Error Bounded Exact BDD Minimization in Approximate Computing

Saman Froehlich[1], Daniel Große[1,2], Rolf Drechsler[1,2]

[1]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
[2]Group of Computer Architecture, University of Bremen, Bremen, Germany
{froehlich,grosse,drechsle}@cs.uni-bremen.de

***Abstract*—The Error Bounded Exact BDD Minimization (EBEBM) problem arises in approximate computing when one is trying to find a functional approximation with a minimal representation in terms of BDD size for a single output function with respect to a given error bound.**

In this paper we present an exact algorithm for EBEBM. This algorithm constructs a BDD representing all functions, which meet the restrictions induced by the given error bound. From this BDD we can derive an optimal solution.

We compute the exact solutions for all functions with up to 4 variables and varying error bounds. Based on the results we demonstrate the benefit of our approach for evaluating the quality of heuristic approximation algorithms.

I. Introduction

Approximate computing refers to a class of problems, which relax the requirements between the specification and its implementation [1]. The practical motivation for approximate computing arises due to the growing number of applications being inherently error resilient, such as media processing, recognition, and data mining. These applications usually don't require exact results, either due to limitations of human senses, because a golden solution does not exist (such as web search, etc.) or because an approximate solution is good enough [2], [3].

There are different approaches to design approximate circuits. One approach is to make use of voltage over-scaling or over-clocking which induces timing errors (see [4], [5] as examples). Another approach is to approximate a given function by substituting the function with a similar other function, which is more cost effective in the number of gates or the critical path length. Examples can be found in [6], [7], [8], [1], [9]. This paper focuses on the latter approach of functional approximation.

Binary Decision Diagrams (BDDs) have become a widely used data structure for the representation of Boolean functions since Bryant introduced efficient algorithms for their construction and manipulation [10]. Among many different areas BDDs have also been studied in logic synthesis, since they allow to combine aspects of circuit synthesis and technology mapping. There has been a renewed interest in multiplexor-based design styles (e.g., [11]), since multiplexor nodes can often be realized at very low cost (e.g., Pass Transistor Logic (PTL)). In addition, layout aspects can be considered during the synthesis step which allows to guarantee high design quality (see e.g. [12], [13]). In this context, circuits derived from BDDs often result in smaller netlists. Recently, the application of BDDs for approximate computing has been investigated in [6], [8], but with the focus on exact error metric computation.

The size of the BDD representation of a Boolean function can be directly mapped to the complexity of its hardware implementation (see for example [14], [15], [16]). Since the BDD representation is canonical [10], there are no degrees of freedom left to reduce the size of the corresponding BDD, once the function and the variable ordering of the BDD are fixed.

In this paper we relax the "exact requirements" for a Boolean function during the BDD construction in such a way, that we allow a given number e of output bits to differ from the given function specification f. In doing so we aim to find the smallest BDD representing f with at most e output deviations. We call this problem *Error Bounded Exact BDD Minimization* (EBEBM). We present an exact algorithm to solve EBEBM in this paper. Solving EBEBM for a given function f allows us to substitute f by another function $\hat{f}$ with a (potentially much) smaller BDD representation. As a consequence a more compact circuit can be created using $\hat{f}$ instead of f.

Since EBEBM determines the smallest BDD and exact BDD minimization for the classical case is know to be NP-complete, one can not expect an algorithm which scales to large functions. However, computing exact results allows to evaluate the quality of heuristic synthesis algorithms. Besides the exact algorithm devised, we demonstrate this in the experimental results of this paper.

In summary, the main contributions of this paper are:

1) Definition of the EBEBM problem
2) Presentation of a BDD-based exact algorithm for EBEBM
3) Demonstration of the effectiveness of the exact algorithm by comparing it to a naive enumeration approach
4) Quality evaluation for heuristics using exact results

The remainder of the paper is structured as follows: At first the related work concerning BDDs and approximate computing is discussed in Section II. We present the preliminaries on BDDs in Section III and give some brief basic definitions applicable in approximate computing. Section IV includes the definition of the EBEBM problem and the BDD-based algorithm to find an exact solution. The experimental results are presented in Section V. This section includes the comparison to the naive approach as well as the quality evaluation for heuristics. Finally, Section VI concludes the paper.

