutions that are not comparable according to the concept explained above. Solutions of SCs indicated by smaller values are preferred to solutions of SCs with larger indices during selection. However, due to the non-transitivity of relation preferred all solutions belonging to an SC are not necessarily preferred to the entire solutions with larger SC indices. For example, two solutions x and y in two successive SCs might be incomparable but x is classified in a better SC since it has been also incomparable with another solution z which is preferred to y.

In [25], relation *ε-preferred* was proposed to make preferred more robust for many objective optimization problems. ε-preferred adds the parameter ε to enhance the quality of final solutions by specifying limits for each dimension. Comparing two objective vectors $f(x) = (1, 1, 100)$ and $f(y) = (5, 5, 5)$ based on preferred, solution x is ranked better while it might not be recognized efficient by the user for the high value of 100 in one dimension. This problem is addressed by setting maximum values ε_i, $i \in \{1, \ldots, m\}$ for each optimization objective.

Using ε-preferred, we first need to compare two given solutions x, $y \in \Omega$ based on the number of times that each solution violates ε. Assuming i, $j \in \{1, \ldots, m\}$, a relation *ε-exceed* is defined as

$$\mathbf{x} \prec_{\varepsilon\text{-exceed}} \mathbf{y} :\Leftrightarrow |\{i : f_i(\mathbf{x}) < f_i(\mathbf{y}) \wedge |f_i(\mathbf{x}) - f_i(\mathbf{y})| > \varepsilon_i\}| \\ > |\{j : f_j(\mathbf{x}) > f_j(\mathbf{y}) \wedge |f_j(\mathbf{x}) - f_j(\mathbf{y})| > \varepsilon_j\}|.$$

According to the equation above solution x ε-exceeds y if the number of times that x exceeds ε in any dimension is smaller than the same count for y. ε-preferred means the same as ε-exceed if the solutions have different ε-exceeding counts, otherwise preferred is required to discriminate between solutions. Using ε-exceed the complete definition of ε-preferred is formally expressed by

$$\mathbf{x} \prec_{\varepsilon\text{-preferred}} \mathbf{y} :\Leftrightarrow \\ \mathbf{x} \prec_{\varepsilon\text{-exceed}} \mathbf{y} \vee (\mathbf{y} \not\prec_{\varepsilon\text{-exceed}} \mathbf{x} \wedge \mathbf{x} \prec_{\text{preferred}} \mathbf{y}).$$

As discussed in Section 3 the main goal of BDD approximation is size minimization under certain error constraints, which makes priority an undeniable requirement. Considering objective priorities with relation preferred was introduced in [19]. The model incorporates a lexicographical ordering of objectives w.r.t. a user defined priority vector. To fulfill the requirements of the approximate BDD optimization problem we use the prioritized ε-preferred relation proposed in [8]. Prioritized ε-preferred denoted by $\prec_{\text{prio-}\varepsilon\text{-pref}}$ exploits the same model used in [19] and is able to handle different levels of priorities for all of the optimization objectives.

Let $p = \{p_1, p_2, \ldots, p_m\}$ be a priority vector determining priorities assigned to an m-objective problem. Each component p_i, $i \in \{1, 2, \ldots, m\}$ can adopt values from the set $\{1, 2, \ldots, n\}$, $n \leq m$, where 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 x, $y \in \Omega$, $x|_j$ and $y|_j$ represent subvectors of x and y only including

objective functions with priority of j, prioritized ε-preferred is defined as

$$\mathbf{x} \prec_{\text{prio-}\varepsilon\text{-pref}} \mathbf{y} :\Leftrightarrow \exists j \in \{1, 2, \ldots, n\} : \mathbf{x}|_j \prec_{\varepsilon\text{-preferred}} \mathbf{y}|_j \wedge \forall k < j : \mathbf{y}|_k \not\prec_{\varepsilon\text{-preferred}} \mathbf{x}|_k.$$

The relation defined above employs ε-preferred to compare subvectors of objective functions with equal priorities. x is prioritized ε-preferred to y if there is a subvector of objective functions in x with priority value j that is ε-preferred to the corresponding subvector in y, and at the same time $x|_j$ is not ε-preferred by any subvector of priority higher than j in y.

