

Разбиение графа на подграфы Выделение модулей в деревьях отказа Бинарные диаграммы решений

Лень Ирина, 371 группа

Разбиение графа на подграфы (разрезание графа) (англ. Graph partition)

— представление исходного графа $G = \langle A, V \rangle$ в виде множества подмножеств вершин $A = \{A_1, A_2, \dots, A_n\}$, $A_i \subseteq V$ по определенным правилам. Обычно по условию задачи требуется: полнота $\bigcup_{i=1}^n A_i = A$, $A_i \neq \emptyset$ и ортогональность разбиения: $\forall i \neq j \ A_i \cap A_j = \emptyset$

- Задача раскраски графа.
- Задача определения числа и состава компонент связности графа.
- В задаче трассировки межсоединений печатных плат или микросхем необходимо разбиение исходной схемы на слои.
- Сегментация картинок (умные ножницы).
- Распараллеливание обработки данных.
- и т.д.

- Вариантов разбиения графа из n вершин на $n/2$ частей:

$$C_n^{n/2} = \frac{n!}{((n/2)!)^2} \approx 2^n \sqrt{2/(\pi n)}$$

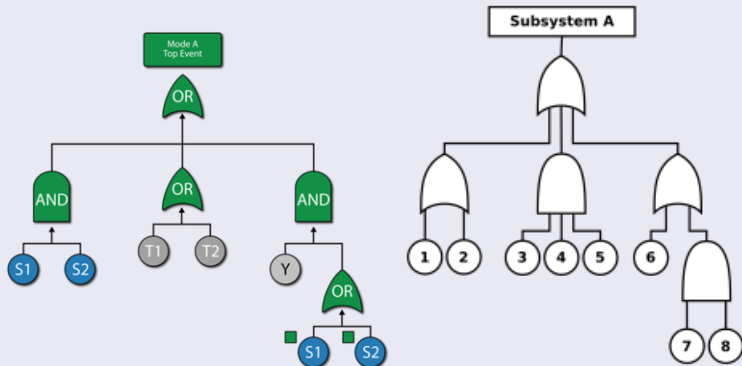
- Поиск оптимального разбиения является NP-complete задачей

Методы разбиения графа

- Покоординатное разбиение
- Рекурсивный инерционный метод деления пополам
- Многоуровневое разделение
- Деление с учётом связности (по сути, поиск в ширину)
- Алгоритм Кернигана — Лина

- METIS/ParMETIS (Karyris) — параллельные вычисления и крупномасштабные численные расчеты.
- Chaco (Sandia) — разбиение на задачи для параллельного вычисления (упрощение разработки параллельных приложений).
- Scotch (INRIA) — библиотека с реализованными классическими параллельными алгоритмами для разбиения графа.
- Jostle (now commercialized) — программный пакет, предназначенный для разделения неструктурированных мешей для использования на параллельных компьютерах с распределенной памятью.
- Zoltan (Sandia) — набор служб управления данными для параллельных, неструктурированных, адаптивных и динамических приложений.

Дерево отказов

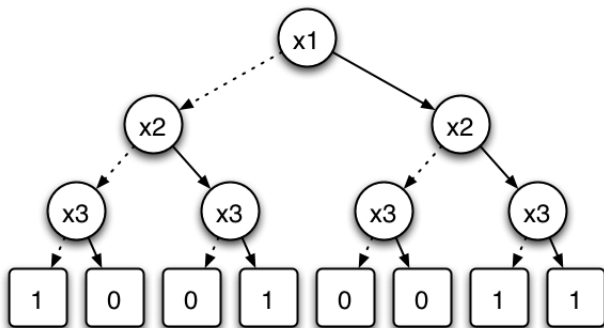


Назовем внутреннюю вершину модулем, если никакие события не встречаются в другом модуле. (Например, на левом рисунке есть 2 модуля — Top Event и OR (T1, T2), а на правом 5 — каждая внутренняя вершина является модулем)

