NEW RESULTS TO BDD TRUNCATION METHOD FOR EFFICIENT TOP EVENT PROBABILITY CALCULATION

YUCHANG MO1*, FARONG ZHONG1, XIANGFU ZHAO1, QUANSHENG YANG2, and GANG CUI3
1Department of Computer Science, Zhejiang Normal University
Jinhua, 321004, P.R. China
2School of Computer Science and Engineering, Southeast University
Nanjing, 210089, P.R. China
3School of Computer Science and Technology, Harbin institute of technology
Harbin, 150001, P.R. China
*Corresponding author. E-mail : myc@zjnu.cn
Received October 19, 2011
Accepted for Publication February 08, 2012

A Binary Decision Diagram (BDD) is a graph-based data structure that calculates an exact top event probability (TEP). It has been a very difficult task to develop an efficient BDD algorithm that can solve a large problem since its memory consumption is very high. Recently, in order to solve a large reliability problem within limited computational resources, Jung presented an efficient method to maintain a small BDD size by a BDD truncation during a BDD calculation. In this paper, it is first identified that Jung’s BDD truncation algorithm can be improved for a more practical use. Then, a more efficient truncation algorithm is proposed in this paper, which can generate truncated BDD with smaller size and approximate TEP with smaller truncation error. Empirical results showed this new algorithm uses slightly less running time and slightly more storage usage than Jung’s algorithm. It was also found, that designing a truncation algorithm with ideal features for every possible fault tree is very difficult, if not impossible. The so-called ideal features of this paper would be that with the decrease of truncation limits, the size of truncated BDD converges to the size of exact BDD, but should never be larger than exact BDD.

KEYWORDS : Binary Decision Diagram, BDD, Fault Tree, Truncation, Top Event Probability

1. INTRODUCTION

Fault tree analysis (FTA) is an important technique for reliability and safety analysis. Bell telephone laboratories developed the concept in 1962 for the U.S. Air Force. It was later adopted, and extensively applied by the Boeing Company. FTA is now widely used in the electronics, nuclear and aerospace industries.

Binary decision diagrams (BDD) are the state-of-the-art data structure to handle Boolean functions [1,2]. Since their introduction to the reliability field, they have proved to be in many cases a very powerful tool [3]. They made possible the assessment of complex fault-trees both qualitatively (computation of minimal cutsets) and quantitatively (exact calculation of the top event probability). Tools such as Aralia can in many cases give more accurate results than conventional tools, while running 1000 times faster [4]. The BDD algorithm has become a very popular method to calculate an exact top event probability (TEP) of a small or intermediate size reliability problem [5-9].

The traditional BDD algorithm calculates an exact TEP since it does not employ any approximations such as a truncation, a rare-event approximation, or a delete-term approximation. Since the BDD algorithm has the time and memory consuming problem, especially for large problems, it has been difficult to solve large reliability problems such as fault trees for accident sequences in a Probabilistic Safety Assessment (PSA). In order to solve a large complex problem with limited computational resources, the following two attempts have been made to reduce BDD size, which is measured by the number of nodes in the BDD structure.

1) To generate smaller exact BDD using efficient ordering.

The size of BDD encoding fault trees depends heavily on the chosen ordering. Actually, there is often a variation of several orders of magnitude between the sizes of two BDD built over different orderings. From a theoretical point of view, finding the best ordering is an intractable task [10,11].

Many ordering heuristics have been proposed in the literature [12,13]. The most simple heuristic consists in numbering basic events by means of a depth first left most (DFLM) traversal of the tree [3,14]. DFLM heuristic is
“good” because it preserves, at least to a certain extent, the locality of the events. For instance, it numbers basic events of modules consecutively. Using the weighting techniques, DFLM can be modified into a weighting DFLM heuristic (WDFLM): Each basic event has a weight with a value of 1. The weight of each gate is obtained by adding the weights of its inputs. When the weights are known in the whole tree, a depth-first traversal of the tree is made. At each level, the sons of a gate are chosen by order of increasing weights. During this traversal, the basic events are put in the ordered list as soon as they are encountered. WDFLM is widely used in the [12-16]. Both DFLM and WDFLM are used in this paper.