4.2 Adaptive ε

To ensure removal of invalid solutions, we should make sure that such solutions are ranked at the end of population and are not survived for the next generations. This approach fails to sort the population properly in the presence of a large number of invalid solutions which occurs when the error thresholds are low for the function. An ε-scheme, which is set based on the existing values of objective functions, can be highly effective for the hard task of threshold setting in the problem of approximate BDD optimization. Indeed, error estimation requires high expertise. For many Boolean functions, there might not be any solution satisfying a small value of threshold, and on the other hand for a high threshold the search might end up missing solutions with low error values. Using an adaptive ε makes it possible to set large threshold values to ensure that the algorithm will not be stuck in finding incomparable invalid solutions, and at the same time, the search is directed toward desired error values considering limits for each error metric.

It is worth noting that the value of ε does not matter for the prioritized optimization objective, i.e., the BDD size. As explained before, prioritized ε-preferred performs comparisons based on relation ε-preferred in subvectors of objectives function with equal priorities starting from the most significant criterion. It is clear that comparisons of solutions for a single objective function $\prec_{\varepsilon\text{-preferred}}$ works the same as Pareto-dominance. In fact, $x|_1 \prec_{\varepsilon\text{-preferred}} y|_1$ here can only mean that the BDD size of solution x is smaller than the same metric in y which fulfills the problem requirement of size minimization. Hence, we can simply set ε_1 to any value such as zero.

ε values can also be set manually to a fraction of corresponding thresholds for both of the error metrics as in [22]. However, an automatic adaptive ε selection scheme is more influential on the performance of the algorithm as explained before. In [8], it was proposed to assign the the averages of objective functions as the ε values for each optimization criterion. Here, we suggest an adaptive method based on the standard normal distribution of those error values of the parent population which satisfy the thresholds. The ε_i, $i \in \{2, 3\}$ at each generation is equal to $N(\mu_i, \sigma i^2)$, where μ_i and σ_i are respectively the mean value and standard deviation of the set of i^{th} error values.

Using the proposed adaptive ε selection method, in the primary generations ε values are larger with a high probability since the objective functions are spread widely over the objective space. As the number of generations increases the ε values decrease due to the fact that the population gets more evolved which leads to the concentration of objective functions on smaller values. This way a higher diversity is provided in the early generations, while convergence tends to be more greedy in the final generations.

The average based ε proposed in [8] decreases steadily as the population evolves and the objective functions become smaller. However, this statement is also true for the proposed adaptive ε selection with a high probability, there is still a poor possibility of too high or low ε values far different from the average of population resulted by the normal distribution. This can cause preferring solutions with worse objective functions during some comparisons. This technique can be useful for allowing individuals dissimilar to the majority of solutions. It increases the diversity of the population, and thus can also improve the convergence of the results due to adding new search directions.

4.3 Evolutionary cycle and operators

The proposed approach minimizes BDDs by performing both variable reordering and approximation. For this purpose, we use an elitist genetic algorithm based on relation prioritized ε-preferred to find the smallest approximated BDDs with valid error values represented by a variable ordering and an approximation vector. Hence, each individual inside the population has two parts which are initialized independently. One part designates the exact BDD by a permutation of input variables of the Boolean function and the other is a vector consisting of pairs designating the approximation operators and the BDD level indices where each operator should be applied. The length of the approximation vector is determined by the maximum allowed number of approximation operators which is defined as an option for initializing the population. The approximation vector in each initial individual might contain a randomly generated number of approximation operators from one to this maximum value.

After initialization, the population is evaluated. First the exact BDDs are created according to the variable orderings, and then the approximation vectors are applied to the corresponding BDDs to create the approximated ones. Thereafter, the objective functions, i.e., the size and error values of the approximated BDDs, are set to each individual of the population. Then, the relation prioritized ε-preferred is used to sort the population into satisfiability classes. We use binary tournament for mating selection and then apply the variation operators to make an offspring of the same size as the parent population. The union of both parent and offspring populations are then sorted and the best individuals are survived as the next generation according to their fitness values. This cycle is continued until a certain number of iterations.

As mentioned before, the fitness value for each solution is equal to the index of the satisfiability class to which the solution belongs to. In fact, the density information of population is not considered in selection. By giving higher priority to one of the objectives the search is focused on the best ever found value of the prioritized criterion instead of finding a good distribution of solutions not preferred to each other.

It is obvious that there should be two types of variation operators for the distinct parts of each individual, i.e., both cross over and mutation operators should be specifically defined for the permutations of variable orderings and the approximation vectors to prevent invalid solutions. For crossover of the variable orderings we use *Partially Matched Crossover* [15] which guarantees valid permutations. The crossover operator for the approximation vectors selects two positions randomly and breaks the parents into three sections. Then two children are produced by combination of sections from both parents.