II. Related Work

From a general perspective most closely related to the considered problem of this paper is the minimization of incompletely specified functions. The idea is that for some values x one *does not care* whether $f(x) = 1$ or $f(x) = 0$. Incompletely specified functions can be represented by a Boolean function f and a Boolean function g, called the *don't care set*, such that $f(x)$ is a don't care value if $g(x) = 0$. The minimization problem is to find a function $\hat{f}$, called *cover*, whose BDD representation has a small number of nodes, referred to as $B(\hat{f})$, such that $f(x) \wedge g(x) \leq \hat{f}(x) \leq f(x) \vee \bar{g}(x)$for all x. Putting simply, $\hat{f}(x)$ must agree with $f(x)$ whenever x satisfies $g(x) = 1$, but we don't care what value $\hat{f}(x)$ assumes when $g(x) = 0$.

The associated decision problem is called *Exact BDD Minimization* (EBM) and asks for a given function f, a don't care set g, and a size bound b, whether there exists a function $\hat{f}$ as in the above equation such that $B(\hat{f}) \leq b$. EBM has been thoroughly examined in many papers. The authors of [17] have proven the NP-completeness. Different approaches have been proposed to solve it. The authors of [18] proposed an implicit approach to find an exact solution. An example for one of the existing heuristics can be found in [19].

In the context of approximate computing different (general) synthesis approaches have been proposed. The methodology of [20] maps the problem of approximate synthesis into a classical logic synthesis problem. Consequently, the capabilities of existing synthesis tools can be fully utilized for approximate logic synthesis. In [21] a two-level synthesis approach is described which aims at minimizing circuit area for a given error rate threshold. The approach has been extended for multi-level synthesis in [22]. Both approaches aim at minimizing circuit area by respecting a given *rate significance* threshold, which is a composite metric based on worst-case error and error rate. The core algorithm is based on automatic test pattern generation methods and relaxes the definition of a redundant fault to minimize circuit area. The authors of [23] have proposed an algorithm for safe over- and underapproximation in the context of verification. The exact problem has not been addressed.

As mentioned in the introduction already, recently two approaches using BDDs in approximate computing have been proposed. In [8] BDDs have been utilized to analyze the exact error rate of imprecise adders. The authors of [6] employed BDDs to solve the *Approximate BDD Minimization* (ABM) problem, which aims at finding a cover with respect to a given error metric and a given BDD size bound. They have also introduced how to compute several error metrics for multi output functions using BDDs. However, neither the exact minimization problem nor an exact algorithm has been considered.

III. Preliminaries

A. Binary Decision Diagrams

Since Bryant introduced efficient algorithms for the construction and manipulation of BDDs in [10], they have become a state-of-the-art data structure in verification and logic synthesis.

A BDD is a graph-based canonical representation of a Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ that is based on the Shannon decomposition $f = x_i f_{x_i} + \overline{x_i} f_{\overline{x_i}}$ $(1 \leq i \leq n)$. Applying this decomposition recursively allows dividing the function into many smaller sub-functions. Solid and dashed lines in BDDs refer to the high and low successor of a node respectively. BDDs are ordered in a sense that the Shannon decomposition is applied with respect to a given variable ordering.

The size of the BDD depends on the variable ordering and many heuristics have been proposed for its optimization. We will only consider a fixed variable ordering throughout this paper and we assume that this is the natural one $x_1 < x_2 < \ldots < x_n$ unless specified otherwise. Recall, we denote the size of the BDD of a function f by $B(f)$.

