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

Лень Ирина, 371 группа

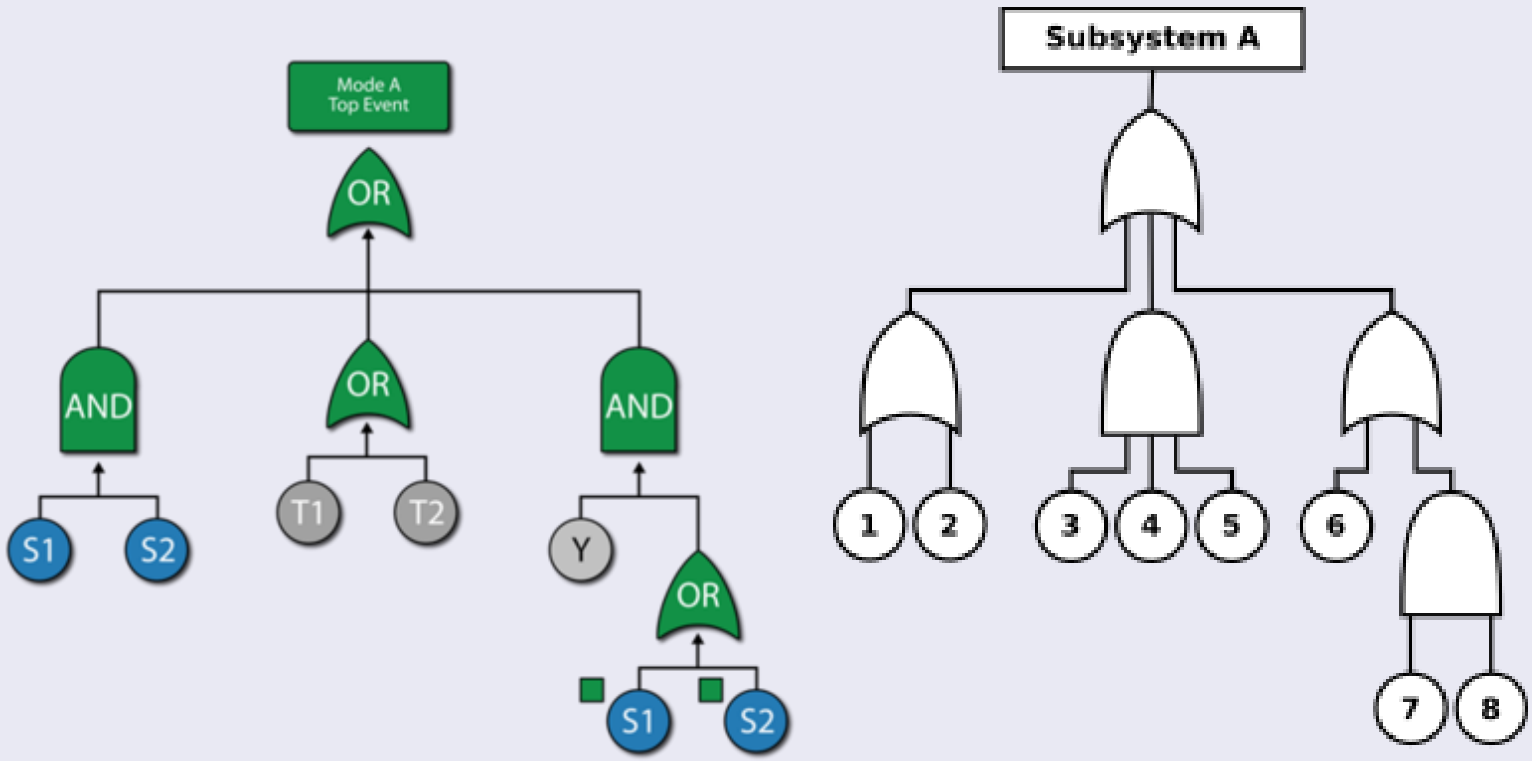
СПбГУ, 2018

# Постановка задачи

- Разбить дерево отказов на модули при помощи линейного и параллельного алгоритмов.
- Преобразовать дерево отказов в BDD.
- Реализовать поиск минимальных сечений по BDD с модулями и без них.