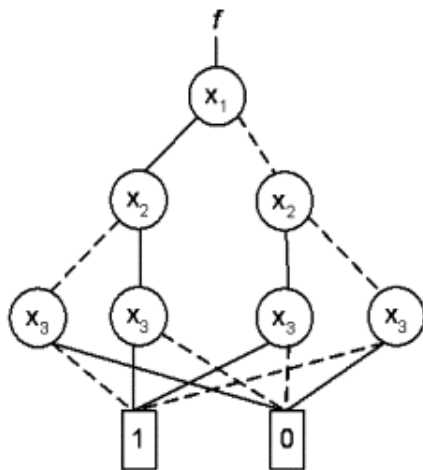
Binary Decision Diagram (BDD)

Бинарная диаграмма решений (БДР)

– форма представления булевой функции $f(x_1, x_2, \dots, x_n)$ от n переменных в виде направленного ациклического графа, состоящего из внутренних узлов решений (помеченных x_i), каждый из которых имеет по два потомка, и двух терминальных узлов (помеченных 0 и 1), каждый из которых соответствует одному из двух значений булевой функции.

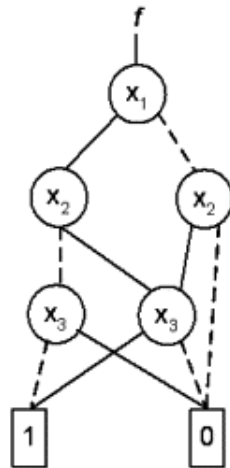


x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



(a)

OBDD



(b)

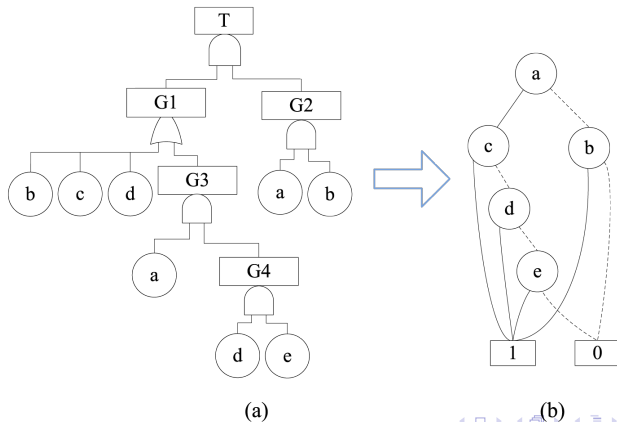
ROBDD

- Булевы базисы Грёбнера (практическую применимость в криптографии, моделирование квантовых вычислений и к задаче «Выполнимость» для конъюнктивной нормальной формы.)
- Упрощение логических схем.
- Поиск путей функционирования/отказа для системы.
- и т.д.

Преобразование FT в BDD

Задача

Хотим найти все комбинации элементов, которые приведут к отказу всей системы — минимальные сечения (MCS) или решения, применив преобразование FT в BDD.

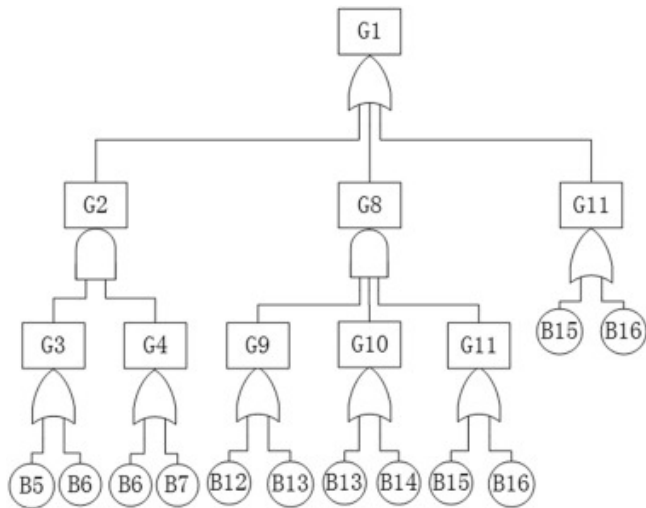


Parallel algorithm for finding modules of large-scale coherent fault trees

Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang

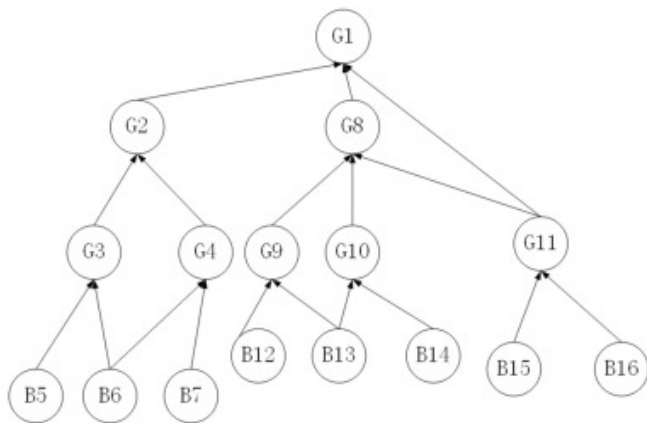
School of Reliability and Systems Engineering, Beihang University, Beijing, China