We utilize the same mutation operators used in [21] for the mutation of variable orderings. Three mutation operators are defined differently for the approximation vectors. The first operator selects one position in the vector randomly and increments or decrements its content by equal probabilities. The validity of the solutions is also maintained by only allowing changes which keep the indices denoting either approximation operators or BDD levels within their acceptable ranges. The second mutation operator applies the first operator on the same approximation vector twice. The third operator chooses one pair of approximation operator and level index and increments or decrements both with equal probabilities.

5 EXPERIMENTAL RESULTS

5.1 Experimental setup

We have assessed the performance of our proposed algorithm on a set of multiple-output benchmark set including 20 Boolean functions. The benchmarks are from ISCAS89 [3] with a range of input variables from 13 to 54, and primary outputs from 9 to 52. For each benchmark, the population size is set to three times the number of input variables but not larger than 120. The algorithm terminates after 500 generations and the results shown in Table 1 represent the best out of ten independent runs for each benchmark function. The sum of the probabilities of crossover operators for both variable ordering and approximation vectors are set to 1. A probability of $1/n$, where n is the number of input variables of the Boolean function, i.e., the length of the ordering vectors, is equally distributed over the three mutation operators.

The maximum number of times that approximation operators are applied to any solution in the population is set to the 3 during all experiments. This gives a fixed length of 6 to all approximate vectors since they are formed of pairs of indices denoting approximation operators and their corresponding BDD levels. Thus, the overall probability for the three mutation operators designed for approximation vectors is set to 1/6. For BDD representation of the benchmark functions and for the implementation of the approximation operators we have used the CUDD package [24].

5.2 Performance evaluation and comparison

Table 1 shows the results obtained by the proposed adaptive ε-preferred evolutionary algorithm. The results are also compared with non-optimized initially ordered BDDs indicated by $\#N$-initial, BDDs optimized by sifting reordering technique [18], and the results obtained by the proposed algorithm when ε is set manually, to see the effect of the adaptive scheme.

The threshold values for the error rate is set to 10%, i.e., each output of the approximated BDD can be different from the corresponding output in the exact BDD for a maximum of 10% of the whole input assignments. We have used the same value of 10% as the threshold for the worst case error denoted by WC. According to the definition of the worst case error, such a threshold means that the maximum difference between the equivalent integer values of the output vectors of the approximated BDD and the exact one can not exceed 2^{m-2} for a Boolean function with m outputs, which is equal to the 10% of the largest possible value that the exact function can adopt. The default manual values of ε for each error metric are set to 5%, and as discussed before the adaptive ε at each generation is selected from the normal distribution of the valid solutions with error values under thresholds.

It should be noted that the results in Table 1 with absolute 0 error values, show that no approximation have been performed. Actually, the approximated BDD representations satisfying the defined thresholds have not been reachable for one or both of the error metrics. Therefore, the algorithm converges to optimized BDDs represented by the best found variable orderings and empty approximation vectors.

The experimental evaluations show a noticeable size reduction at a small cost of error for both manual and adaptive versions of the proposed algorithm. More precisely, the adaptive ε approach obtains a size improvement of 68.02% on average in comparison with the initial BDDs. This improvement has been achieved at a low cost of total inaccuracy, i.e. the average of both error metrics, equal to 2.12%, which is insignificant compared to the amount of size reduction.

Comparison of the results with the sifting [18] reordering technique also shows that the proposed algorithm using both manual and adaptive ε lowers the number of BDD nodes considerably. The average number of BDD nodes over the entire benchmark set by the proposed adaptive ε-preferred algorithm is reduced by 23.51% compared to the same value obtained by sifting [18]. The gain achieved in the BDD sizes is more than 10 times the total inaccuracy of 2.12%, which proves that incorporating approximation with BDD optimization, i.e. variable reordering, has been highly effective.

According to Table 1, the algorithm using adaptive ε outperforms the manual approach in both BDD sizes and inaccuracy. The difference between the average size improvement achieved by the manual and adaptive ε methods is very small. However, the average error rate by the adaptive approach is reduced by 25.19% compared to the algorithm using the manual ε. This confirms that the adaptive ε approach has been successful in guiding the search towards minimum BDDs with smaller error values.