# Разбиение на модули

## Дерево отказов

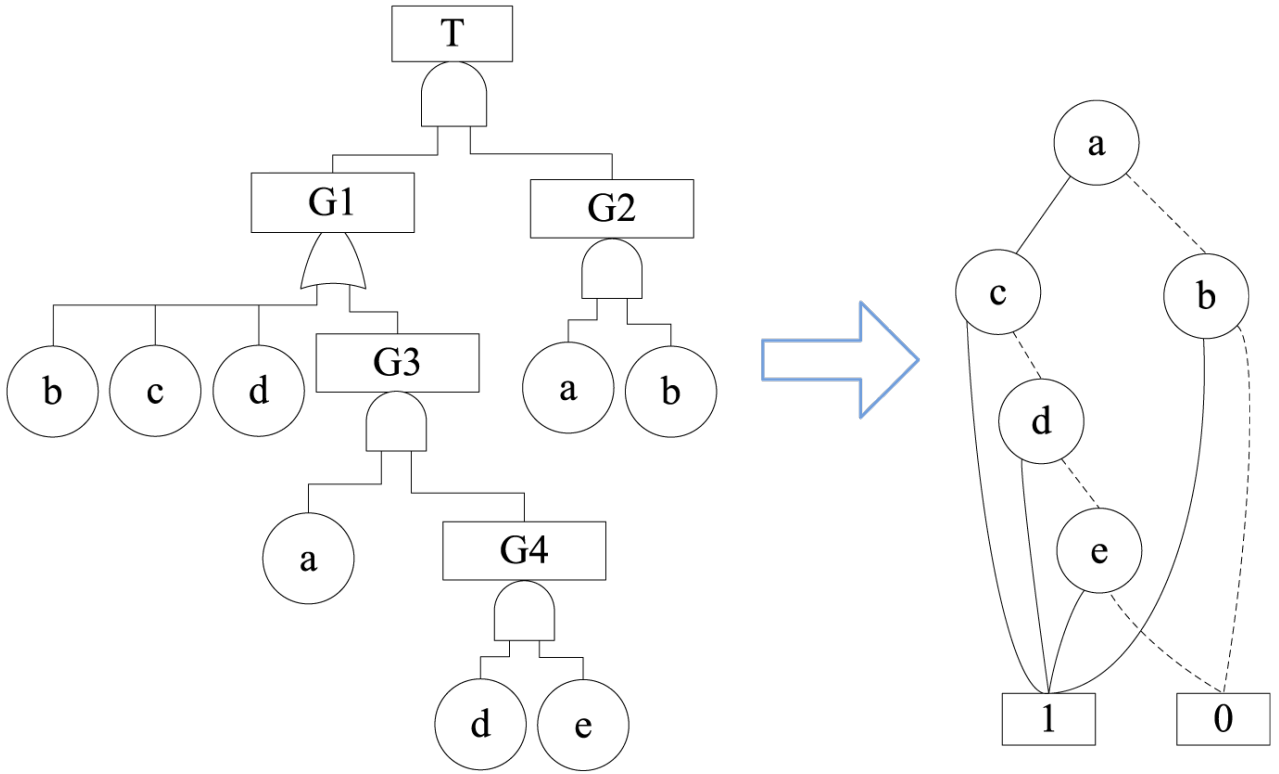



Назовем внутреннюю вершину модулем, если никакие события не встречаются в другом модуле. (Например, на левом рисунке есть 2 модуля — Top Event и OR (T1, T2), а на правом 5 — каждая внутренняя вершина является модулем)

# 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.




(a) (b)