B. Error Metrics in Approximate Computing

Before we can define the considered error metric, we provide some basic definitions. We define the ON- and the OFF-set of a function f as follows: The ON-set $ON(f)$ denotes the set of all input-vectors x for which $f(x) = 1$. The OFF-set $OFF(f)$ denotes the set of all input-vectors x for which $f(x) = 0$. By $|ON(f)|$ we mean the size of the ON-set of f and by $|OFF(f)|$ the size of the corresponding OFF-set.

The *error-rate* of an approximation $\hat{f}$ for a given function f is defined as

$$er\left(f, \hat{f}\right) = \frac{\sum\limits_{x \in \mathbb{B}^n} \left[\hat{f}(x) \neq f(x)\right]}{2^n}.$$

i.e. the ratio of the errors observed in the output value as a result of approximation to the total number of input combinations.

Please note in literature also other error metrics are considered (for instance worst-case error, average-case error). However, these metrics are not in the focus of this paper.

IV. Error Bounded Exact BDD Minimization

In this section we formulate the EBEBM problem and introduce an algorithm for finding an exact solution. The core idea of the proposed exact algorithm is to reduce the EBEBM problem itself to the construction of a BDD from which we can extract the exact solution.

A. EBEBM Problem Formulation

Error Bounded Exact BDD Minimization (EBEBM) aims at finding a Boolean function $\hat{f}$, called *cover*, for a given Boolean single output function $f : \mathbb{B}^n \to \mathbb{B}$ and an error bound e, such that $B(\hat{f})$ is minimal and there exist at most k input combinations X_{i_j}, $j = 1 \dots k$, $i_j \in \{1, \dots, 2^n\}, k \leq e$ for which $\hat{f}(X_{i_j}) \neq f(X_{i_j})$.

A necessary condition for a function $\hat{f}$ to be a solution to the EBEBM problem is to fulfill the equation

$$er\left(f, \hat{f}\right) \leq \frac{e}{2^n}, \tag{1}$$

where n is the number of inputs for f.

The problem is *trivial*, if $e \geq |ON(f)|$ or $e \geq |OFF(f)|$. In this case the result is the constant $zero$- or one-function respectively. We will only consider the non trivial case from here on. We handle it as follows:

Let $f : \mathbb{B}^n \to \mathbb{B}$. There are 2^n possible input combinations to f. Let $X_i \in \mathbb{B}^n, i = 1 \dots 2^n$ denote a possible input combination. Let $Y \in \mathbb{B}^{2^n}$ and y_i be the ith entry of Y. We can then define a function $\phi : \mathbb{B}^{2^n} \times \mathbb{B}^n \to \mathbb{B}$ for a fixed Y as

$$\phi(Y; X_i) = \begin{cases} f(X_i), & y_i = 0 \\ \\ \overline{f(X_i)}, & y_i = 1 \end{cases}, \quad i = 1 \dots 2^n. \tag{2}$$

The circuit representing ϕ is depicted in Fig. 1. One can see, that y_i encodes whether $\phi(Y; X_i)$ equals $f(X_i)$ or the inversion of $f(X_i)$. Thus the number of input combinations for which $\phi \neq f$ is equal to the number of ones in Y.

This leads to the additional condition

$$\sum_{i=1}^{2^n} y_i \leq e. \tag{3}$$

Input vectors Y for which Eq. 3 holds will be called $valid$.

It is important to note that whenever an input vector Y fulfills Eq. 3, ϕ also fulfills Eq. 1. This is caused by the fact that

$$\sum_{i=1}^{2^n} y_i = \sum_{i=1}^{2^n} \left[\phi(Y; X_i) \neq f(X_i)\right], \tag{4}$$

since

$$y_i = 1 \Leftrightarrow \phi(Y; X_i) \neq f(X_i), \quad \forall i$$

due to the definition of ϕ (see Eq. 2). Substituting the left hand term of Eq. 3 by the right hand term of Eq. 4 and dividing it by 2^n leads to the proposition.