2) To generate smaller truncated BDD using truncation algorithm

If the fault tree consists of a large number of repeated events and gates, the exact equivalent BDD can not be generated due to the limitations of the computational resources. Thus, the MCSs and exact TEP can not be calculated.

A common practice in the evaluation of large fault trees is to truncate the BDD. Using the smaller truncated BDD, approximate TEP and truncated MCSs can be calculated. Truncation on order can be applied to keep only disjoint paths that contribute significantly to the top event probability. Reference [17] developed such a truncation algorithm to compute truncated BDD from the exact BDD or directly from the tree, where disjoint paths of order (number of variables) more than k can be discarded without much consequence. Truncation on probability can equivalently be applied to keep only the significant disjoint paths, where the BDD truncation is applied efficiently to all BDD operation levels by using the upper probability. Reference [18, 19]. Whenever the upper probability is less than the truncation limit during the recursive BDD operation, the remaining recursive BDD operations are cancelled and the calculation returns with a terminal node 0.

In this paper, we focus on the second line of research, i.e., generating smaller truncated BDD efficiently. The major contributions of our work are as follows:

1. It is identified that, for practical use, the most recent BDD truncation algorithm in [19] can be improved upon.
2. A more efficient truncation algorithm is proposed, which can generate truncated BDD with smaller size and approximate TEP with smaller truncation error.
3. It is found that, to design a truncation algorithm with ideal features for every possible fault tree is very difficult, if not impossible. The so-called ideal features would be that with the decrease of truncation limits, the size of truncated BDD converges to the size of exact BDD, but should never be larger than exact BDD.

The rest of this paper is organized as follows: Section 2 introduces the related algorithms: traditional exact BDD algorithm and Jung’s BDD truncation algorithm. Section 3 explains that why Jung’s BDD truncation algorithm can be improved for practical use. Section 4 presents our more efficient BDD truncation algorithm, and gives both the experimental results and conclusion on the performance comparison. Finally, Section 5 concludes our paper with an outline of our future work.

2. PROBLEM STATEMENT

2.1 Jung's BDD Truncation Algorithm

A BDD is a directed acyclic graph where a Shannon decomposition is implemented at every node. The Shannon decomposition is succinctly defined in terms of the ternary If-Then-Else (ITE) connectives as

\[ F = \text{ite}(x, F_1, F_0) = x F_1 + \bar{x} F_0 \]  

In the fault tree to BDD conversion, the BDD operation is recursively performed on the higher priority variable x as

\[ F \text{ op } G = \text{ite}(x, F_1, F_0) \text{ op } \text{ite}(y, G_1, G_0) \]  

where \( \text{op} \) is an AND or OR Boolean operator.

In BDD truncation algorithm as shown in Table 1, the BDD truncation is applied efficiently to all BDD operation levels by using the upper probability p.

The upper probability p is maintained as follows: 1) \( p \leq 1 \) when the BDD operation starts. 2) \( p \leq p^*p \), when the first Boolean operation \( F_1 \text{ op } G_1 \) or \( F_1 \text{ op } G_1 \) in Eq.(2) is started. 3) \( p \leq p^*p^*(1.0-p_1) \) when the second Boolean operation \( F_2 \text{ op } G_2 \) or \( F_2 \text{ op } G_2 \) in Eq.(2) is initiated. Here, \( p_1 \) denotes the probability of x.

The upper probability p is also incorporated into the computation-table, which maps hash_key(\( \text{op}, F, G \)) to a BDD node H encoding \( \text{op}, F, G \) and an upper probability p.

The calculation \( \text{op}, F, G \) will be performed and the calculated BDD H together with hash_key(\( \text{op}, F, G \)) and p will be stored in the computation-table if the computation-table does not have an entry for calculation \( \text{op}, F, G \).

If \( \{ \text{hash_key}(\text{op}, F, G), T, q \} \) exists in the computation-table and the stored upper probability q \( \geq \) the current upper probability p, that is to say, the stored T is bigger than the BDD to be calculated, the current calculation \( \text{op}, F, G \) is not performed and T is returned. If q<p, the stored and small BDD T and q are replaced with new calculated larger BDD H and p.

Whenever the upper probability is less than the truncation limit during the recursive BDD operation, the remaining recursive BDD operations are cancelled and the calculation returns with a terminal node 0.
2.2 Performance Analysis

An ideal BDD truncation algorithm should have following two features: 1) Whatever the truncation limits are, the size of truncated BDD should not be larger than exact BDD. 2) With the decrease of truncation limits, the sizes of truncated BDD converges to the size of exact BDD.