Дерево отказов (FT – fault tree)



Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang, Parallel algorithm for finding modules of large-scale coherent fault trees

Направленный ациклический граф (DAG – Directed Acyclic Graph)



Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang, Parallel algorithm for finding modules of large-scale coherent fault trees

Линейный алгоритм (LTA – Linear time algorithm)

The algorithm has 3 steps:

1. Initialize the counters; traverse the list of nodes.
2. Perform the first 'depth-first left-most' traversal of the graph to set counters; for leaves, dates #1 and #2 are identical.
3. Perform the second 'depth-first left-most' traversal of the graph in which it collects, for each internal event v , the minimum of the first dates and the maximum of the last dates of its children. v is a module if only and if:
 - a) the collected minimum is greater than the first date of v .
 - b) the collected maximum is less than the last date of v .

Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang, Parallel algorithm for finding modules of large-scale coherent fault trees

Параллельный алгоритм

V_{out} the output edge number of node in the directed acyclic graph
 V_M it means whether the node is module top or not, if and only if $V_M \leq 1$, the node is module top.

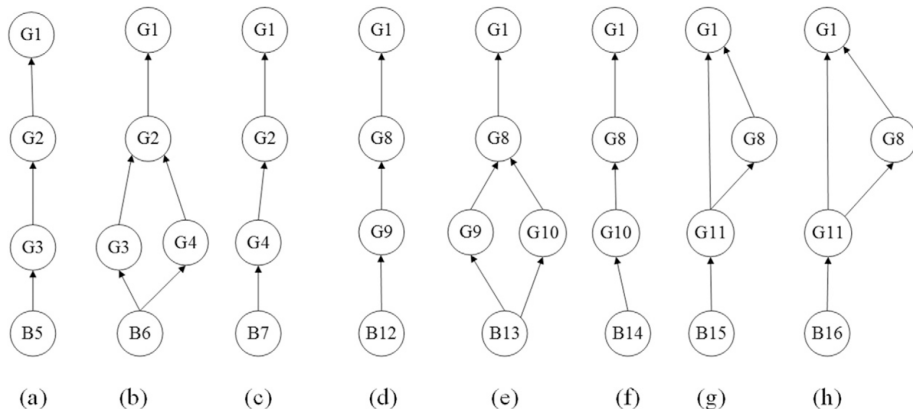
We set the V_{out} when we construct the directed acyclic graph from fault tree. So we can conclude the following parallel algorithm basing on the above description.

The algorithm consists of 4 steps:

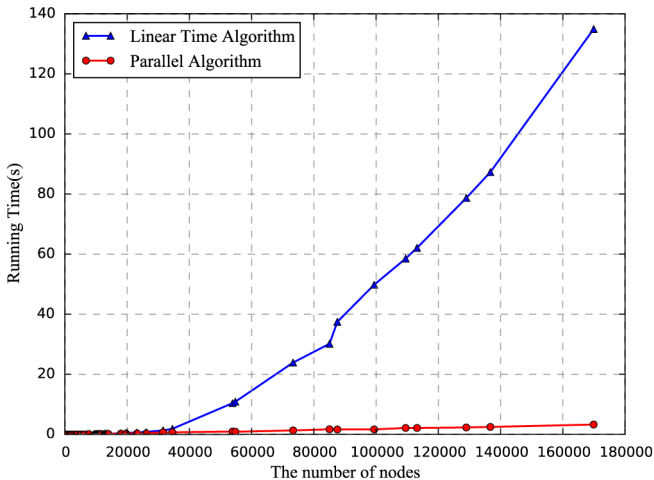
1. Initial variable. For all leave node, $V_M = V_{out}$.
2. Perform the 'depth-first left-most' traversal from the leave nodes, and judge the internal node e whether it is visited or not. If it is not visited, updating its V_M value according to the formula $e_M = \max(s_M, s_{out})$. s is an input of node e . Or determine whether the path $\bar{s}e$ is passed or not. If the path is passed, updating its V_M value according to the formula $e_M = \min(s_M, e_M - 1)$. Or updating the V_M according to formula $e_M = \min(s_M, e_M) - 1$.
3. Perform the procedure described in step 2 on every subgraph.
4. Union of the internal nodes of all subgraphs marked through above procedures, so we get all module tops. The rule to union of the nodes is described as follows. If any $V_M \leq 1$ for the same node in all subgraphs, we predict that the node is a module top. If the node does not follow the rule, it is not a module top.

Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang, Parallel algorithm for finding modules of large-scale coherent fault trees

Подграфы для примера

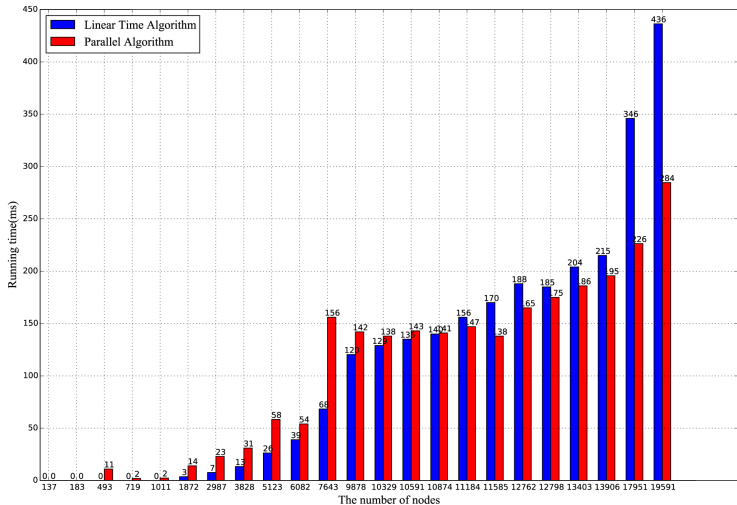


Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang, Parallel algorithm for finding modules of large-scale coherent fault trees



Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang, Parallel algorithm for finding modules of large-scale coherent fault trees

Результаты



Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang, Parallel algorithm for finding modules of large-scale coherent fault trees

BDD algorithms based on modularization for fault tree analysis

Yunli Deng, He Wang, Biao Guo

Fundamental Science on Nuclear Safety and Simulation Technology Laboratory, College of Nuclear Science and Technology, Harbin Engineering University, PR China

Рассматриваемый пример FT

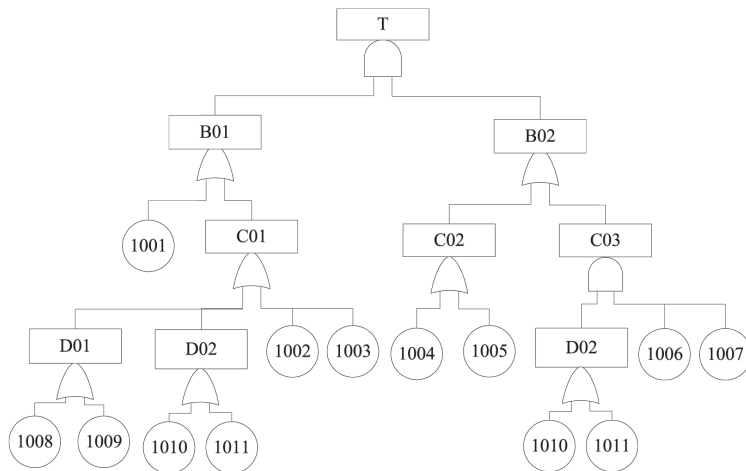


Fig. 3. An example of a fault tree.