In the next section we describe how to reduce this general problem formulation itself to a BDD problem, i.e. we use a BDD to represent valid solutions to ϕ for a concrete given function f and an error bound e. Note that this BDD represents *all* possible approximations $\hat{f}$ for f with at most e output deviations. The best one can be extracted from it.

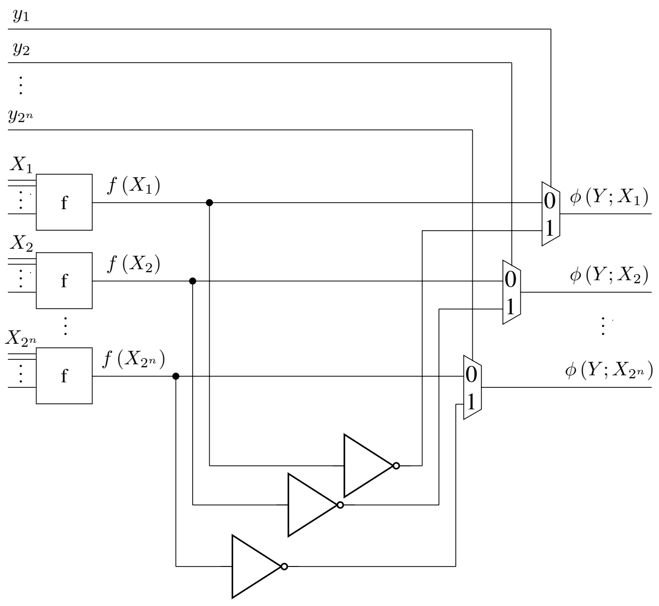


Fig. 1. Circuit representing ϕ

B. EBEBM as BDD

We can now define how to create a BDD representing Eq. 2 with respect to Eq. 3. Setting $(y_1, \dots, y_{2^n}, x_1, \dots, x_n) =: \sigma$, we define a function $F(\sigma)$:

$$F(\sigma) = \begin{cases} \phi(Y; x), & Y \text{ is valid} \\ 0, & \text{otherwise} \end{cases}.$$

The BDD of $F(\sigma)$ contains the BDDs of $\phi(y; x)$ with all valid combinations of Y as sub-graphs (according to Eq. 3). Due to the chosen variable order where the variables Y, which control the output deviations, are the top-variables, we can extract the BDD of ϕ with a valid combination of Y by simply traversing the BDD of F for 2^n levels downwards while taking no more than e high-branches. This limits the maximum number of input combinations for which the resulting function ϕ differs from f to at most e. The remaining sub-graph is the BDD, which represents ϕ with a valid combination of Y.

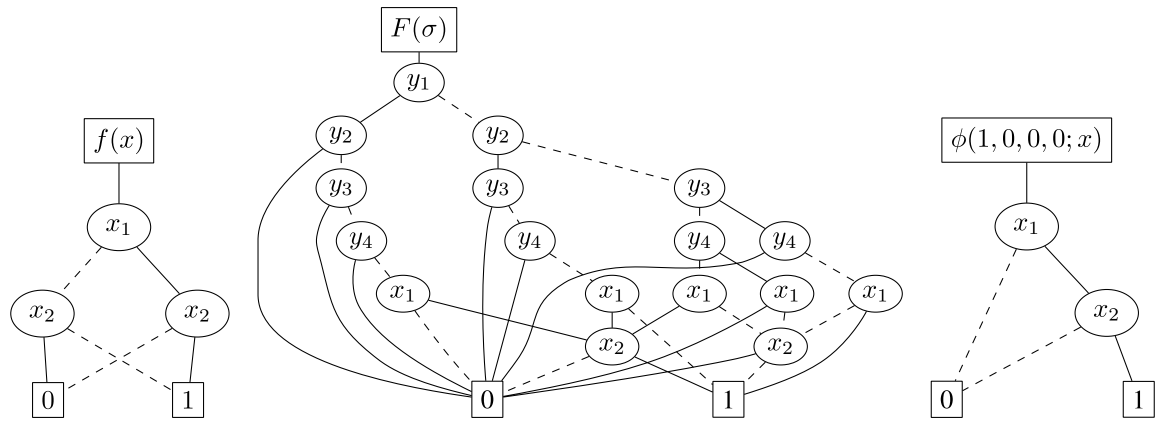
We can easily find the BDD for a minimal cover $\hat{f}$, by comparing all sub-graphs and picking one with the smallest size.