5.3 Assessment of ε-setting methods

To see the effect of different ε schemes on the quality of results, we have performed experiments on a set of six selected benchmark functions by setting the threshold values for both error rate and the worst case error at 0%, 5%, 10%, 25%, 50%, 75% and 95%. The default manual ε values are set to half of the thresholds for both error metrics and each benchmark

Table 1: Experimental Evaluation and comparison with initially ordered and reordered BDDs by Sifting [18]

Benchmark	$\#I/O$	$\#N$-initial	$\#N$-Sifting	Manual ε				Proposed adaptive ε			
			[18]	$\#N$	$ER(\%)$	$WC(\%)$	impr.(%)	$\#N$	$ER(\%)$	$WC(\%)$	impr.(%)
s208	18/9	1033	61	30	9.32	0.19	97.09	**29**	7.76	0.19	97.19
s298	17/20	125	78	**72**	0.78	3.12	42.40	73	0.78	3.12	41.60
s344	24/26	206	104	106	6.25	3.12	48.54	**103**	6.25	0.39	50.00
s349	24/26	206	**104**	105	0.00	0.00	49.02	**104**	0.00	0.00	49.51
s382	24/27	168	121	121	0.00	0.00	27.97	**119**	0.00	0.00	29.16
s386	13/13	281	123	114	0.39	0.19	59.43	**110**	0.39	0.19	60.85
s400	24/27	168	**121**	122	2.25	0.19	27.38	126	0.00	0.00	25.00
s420	34/17	262227	185	**63**	6.24	<0.001	99.97	**63**	6.24	<0.001	99.97
s444	24/27	226	161	126	0.00	0.00	44.24	**122**	0.00	0.00	46.01
s510	25/13	19076	165	147	0.00	0.00	99.22	**146**	0.00	0.00	99.23
s526	24/27	232	141	**116**	0.00	0.00	50.00	119	0.00	0.00	48.70
s641	54/42	1352	629	408	8.98	3.12	69.82	**405**	2.72	6.83	70.04
s713	54/42	1352	629	403	2.72	6.83	70.19	**387**	2.72	6.83	71.37
s820	23/24	2651	258	221	0.09	9.76	91.66	**219**	0.09	9.76	91.73
s832	23/24	2651	258	**218**	0.09	9.37	91.77	**218**	0.09	9.37	91.77
s953	45/52	1723	402	**102**	0.38	5.42	94.08	106	0.19	5.42	93.84
s967	45/52	1755	417	264	9.84	3.61	84.95	**259**	9.78	3.61	85.24
s1196	32/32	2295	642	627	2.59	<0.001	72.67	**617**	<0.001	<0.001	73.11
s1238	32/32	2295	642	627	2.59	<0.001	72.67	**600**	2.59	<0.001	73.85
s1488	14/25	1016	391	**375**	0.00	0.00	63.09	383	0.00	0.00	62.30
AVG		15051.90	281.60	218.35	2.62	**2.24**	67.80	**215.40**	**1.96**	2.28	**68.02**

$\#I$: number of inputs, $\#O$: number of outputs, $\#N$-initial: non-approximated BDD size, $\#N$-sifting: BDD size obtained by sifting reordering technique [18], $\#N$: BDD size after approximation, ER: error rate, WC: worst case error, improvement is calculated compared to the initially ordered non-approximated BDD

has been run 5 times for each given threshold value for the same number of 500 generations. We have also compared the performance of our algorithm with the adaptive ε scheme proposed in [8], which is the average value of each objective function over the entire parent population at each generation.

Figure 2 shows the BDD sizes obtained by the three different ε-selection methods at the given threshold values. We have removed duplication of data points representing BDDs with equal sizes which is the reason for having less than seven measurements in a few plots. It is obvious that the BDDs representing different functions slope down to smaller sizes with different paces. However, all of the plots show that the BDD sizes decrease as the threshold values increase as expected, which is the main purpose of approximation.

A quick look at Figure 2 reveals that smaller BDD sizes have been obtained by the proposed adaptive ε in comparison with the manual ε and average ε [8] approaches. The results by the proposed ε scheme show smaller BDD sizes compared to the two other ε selection methods for all of the functions, except s1196 at threshold 50%. These results confirm that the proposed adaptive ε approach is also influential for the convergence of the BDD sizes as well as the error values.