Yunli Deng, He Wang, Biao Guo, BDD algorithms based on modularization for fault tree analysis

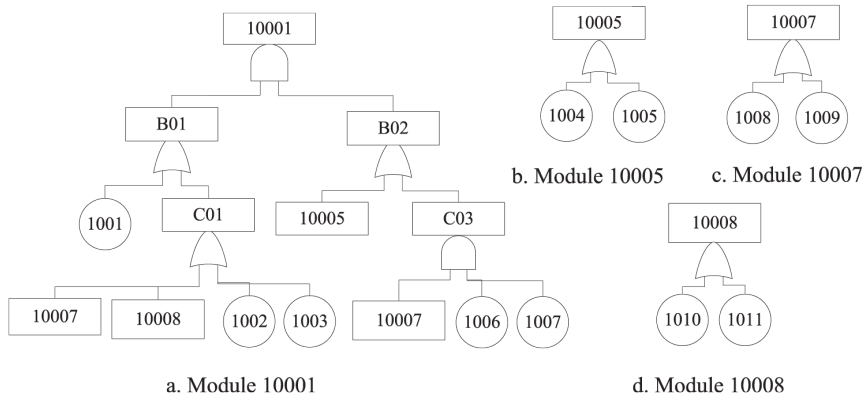


Fig. 4. Modules of the example FT.

Yunli Deng, He Wang, Biao Guo, BDD algorithms based on modularization for fault tree analysis

Модули, преобразованные в BDD

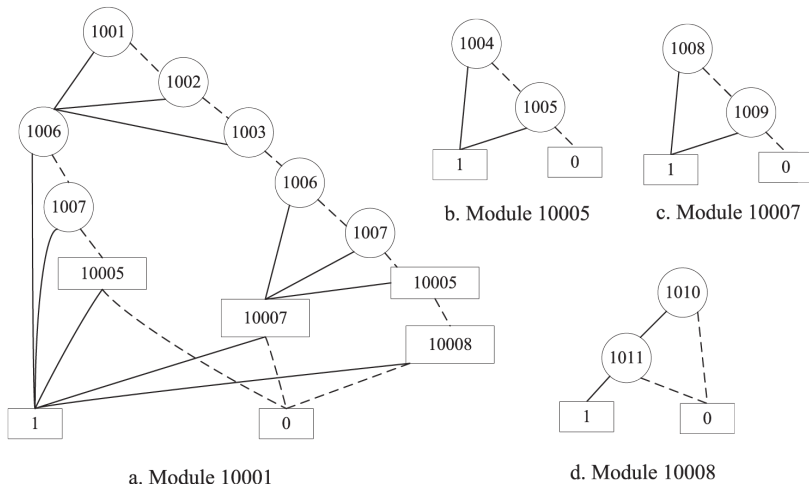


Fig. 5. BDDs of each module.

Yunli Deng, He Wang, Biao Guo, BDD algorithms based on modularization for fault tree analysis

Возможные варианты присоединения модулей

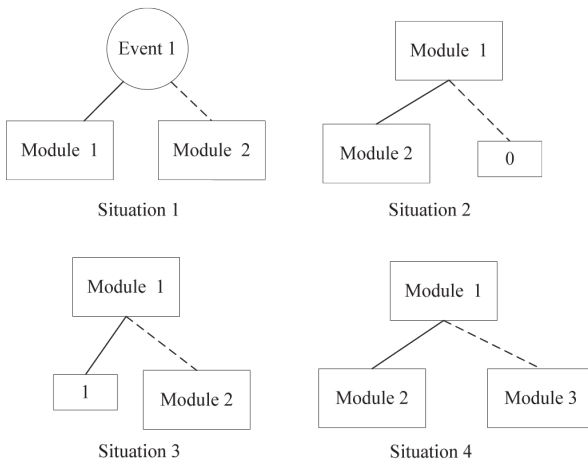


Fig. 7. Four situations in BDD structure with modules.

Yunli Deng, He Wang, Biao Guo, BDD algorithms based on modularization for fault tree analysis

Алгоритм для случая 1

```
Traversing (T) /* T=ite(x, F, G,  $p_x$ )/  
    /* F=ite(y,  $F_1, G_1, p_y$ )/  
    /* G=ite(z,  $F_2, G_2, p_z$ )/  
    if( $y > 10000$ )/*it's a module*/  
        MCS  $\leftarrow$  add( $x$ ),  $p = p_x * p$   
        Traversing(BDDs[y])/*BDDs is a hashtable to store BDDs  
of modules with the key equaling to their number*/  
    if( $z > 10000$ )  
        MCS  $\leftarrow$  add( $-x$ ),  $p = (1 - p_x) * p$   
        Traversing(BDDs[z])
```

Fig. 8. Key to deal with situation 1.