An example is depicted in Fig. 2. Fig. 2(a) shows the BDD of $f(x) = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_2$.

Fig. 2(b) shows $F(\sigma)$ for $e = 1$. We give a description of several paths shown in this figure for clarity:

$y_1 = 1 \to y_2 = 1 \to 0$
: This path leads directly to zero, because Eq. 3 is violated. e is set to 1. Since y_1 and y_2 both are set to 1 by taking the first two high-branches, the sum of all y_i is at least 2. But this means at least 2 changes would be set which contradicts $e = 1$. Hence, the resulting function ϕ is not valid and we get 0 in the BDD.

$y_1 = 0 \to y_2 = 0 \to y_3 = 0 \to y_4 = 0 \to \phi(0, 0, 0, 0; x)$
: This path leads to the BDD of the function



(a) $f(x) = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_2$ (b) $F(\sigma)$, $e = 1$ (c) $\hat{f}(x) = \phi(1,0,0,0;x)$

Fig. 2. Example for finding $\hat{f}$

$\phi(0,0,0,0;x)$. Since all elements in Y are set to 0, this function equals the original function f.

$y_1 = 1 \rightarrow y_2 = 0 \rightarrow y_3 = 0 \rightarrow y_4 = 0 \rightarrow \phi(1,0,0,0;x)$

This path leads to the BDD of the function $\phi(1,0,0,0;x)$. This BDD is smaller than the BDD of f and is valid, since we only take one high-branch.

One result of our exact algorithm $\hat{f}(x) = x_1 \cdot x_2$ can be seen in Fig. 2(c). Note that the result is not unique. For instance the function $\hat{f}_2(x) = \overline{x_1} \cdot \overline{x_2}$ would have also been a valid result, since its BDD has the same size as that of $\hat{f}$ and it differs from f only for the input vector $X = (1,1)$.

In the following section we present the experimental results.

V. Experimental Results

All experiments have been carried out on an Intel® Xeon® CPU E5-2630 v3 @ 2.40GHz with 64GB memory running Linux (Fedora release 22). We've implemented all approaches in C++ using the CUDD 3.0.0 package [24].

A. Comparison to the Naive Approach

In order to evaluate our approach, we have implemented a naive approach for comparison. This naive approach enumerates all possible don't care sets, enumerates all possible truth tables for each don't care set and creates the corresponding BDDs. Finally, the smallest BDD is returned.

We have executed both algorithms for all 256 functions with 3 variables and all 65536 functions with 4 variables, using the error bounds $e = 1 \dots 3$ and $e = 1 \dots 7$, respectively. All problems become *trivial* for higher values of e.

The obtained results can be found in the Tables I and II. The total computation time, the average computation time at and the worst case computation time wt for each value of e can be found in the respective columns. The term worst case computation time refers to the longest time needed to calculate the result for a single function. The term average computation time refers to the total time needed to calculate the results of all functions divided by the number of functions.

Due to the rising number of *trivial* cases, the average computation time is significantly smaller than the worst case computation time for higher values of e.

Because of the fact that the naive approach would check the same function more then once, if implied by different don't care sets, our approach scales much better for large values of e. This results in smaller average and worst case computation times. We reach an improvement of one order of magnitude in computation time for 4 variables at around $e = 5$.

The naive approach is faster, when it comes to smaller values for e. This is caused by the additionally introduced variables in our algorithm. While there are not many functions, which are checked twice in the naive approach if e is small, our algorithm builds a much larger BDD. This increase in runtime is usually negligible, since the main purpose of the proposed approach is to generate benchmarks with exact results to evaluate the quality of approximation heuristics using a small number of variables. When the number of variables is small and the value for e is small, then the runtime is fast enough using either the naive approach or the proposed BDD method.