# 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.

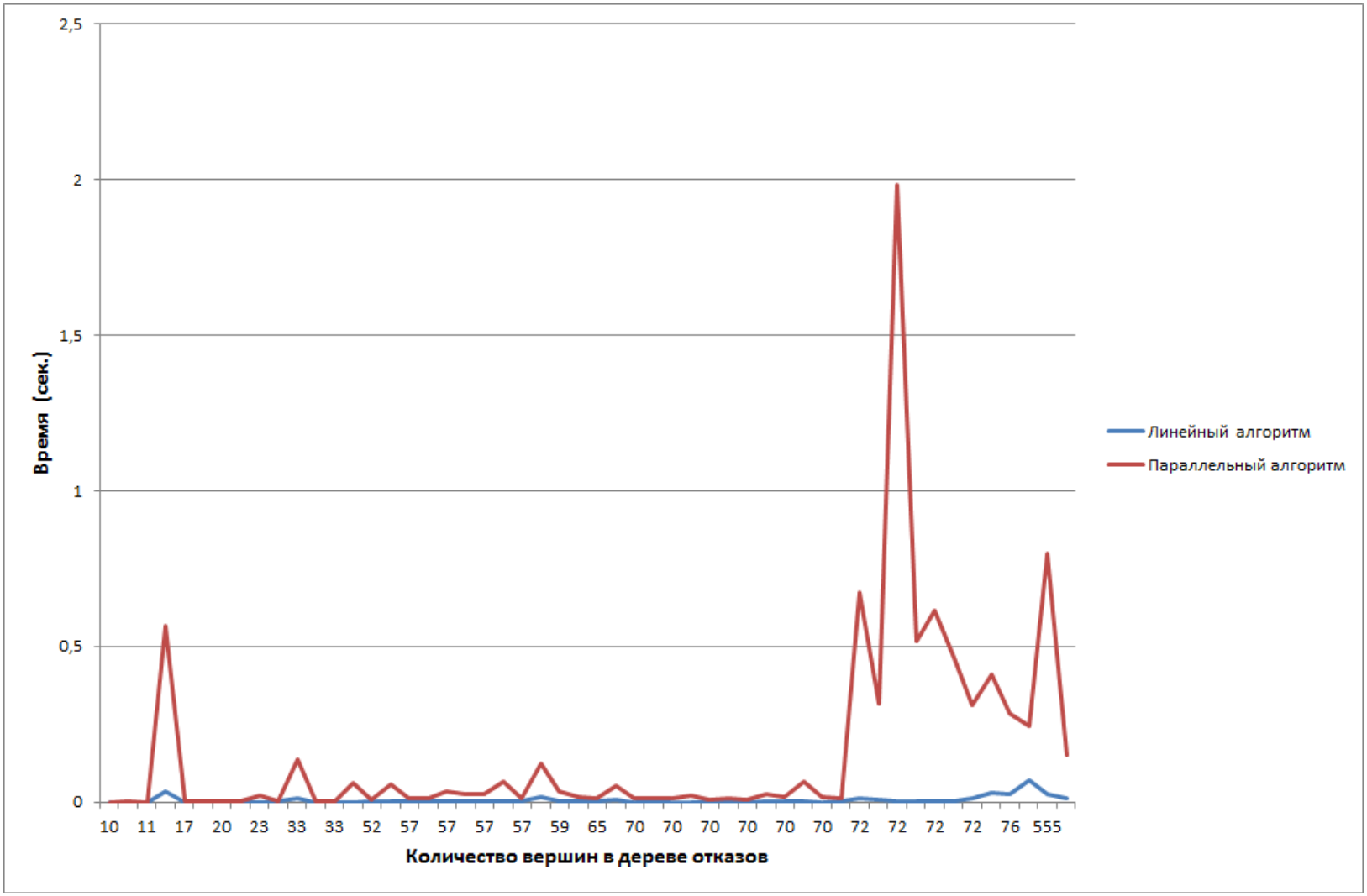
We set the $V_{out}$ when we construct the directed acyclic graph from fault tree. So we can conclude the following parallel algorithm basing on the above description.

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 $\overline{se}$ 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

# Результаты разбиения на модули




На небольших графах лучше справляется линейный алгоритм.