Yunli Deng, He Wang, Biao Guo, BDD algorithms based on modularization for fault tree analysis

Алгоритм для случая 2

```
Module2_Traversing (T, H) /* T=ite(x, F, G,  $p_x$ ) , where  $x < 10000$  */  
    /* H=ite(h,  $F_h$ ,  $G_h$ ,  $p_h$ ) */  
    MCS  $\leftarrow$  add(x),  $p = p_x * p$   
    if(F=1)  
        Traversing(H)  
        if(G=0)  
            else  
                MCS  $\leftarrow$  add(-x),  $p = (1 - p_x) * p$   
                Module2_Traversing(G, H)  
    else  
        Modul2_Traversing(F, H)  
        if(G=0)  
            else  
                MCS  $\leftarrow$  add(-x),  $p = (1 - p_x) * p$   
                Module2_Traversing(G, H)
```

Fig. 9. Recursive function to deal with situation 2.

Алгоритм для случая 3

```
Module3_Traversing (T, H) /* T=ite(x, F, G,  $p_x$ ) , where  $x < 10000$  */  
    /* H=ite(h, Fh, Gh,  $p_h$ ) */  
    MCS  $\leftarrow$  add(x),  $p = p_x * p$   
    if(F=1)  
        MCSs  $\leftarrow$  MCS,  $p_{mcs} \leftarrow p$ ,  $p_{top} = p + p_{top}$   
        MCS=new ,  $p = 1$   
        MCS  $\leftarrow$  add(-x),  $p = (1 - p_x) * p$   
        if(G=0)  
            Traversing(H)  
        else  
            Module3_Traversing(G, H)  
    else  
        Modul3_Traversing(F, H)  
        MCS  $\leftarrow$  add(-x),  $p = (1 - p_x) * p$   
        if(G=0)  
            Traversing(H)  
        else  
            Module3_Traversing(G, H)
```

Fig. 10. Recursive function to deal with situation 3.

Yunli Deng, He Wang, Biao Guo, BDD algorithms based on modularization for fault tree analysis

Алгоритм для случая 4

```
Module4_Traversing (T, H, K) /* T=ite(x, F, G,  $p_x$ ), where  $x < 10000$  */  
    /* H=ite( $h$ ,  $F_h$ ,  $G_h$ ,  $p_h$ ) */  
    /* K=ite( $h$ ,  $F_k$ ,  $G_k$ ,  $p_k$ ) */  
    MCS  $\leftarrow$  add( $x$ ),  $p = p_x * p$   
    if(F=1)  
        Traversing(H)  
        MCS  $\leftarrow$  add( $-x$ ),  $p = (1 - p_x) * p$   
        if(G=0)  
            Traversing(K)  
        else  
            Module3_Traversing(G, K)  
    else  
        Module2_Traversing(F, H)  
        MCS  $\leftarrow$  add( $-x$ ),  $p = (1 - p_x) * p$   
        if(G=0)  
            Module3_Traversing(F, K)  
        else  
            Module4_Traversing(F, G, H)
```

Fig. 11. Recursive function to deal with situation 4.

Table 4

FT test results of BDD and BDD based on modularization.

FT	Size	BDD based on modularization			BDD	
		Modules	Solutions	Time(s)	Solutions	Time(s)
Das9201	160	32	14217	0.97	14217	>300
Das9204	83	13	16704	0.10	16704	1.05
Chinese	61	1	392	0.03	392	0.03
Isp9604	338	76	746574	3.82	—	—
Isp9606	130	17	1776	0.27	1776	106
Isp9607	138	14	150436	0.94	150436	>300
edf9201	314	23	579720	10.6	—	—

Size — количество элементов в дереве, Solutions — количество минимальных сечений

Yunli Deng, He Wang, Biao Guo, BDD algorithms based on modularization for fault tree analysis

- Z.F. Li, Y. Ren, L.L. Liu, Z.L. Wang, Parallel algorithm for finding modules of large-scale coherent fault trees
- Y. Dutuit, A. Rauzy, A linear-time algorithm to find modules in fault trees
- Yunli Deng, He Wang, Biao Guo, BDD algorithms based on modularization for fault tree analysis
- <http://www.cs.cornell.edu/bindel/class/cs5220-f11/slides/lec19.pdf>