To further investigate the influence of the value of e, we have chosen the function *pope* with 6 variables from the benchmark-suite *ESPRESSO* [25], [26] and compared the runtime of the algorithms for different values of e. The *ESPRESSO*-suite, is a well known two-level minimizer. We use the first bit of the results as output. The results can be seen in Table IV. The first column denotes the values for e. The second and the third column give the runtime for the naive and the BDD approach, respectively. The fourth column denotes the ratio of the runtimes of both approaches. The BDD approach is slower if we set e to 1 or 2. As soon as e reaches 3, the BDD-based approach turns out to be faster as has been observed already in the previous experiments. It is already more than twice as fast for e set to 5.

B. Evaluation of the Quality of Heuristic Approximation Approaches

We've developed the exact algorithm to be able to evaluate the quality of heuristics to solve the EBEBM problem.

TABLE I
COMPARISON OF NAIVE APPROACH AND BDD-SOLVER FOR FUNCTIONS WITH 3 VARIABLES

e	total naive [s]	total BDD [s]	at naive [s]	at BDD [s]	wt naive [s]	wt BDD [s]
1	0.007013	0.018062	0.00003	0.00007	0.00004	0.00012
2	0.025687	0.030604	0.00010	0.00012	0.00017	0.00018
3	0.039645	0.021506	0.00015	0.00008	0.00062	0.00032

TABLE II
COMPARISON OF NAIVE APPROACH AND BDD-SOLVER FOR FUNCTIONS WITH 4 VARIABLES

e	total naive [s]	total BDD [s]	at naive [s]	at BDD [s]	wt naive [s]	wt BDD [s]
1	7.67	21.28	0.00012	0.00032	0.00029	0.00102
2	98.94	100.07	0.00151	0.00153	0.00230	0.00260
3	917.69	444.73	0.01400	0.00679	0.02800	0.01300
4	5,663.40	1,289.70	0.08642	0.01968	0.15000	0.03000
5	23,330.00	2,798.10	0.35599	0.04270	0.80000	0.10200
6	59,119.00	3,948.30	0.90208	0.06025	2.00000	0.15000
7	60,612.00	2,392.00	0.92486	0.03650	5.10000	0.22000

TABLE III
COMPARISON OF APPROACH FOR ABM FOUND IN [6] AND BDD-SOLVER FOR FUNCTIONS FROM THE $ESPRESSO$-SUITE [25].

Example	original BDD size	# inputs	e	CPU time BDD [s]	size BDD	size $rounding\ down$	size $rounding\ up$	size $rounding$
con1	11	7	4	$timeout$		-	-	-
dc1	8	4	5	0.053	2	4	5	**3**
dc2	11	8	2	106.899	9	**9**	10	**9**
dist	32	8	2	122.768	25	**25**	-	-
ex5	4	8	1	$timeout$		-	-	-
exps	39	8	2	91.897	34	-	-	-
f51m	21	8	2	142.224	19	-	-	-
inc	12	7	4	$timeout$		-	-	-
lin	39	7	3	506.834	35	-	-	-
m1	2	6	5	$timeout$		-	-	-
m2	4	8	3	$timeout$		-	-	-
m3	2	8	3	$timeout$		-	-	-
m4	6	8	3	$timeout$		-	-	-
misex1	7	8	3	$timeout$		-	-	-
newcpla2	23	7	3	494.588	15	-	-	-
newcwp	8	4	5	0.100	4	6	**5**	6
newtag	10	8	2	189.125	7	9	**7**	**7**
newill	19	8	2	178.562	15	**18**	-	-
pope	14	6	4	233.978	9	-	-	-
risc	11	8	2	83.258	8	10	**9**	**9**
root	9	8	2	94.929	5	8	**5**	**5**
sqn	25	7	3	515.105	21	-	-	-
sqr6	6	6	3	9.323	3	**3**	5	**3**
tms	15	8	2	81.731	10	14	-	-
Z5xp1	8	7	3	465.217	5	**5**	6	**5**