With the ideal BDD truncation algorithm, the sizes of truncated BDD with different truncation limits can be depicted in Fig.1.

![Fig. 1. Performance of an Ideal BDD Truncation Algorithm](image)

Table 1. Jung’s BDD Truncation Algorithm

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TruncComputation ( op, F, G, p)=</td>
</tr>
<tr>
<td>1</td>
<td>If (p &lt; Truncation_limit ) Return 0</td>
</tr>
<tr>
<td>2</td>
<td>If ( F=0 or 1 ) or ( G=0 or 1 ) or ( F=G ) Return terminal_case ( op, F, G )</td>
</tr>
<tr>
<td>3</td>
<td>If ( x&gt;y ) swap ( F, G )</td>
</tr>
<tr>
<td>4</td>
<td>k1←hash_key(&lt; op, F, G &gt;)</td>
</tr>
<tr>
<td>5</td>
<td>If (computation-table has entry (k1, T, q ) )</td>
</tr>
<tr>
<td>6</td>
<td>If ( q ≥ p ) return T</td>
</tr>
<tr>
<td>7</td>
<td>If ( x=y )</td>
</tr>
<tr>
<td>8</td>
<td>H₁←TruncComputation ( op, F₁, G₁, p·p₁ )</td>
</tr>
<tr>
<td>9</td>
<td>H₂←TruncComputation ( op, F₉, G₉, p·(1.0-p₁) )</td>
</tr>
<tr>
<td>10</td>
<td>Else</td>
</tr>
<tr>
<td>11</td>
<td>H₁←TruncComputation ( op, F₁, G₁, p·p₁ )</td>
</tr>
<tr>
<td>12</td>
<td>H₂←TruncComputation ( op, F₉, G₉, p·(1.0-p₁) )</td>
</tr>
<tr>
<td>13</td>
<td>If (unique-table has entry (hash_key(&lt;x, H₁, H₂&gt;),Q ))</td>
</tr>
<tr>
<td>14</td>
<td>H←Q</td>
</tr>
<tr>
<td>15</td>
<td>Else</td>
</tr>
<tr>
<td>16</td>
<td>H←Insert-in-ITE-table (x, H₁, H₂)</td>
</tr>
<tr>
<td>17</td>
<td>If (computation-table has entry (k₁, T, q ) )</td>
</tr>
<tr>
<td>18</td>
<td>If ( p &gt; q ) update-computation-table ( { k₁ , T , q }, { k₁, H₁ , p } )</td>
</tr>
<tr>
<td>19</td>
<td>Else</td>
</tr>
<tr>
<td>20</td>
<td>Insert-in-computation-table (k₁, H, p )</td>
</tr>
<tr>
<td>21</td>
<td>Return H</td>
</tr>
</tbody>
</table>
Unfortunately, we find that Jung’s algorithm does not have the above identified features. As an illustration, an example fault tree shown in Fig.2 is used. All event probabilities are 0.001. The results are listed in Table 2. All related BDDs are presented in Fig.3-5.

The BDD in Fig.3 is generated by the exact BDD algorithm with DFLM ordering. Its BDD size is 14. The left BDD in Fig.4 is generated by the BDD truncation algorithm with DFLM ordering and truncation limit = 1.0E-2*exact TEP. Its BDD size is 20. All the truncation points in this BDD are identified. For the exact BDD on the right hand, the sub-BDD where the truncation happens is also identified. The left BDD in Fig.5 is generated by the BDD truncation algorithm with DFLM ordering and truncation limit = 1.0E-4*exact TEP or 1.0E-6*exact TEP. Its BDD size is 19. All the truncation points in this BDD are identified. For the exact BDD on the right hand, the sub-BDD where the truncation happens is also identified.

Since identical BDDs are reused in order to save required memory, there can be several different paths from the top node to one of sub-BDDs. Thus, different upper probability \( p \) can be calculated. According to "If \( (p > q) \) update-computation-table \( \{ \text{hash key}(< \text{op}, F, G>), T, q \}, \{ \text{hash key}(< \text{op}, F, G>), H, p \} \) " in the BDD truncation algorithm, for the same reused exact sub-BDD, the BDD truncation should be performed if the current upper probability \( p \) is larger than previous upper probabilities. Thus, different truncated BDDs for the same reused exact sub BDD are generated. Most of them are not isomorphism and can not be reused.