6 CONCLUSION

Approximate computing is an emerging paradigm that can be exploited in a variety of applications which can tolerate some levels of inaccuracy. In this paper we proposed an approach for approximate optimization of BDDs which are widely used in formal verification and VLSI CAD. We proposed an adaptive ε-preferred evolutionary algorithm for a three-objective optimization problem. The first objective, i.e., the number of nodes of BDDs, was set to a higher priority which projects the main purpose of the BDD approximation. The two other objectives are error metrics measuring the inaccuracy caused by approximation. For given error thresholds, the algorithm is designed to find the BDD with the minimum possible number of nodes while by the adaptive ε schemes we ensure to prefer lower error values for equal BDD sizes. The experimental results show that the proposed approach has accomplished a considerable size reduction of BDDs for small error values.

ACKNOWLEDGMENTS

This research was supported by the University of Bremen's graduate school SyDe funded by the German Excellence Initiative, by the German Research Foundation within the projects MANIAC (DFG) (DR 287/29-1), and Reinhart Koselleck (DFG) (DR 287/23-1), and the subproject P01 'Predictive function' of the Collaborative Research Center SFB1232, by the Swiss National Science Foundation within project MAJesty (SNF) (200021 169084), and by H2020-ERC-2014-ADG 669354 CyberCare.

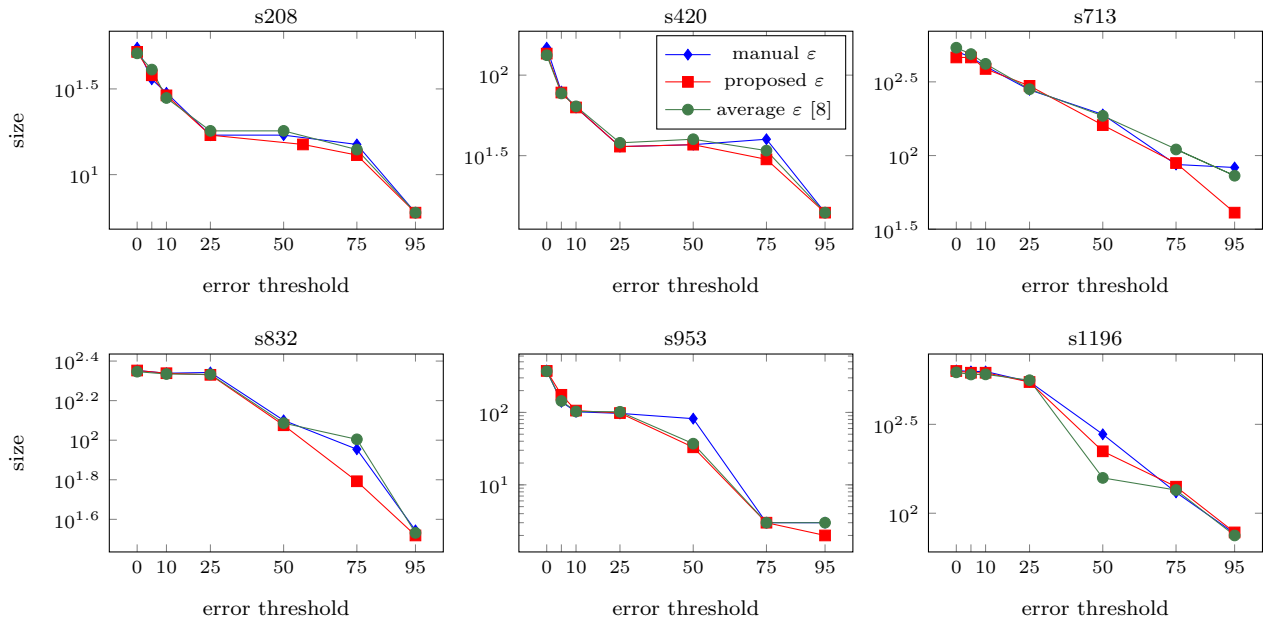

Figure 2: Comparison of different ε selection approaches

REFERENCES

[1] Beate Bollig, Martin Löbbing, and Ingo Wegener. 1995. Simulated Annealing To Improve Variable Orderings For OBDDs. In *International Workshop on Logic Synth.*

[2] Beate Bollig and Ingo Wegener. 1996. Improving the Variable Ordering of OBDDs Is NP-Complete. *IEEE Trans. Comput.* 45, 9 (1996), 993–1002.

[3] Franc Brglez, David Bryan, and Krzysztof Kozminski. 1989. Combinational Profiles of Sequential Benchmark Circuits. In *International Symposium on Circuits and Systems*. 1929–1934.