As benchmarks we have chosen various problems from the benchmark-suite *ESPRESSO*. The heuristics for functional approximation proposed in [6] are used for approximation and compared to the exact algorithm. Again, we only use the first bit of the results as output. We discard the problems with incompletely specified functions, because we don't need a don't care set and only use the subset of functions with 8 inputs or less and no don't cares in the first output bit. We choose e to be at most 2 for functions with 8 variables, 3 for functions with 7 variables, 4 for functions with 6 variables, 5 for functions with 5 variables and up to 7 for functions with 4 variables, unless the problem becomes trivial or no reduction is possible.

If no reduction was possible, we increase e until either a reduction takes place, we reach the time limit or the problem becomes trivial. If the problem is trivial in the first place, we reduce e until the problem is non-trivial.

We use our algorithm to calculate the exact result for each problem and report the size of the BDD and the runtime. As approximation heuristics we use the algorithms proposed in [6], i.e. we apply the *rounding down*, the *rounding up* and the *rounding* operators to the problem set. For each operator we increase the number of rounding levels until Eq. 1 is violated and use the result with the highest number of rounding levels not violating Eq. 1 as an approximate result. The time limit has been set to $3,600$ seconds.

Table III contains the results. The first column denotes the name of the used circuit. The second and third column give the original size of the corresponding BDD and the number of inputs. The fourth column denotes the used value for e. The next two columns give the results for our proposed exact BDD-algorithm: The CPU time needed and the size of the BDD for

TABLE IV
COMPARISON OF RUNTIME FOR DIFFERENT VALUES OF e FOR BENCHMARK *pope* FROM THE $ESPRESSO$-SUITE [25].

e	runtime naive [s]	runtime BDD [s]	ratio $\frac{BDD}{\text{naive}}$
1	0.01	0.05	5.00
2	0.39	0.63	1.62
3	10.92	9.12	0.84
4	330.03	233.63	0.71
5	8,089.07	3,626.30	0.45

the resulting function. The remaining three columns provide the BDD sizes for the mentioned heuristic approximation operators. If a heuristic was not able to find a function with a smaller BDD than the original function with respect to e, then the entry for the corresponding result is -. The runtimes for the heuristics are not shown, since it is below 0.01 seconds for every problem. The results of the best heuristics in terms of BDD size for each problem are in given in bold.

It can be seen, that the heuristic algorithms failed to find a function with the smallest possible BDD representation with respect to e for the majority of problems. Here we would like to stress the importance of an exact algorithm: It is not guaranteed that a reduction in terms of BDD size is possible, if e is not large enough. Only the optimal results obtained by our approach allows to judge the quality of the heuristic. In particular in cases where the heuristic was not able to find a reduction we know based on the optimal results if a better solutions exists or no reduction is possible at all.

From the table it can be seen that the operator *rounding down* succeeded to reduce the BDD size for more problems than the other two operators, but the best results in terms of BDD size were determined by the *rounding* operator.

VI. CONCLUSIONS

In this paper we have introduced the *Error Bounded Exact BDD Minimization* (EBEBM) problem and presented an exact algorithm to solve it. The algorithm reduces the problem itself to the construction of a BDD which contains all possible approximations for the considered function for a given error bound. An optimal solution can be easily extracted.

The evaluation of the experiments has shown, that our approach is more effective than the naive approach. It scales better for larger error bounds, since unlike the naive approach, it does only check every possible solution exactly once and makes use of efficient BDD reduction techniques. Furthermore, we have demonstrated the benefits of optimal solutions when evaluating the quality of functional approximation heuristics.