When truncation limit = 1.0E-8*exact TEP, the BDD generated by the BDD truncation algorithm is same as the exact BDD, i.e., no truncation happens.

![Fig. 2. An Example Fault Tree](image)

![Fig. 3. BDD generated by exact BDD algorithm](image)

**Table 2.** Results of Jung’s BDD Truncation Algorithm for the Fault tree in Fig.2

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Exact BDD</th>
<th>BDD truncation</th>
<th>BDD truncation</th>
<th>BDD truncation</th>
<th>BDD truncation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truncation limit</td>
<td>0</td>
<td>1.0E-2 *exact TEP</td>
<td>1.0E-4 *exact TEP</td>
<td>1.0E-6 *exact TEP</td>
<td>1.0E-8 *exact TEP</td>
</tr>
<tr>
<td>TEP</td>
<td>5.997e-009</td>
<td>5.994e-009</td>
<td>5.99699e-009</td>
<td>5.99699e-009</td>
<td>5.997e-009</td>
</tr>
<tr>
<td>BDD SIZE</td>
<td>14</td>
<td>20</td>
<td>19</td>
<td>19</td>
<td>14</td>
</tr>
</tbody>
</table>
Now, further tests are performed. Nine different fault trees are selected in the website http://iml.univmrs.fr/~arauzy/aralia/benchmark.html. All event probabilities are 0.001. Subsequent gates of the same type are contracted to form a single gate. WDFLM ordering is used. The fault trees are solved with two kinds of calculations:

1. Exact TEP calculations by the exact BDD algorithm.
2. TEP calculations by the BDD truncation algorithm with truncation limits that are $10^{-i}$ ($1 \leq i \leq 10$) times smaller than the exact TEPs. The truncation limits are calculated by the multiplication of the scaling factor $10^{-i}$ and the exact TEPs.

---

**Fig. 4.** BDDs Generated with Truncation Limit = 1.0E-2*Exact TEP

---

**Fig. 5.** BDDs Generated with Truncation Limit = 1.0E-4*Exact TEP or 1.0E-6*Exact TEP
All results of the exact BDDs or truncated BDDs are depicted in Fig. 6 (--●--: exact BDD, ---■--: truncated BDD). All the scaling factors of truncation limits are on the x-axis, and the BDD size data are on the y-axis. It can be concluded that for Jung’s BDD truncation algorithm, there are only two samples “Das9202” and “Edf9203” have similar results as Fig.1, the ideal scenario. For the remaining seven samples except for “Baobab1”, there are some truncated BDD larger than exact BDD before calculated approximate TEP reaches exact TEP.

3. NEW BDD TRUNCATION ALGORITHM

3.1 Algorithm Description

The main drawback of Jung’s BDD truncation algorithm is that: for the same reused exact sub-BDD, in other words, for the same calculation <op, F, G>, different truncated BDDs are generated and most of them are not isomorphism.

The basic idea of our new BDD truncation algorithm
as shown in Table 3 is that for the same calculation \(<\text{op}, F, G>\), another BDD truncation is still performed if the current upper probability \(p\) is larger than previous upper probabilities. The difference is that the generated larger truncated BDD is used to replace previous stored truncated BDD related to the same calculation \(<\text{op}, F, G>\) with smaller upper probabilities. Thus, the chance of BDD reuse increases and the accuracy of calculated approximate TEP is improved.

The operations on unique-table and ITE table at Lines 13 to 16 in Table 2 is updated with some new operations in Lines 13 to 22 in Table 3. The implementation of above idea has following key points:

1) The unique-table should be revised. \(\text{hash\_key}(<\text{op}, F, G>)\) is also incorporated into the new unique-table, which maps \(\text{hash\_key}(<x, H_1, H_0>)\) to a BDD node \(H=\text{ite}(x, H_1, H_0)\) and the corresponding \(\text{hash\_key}(<\text{op}, F, G>)\).