[4] Randal E. Bryant. 1995. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. In *International Conference on Computer-Aided Design*. 236–243.

[5] Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Analysis and Characterization of Inherent Application Resilience for Approximate Computing. In *Design Automation Conference*. 113:1–113:9.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[7] Nicole Drechsler, Rolf Drechsler, and Bernd Becker. 2001. Multi-objective Optimisation Based on Relation *Favour*. In *International Conference on Evolutionary Multi-Criterion Optimization*, Vol. 1993. 154–166.

[8] Nicole Drechsler, André Sülflow, and Rolf Drechsler. 2015. Incorporating User Preferences in Many-Objective Optimization Using Relation ε-Preferred. *Natural Computing* 14, 3 (2015), 469–483.

[9] Rolf Drechsler, Bernd Becker, and Nicol Göckel. 1996. A Genetic Algorithm for Variable Ordering of OBDDs. In *IEE Proceedings of Computers and Digital Techniques*, Vol. 143(6). 364–368.

[10] Rolf Drechsler and Detlef Sieling. 2001. Binary decision diagrams in theory and practice. *Software Tools for Technology Transfer* 3, 2 (2001), 112–136.

[11] Rüdiger Ebendt, Görschwin Fey, and Rolf Drechsler. 2005. *Advanced BDD Optimization*. Springer.

[12] Rüdiger Ebendt, Wolfgang Günther, and Rolf Drechsler. 2003. An Improved Branch and Bound Algorithm for Exact BDD Minimization. *IEEE Transactions on CAD of Integrated Circuits and Systems* 22, 12 (2003), 1657–1663.

[13] Görschwin Fey and Rolf Drechsler. 2006. Minimizing the Number of Paths in BDDs: Theory and Algorithm. *IEEE Transactions on CAD of Integrated Circuits and Systems* 25, 1 (2006), 4–11.

[14] Jie Han and Michael Orshansky. 2013. Approximate Computing: An Emerging Paradigm for Energy-Efficient Design. In *IEEE European Test Symposium*. 1–6.

[15] Mario Hilgemeier, Nicole Drechsler, and Rolf Drechsler. 2003. Minimizing the Number of One-Paths in BDDs by an Evolutionary Algorithm. In *IEEE Congress on Evolutionary Computation*. 1724–1731.

[16] Nagisa Ishiura, Hiroshi Sawada, and Shuzo Yajima. 1991. Minimazation of Binary Decision Diagrams Based on Exchanges of Variables. In *International Conference on Computer-Aided Design*. 472–475.

[17] Kavita Ravi and Fabio Somenzi. 1995. High-density Reachability Analysis. In *International Conference on Computer-Aided Design*. 154–158.

[18] Richard Rudell. 1993. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *International Conference on Computer-Aided Design*. 42–47.

[19] Frank Schmiedle, Nicole Drechsler, and Rolf Drechsler. 2001. Priorities in Multi-Objective Optimization for Genetic Programming. In *Genetic and Evolutionary Computation Conference*. 129–136.

[20] Doochul Shin and Sandeep K. Gupta. 2010. Approximate Logic Synthesis for Error Tolerant Applications. In *Design, Automation and Test in Europe*. 957–960.

[21] Saeideh Shirinzadeh, Mathias Soeken, and Rolf Drechsler. 2015. Multi-Objective BDD Optimization with Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference*. 751–758.

[22] Saeideh Shirinzadeh, Mathias Soeken, Daniel Große, and Rolf Drechsler. 2016. Approximate BDD Optimization with Prioritized ε-Preferred Evolutionary Algorithm. In *Genetic and Evolutionary Computation Conference*. 79–80.

[23] Mathias Soeken, Daniel Große, Arun Chandrasekharan, and Rolf Drechsler. 2016. BDD Minimization for Approximate Computing. In *Asia and South Pacific Design Automation Conference*. 474–479.

[24] Fabio Somenzi. 2012. CUDD: CU Decision Diagram Package Release 2.5.0. University of Colorado at Boulder.

[25] André Sülflow, Nicole Drechsler, and Rolf Drechsler. 2007. Robust Multi-Objective Optimization in High Dimensional Spaces. In *International Conference on Evolutionary Multi-Criterion Optimization*. 715–726.

[26] Swagath Venkataramani, Amit Sabne, Vivek J. Kozhikkottu, Kaushik Roy, and Anand Raghunathan. 2012. SALSA: Systematic Logic Synthesis of Approximate Circuits. In *Design Automation Conference*. 796–801.