In future work we plan to consider various heuristics to solve the EBEBM problem. The quality of these heuristics can be measured using the algorithm presented in this paper. We also want to investigate the more general problem, where the variable ordering for the given function is not fixed, and extend it from Boolean to multivalued functions to further increase its applicability. First tests have shown that the size of $F(\sigma)$ can be greatly reduced using dynamic variable ordering.

Acknowledgments. This work was supported in part by the German Research Foundation (DFG) within the project MANIAC (DR 287/29-1), the Reinhart Koselleck project DR 287/23-1 and by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

REFERENCES

[1] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: modeling and analysis of circuits for approximate computing," in *ICCAD*, Nov. 2011, pp. 667–673.

[2] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *DAC*, 2013, pp. 113:1–113:9.

[3] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *DAC*, 2015, pp. 120:1–120:6.

[4] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *International Symposium on Microarchitecture*, 2003, pp. 7–18.

[5] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Designing a processor from the ground up to allow voltage/reliability tradeoffs," in *HPCA*, 2010, pp. 1–11.

[6] M. Soeken, D. Große, A. Chandrasekharan, and R. Drechsler, "BDD minimization for approximate computing," in *ASP-DAC*, 2016, pp. 474–479.

[7] N. Zhu, W. L. Goh, and K. S. Ye, "An enhanced low-power high-speed adder for error-tolerant application," in *ISIC*, 2009, pp. 69–72.

[8] C. Yu and M. Ciesielski, "Analyzing imprecise adders using BDDs - a case study," in *ISVLSI*, 2016, pp. 152–157.

[9] C.-H. Lin and I.-C. Lin, "High accuracy approximate multiplier with error correction." in *ICCD*, 2013, pp. 33–38.

[10] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *TC*, 1986.

[11] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," in *DAC*, 2009, pp. 270–275.

[12] L. Macchiarulo, L. Benini, and E. Macii, "On-the-fly layout generation for ptl macrocells," in *DATE*, 2001, pp. 546–551.

[13] A. Mukherjee and M. Marek-Sadowska, "Wave steering to integrate logic and physical syntheses," *TVLSI*, vol. 11, no. 1, pp. 105–120, 2003.

[14] C. Scholl and B. Becker, "On the generation of multiplexer circuits for pass transistor logic," in *DATE*, 2000, pp. 372–379.

[15] S. B. Akers, "Binary decision diagrams," *TC*, vol. 27, no. 6, pp. 509–516, Jun. 1978.

[16] R. Drechsler, M. Kerttu, P. Lindgren, and M. Thornton, "Low power optimization techniques for BDD mapped circuits using temporal correlation," *Canadian Journal of Electrical and Computer Engineering*, vol. 27, no. 4, 2002.

[17] M. Sauerhoff and I. Wegener, "On the complexity of minimizing the obdd size for incompletely specified functions," *TCAD*, vol. 15, pp. 1435–1437, 1996.

[18] A. L. Oliveira, L. P. Carloni, T. Villa, and A. L. Sangiovanni-Vincentelli, "An implicit formulation for exact BDD minimization of incompletely specified functions," in *VLSI: Integrated Systems on Silicon*, L. C. R Reis, Ed. Springer US, 1997, pp. 315–326.

[19] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton, "Heuristic minimization of bdds using don't cares," in *Proc. of DAC*, 1994, pp. 225–231.

[20] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: systematic logic synthesis of approximate circuits," in *DAC*, 2012, pp. 796–801.

[21] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *DAC*, 2010, pp. 957–960.

[22] ——, "A new circuit simplification method for error tolerant applications," in *DATE*, 2011, pp. 1566–1571.

[23] K. Ravi, K. L. McMillan, T. R. Shipple, and F. Somenzi, "Approximation and decomposition of binary decision diagrams," in *DAC*, 1998.

[24] F. Somenzi, "CUDD: CU Decision Diagram package-release 3.0.0," University of Colorado at Boulder, 2015.

[25] T. R. of the University of California, "Espresso," https://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm.

[26] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984.