2) Line 21. The generated larger truncated BDD is used to replace the previous stored truncated BDD

<table>
<thead>
<tr>
<th>Table 3. New BDD Truncation Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
</tbody>
</table>
related to the same calculation <op, F, G> with smaller upper probabilities.

3) Line 15. If a BDD node H=ite(x, H1, H0) is shared by different <op, F, G> and <op, M, N> (hash_key(<op, F, G>)=k1 and hash_key(<op, M, N>)=k1'), it should be excluded from following replacement operation caused by Line 21. Thus, we update the entry \{ hash_key(<x, H1, H0>), hash_key(<op, F, G>, Q) \} in the new unique-table with \{ hash_key(<x, H1, H0>), -1 Q \}, where “-1” is a meaningless key.

As an illustration, the example fault tree shown in Fig.2 is used. The results are listed in Table 4. The data above shows conclusively that all truncated BDDs are same as the exact BDD. The reason is that although there are different calculations relating to the same calculation <op, F, G>, no truncation happens in the last calculation. Thus, with the replacement operation, previous truncations are erased and the exact BDD is obtained.

3.2 Performance Comparison

Now, performance comparison between Jung’s BDD truncation algorithm and our BDD truncation algorithm are performed. The nine fault trees are solved with our new BDD truncation algorithm. All event probabilities are 0.001. Subsequent gates of the same type are contracted to form a single gate. WDFLM ordering is used.

All truncation results (BDD size and approximate TEP) are depicted in Fig. 7 († : exact results, ■ : Jung’s

Table 4. Results of our BDD Truncation Algorithm for the Fault Tree in Fig.2

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Exact BDD</th>
<th>BDD truncation</th>
<th>BDD truncation</th>
<th>BDD truncation</th>
<th>BDD truncation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truncation limit</td>
<td>0</td>
<td>1.0E-2 *exact TEP</td>
<td>1.0E-4 *exact TEP</td>
<td>1.0E-6 *exact TEP</td>
<td>1.0E-8 *exact TEP</td>
</tr>
<tr>
<td>BDD SIZE</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

Fig. 7. The Truncation Results of Both Two BDD Truncation Algorithms
Fig. 7. The Truncation Results of Both Two BDD Truncation Algorithms
results, -▲:- our results). It can be concluded that: 1) Our new algorithm is almost always able to obtain smaller truncated BDD and better approximate TEP than Jung’s algorithm. 2) For the new algorithm, there are five samples “Baobab1”, “Baobab3”, “Das9202”, “Das9203”, “Edf9202” and “Edf9203” have similar results as Fig.1, the ideal scenario. 3) For only three samples “Baobab2”, “Das9201” and “Edf9201”, there are some truncated BDD larger than the exact BDD before the calculated approximate TEP reaches exact TEP. According to Line 16, we know that non-isomorphic truncated BDDs are still possible to be generated for the same <op, F, G> calculation. Thus, it should be noted that to design a truncation algorithm with ideal features for every possible fault tree is very difficult, if not impossible.

The running time and storage usage of these two BDD truncation algorithms are listed in Table 5. The row marked “Ratio of running time” gives the running time ratio between Jung’s algorithm and our algorithm. The row marked “Ratio of storage usage” gives the storage usage ratio between Jung’s algorithm and our algorithm. The results of the analyzed data show that our algorithm uses less running time in 30 out of 45 cases and more storage usage in 36 out of 45 cases. The advantage in running is mainly caused by those smaller truncated BDD, whereas the disadvantage in storage usage is mainly...

<table>
<thead>
<tr>
<th>Truncation limit</th>
<th>1.0E-2 * exact TEP</th>
<th>1.0E-4 * exact TEP</th>
<th>1.0E-6 * exact TEP</th>
<th>1.0E-8 * exact TEP</th>
<th>1.0E-10 * exact TEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baobab1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>0.99224</td>
<td>1.07145</td>
<td>1.03297</td>
<td>1.02673</td>
<td>1.09108</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.999155</td>
<td>0.999652</td>
<td>0.999652</td>
<td>1.00081</td>
<td>1.00458</td>
</tr>
<tr>
<td>Baobab2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>1.00782</td>
<td>0.977328</td>
<td>0.955081</td>
<td>0.981647</td>
<td>1.01282</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.998543</td>
<td>0.998062</td>
<td>0.998062</td>
<td>1.00031</td>
<td>0.999387</td>
</tr>
<tr>
<td>Baobab3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>1.00052</td>
<td>0.965549</td>
<td>1.10624</td>
<td>1.07869</td>
<td>1.06097</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.999264</td>
<td>0.999264</td>
<td>1.00189</td>
<td>1.00584</td>
<td>1.00584</td>
</tr>
<tr>
<td>Das9201</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>1.04367</td>
<td>1.00226</td>
<td>1.14974</td>
<td>1.3055</td>
<td>1.27077</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.999227</td>
<td>0.999227</td>
<td>1.00063</td>
<td>1.00211</td>
<td>1.00211</td>
</tr>
<tr>
<td>Das9202</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>0.994148</td>
<td>1.03065</td>
<td>1.03055</td>
<td>0.925572</td>
<td>0.967628</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.999865</td>
<td>0.999865</td>
<td>0.999865</td>
<td>0.999865</td>
<td>0.999865</td>
</tr>
<tr>
<td>Das9203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>0.782353</td>
<td>0.914038</td>
<td>1.00909</td>
<td>0.981706</td>
<td>1.24916</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.999871</td>
<td>0.999742</td>
<td>0.999742</td>
<td>0.999742</td>
<td>0.999742</td>
</tr>
<tr>
<td>Edf9201</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>1.03306</td>
<td>0.998396</td>
<td>1.0171</td>
<td>1.00737</td>
<td>0.977744</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.996571</td>
<td>0.996571</td>
<td>0.994624</td>
<td>0.994111</td>
<td>0.994111</td>
</tr>
<tr>
<td>Edf9202</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>1.00849</td>
<td>0.998531</td>
<td>1.02479</td>
<td>1.00174</td>
<td>1.09554</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.996206</td>
<td>0.993201</td>
<td>0.987689</td>
<td>0.987689</td>
<td>0.988952</td>
</tr>
<tr>
<td>Edf9203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of running time</td>
<td>1.03934</td>
<td>0.991434</td>
<td>1.02915</td>
<td>1.08832</td>
<td>1.05646</td>
</tr>
<tr>
<td>Ratio of Storage usage</td>
<td>0.965413</td>
<td>0.965413</td>
<td>0.959901</td>
<td>0.975858</td>
<td>0.976009</td>
</tr>
</tbody>
</table>
caused by incorporating hash_key(<op, F, G>) into the new unique-table. It should be noted that both the advantages and disadvantages are insignificant because all ratios are very close to 1.

4. CONCLUSION AND FURTHER RESEARCH

In order to solve a large reliability problem with limited computational resources, numerous attempts have been made to reduce BDD size. The traditional approach was to find an optimal ordering by using some heuristics. An additional attempt was the development of a BDD truncation algorithm to calculate an approximate TEP.

According to Jung’s BDD truncation algorithm, the BDD truncation is applied efficiently to all BDD operation levels by using the upper probabilities. Whenever the upper probability is less than the truncation limit during the recursive BDD operation, the remaining recursive BDD operations are cancelled and the calculation returns with a terminal node 0. Using the smaller truncated BDD, approximate TEP and truncated MCSs can be calculated.

This paper presents a more efficient truncation algorithm than Jung’s BDD truncation algorithm, which can generate truncated BDD to a smaller size and approximate TEP with a smaller truncation error. It can also be concluded that to design a truncation algorithm with ideal features for every possible fault tree is very difficult, if not impossible. The so-called ideal features are that with the decrease of truncation limits, the size of truncated BDD converges to the size of exact BDD, but should never be larger than exact BDD.

The most interesting further work, in our minds, is as follows:
1) find efficient ordering to improve the performance of truncation algorithm further.
2) extend the BDD truncation practice to multistate systems and multiphase systems, where fault trees have dependent events and mutually exclusive events in addition to repeated events.

ACKNOWLEDGEMENT

This paper is supported by the NSF of China under Grant No. 60903011, 61272130, 61003101, the Research Fund for the Doctoral Program of Higher Education of China under Grant No. 2009092120030, the NSF of Zhejiang Province under Grant No. Y1100689, Y1100191, the NSF of Jiangsu Province under Grant No. BK2009267 as well as the Science and Technology Department of Zhejiang Province under Grant No. 2010C31122.

REFERENCES