# Возможные варианты присоединения модулей

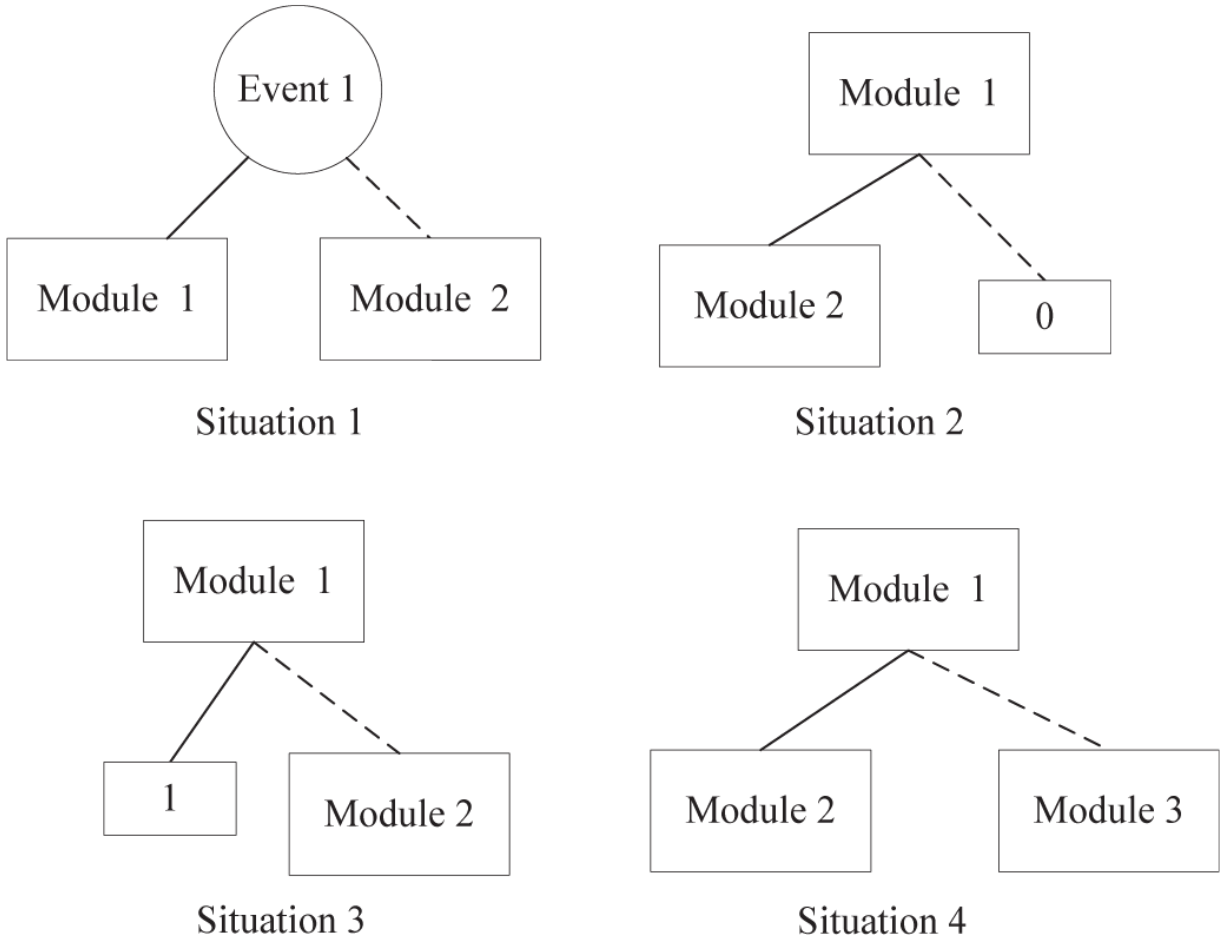



**Fig. 7.** Four situations in BDD structure with modules.

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 ← 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 ← 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 ← add(x), p = p_x * p
        if(F=1)
            Traversing(H)
            if(G=0)
            else
                MCS ← add(-x), p = (1 − p_x) * p
                Module2_Traversing(G, H)
        else
            Modul2_Traversing(F, H)
            if(G=0)
            else
                MCS ← add(-x), p = (1 − p_x) * p
                Module2_Traversing(G, H)
```

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

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

# Алгоритм для случая 3

```
Module3_Traversing (T, H) /* T=ite(x, F, G, p_x) , where x<10000*/
                          /* H=ite(h, F_h, G_h, p_h)*/
        MCS ← add(x), p = p_x * p
        if(F=1)
            MCSs ← MCS, p_mcs ← p, p_top = p + p_top
            MCS=new , p = 1
            MCS ← add(-x), p = (1 − p_x) * p
            if(G=0)
                Traversing(H)
            else
                Module3_Traversing(G, H)
        else
            Modul3_Traversing(F, H)
            MCS ← 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

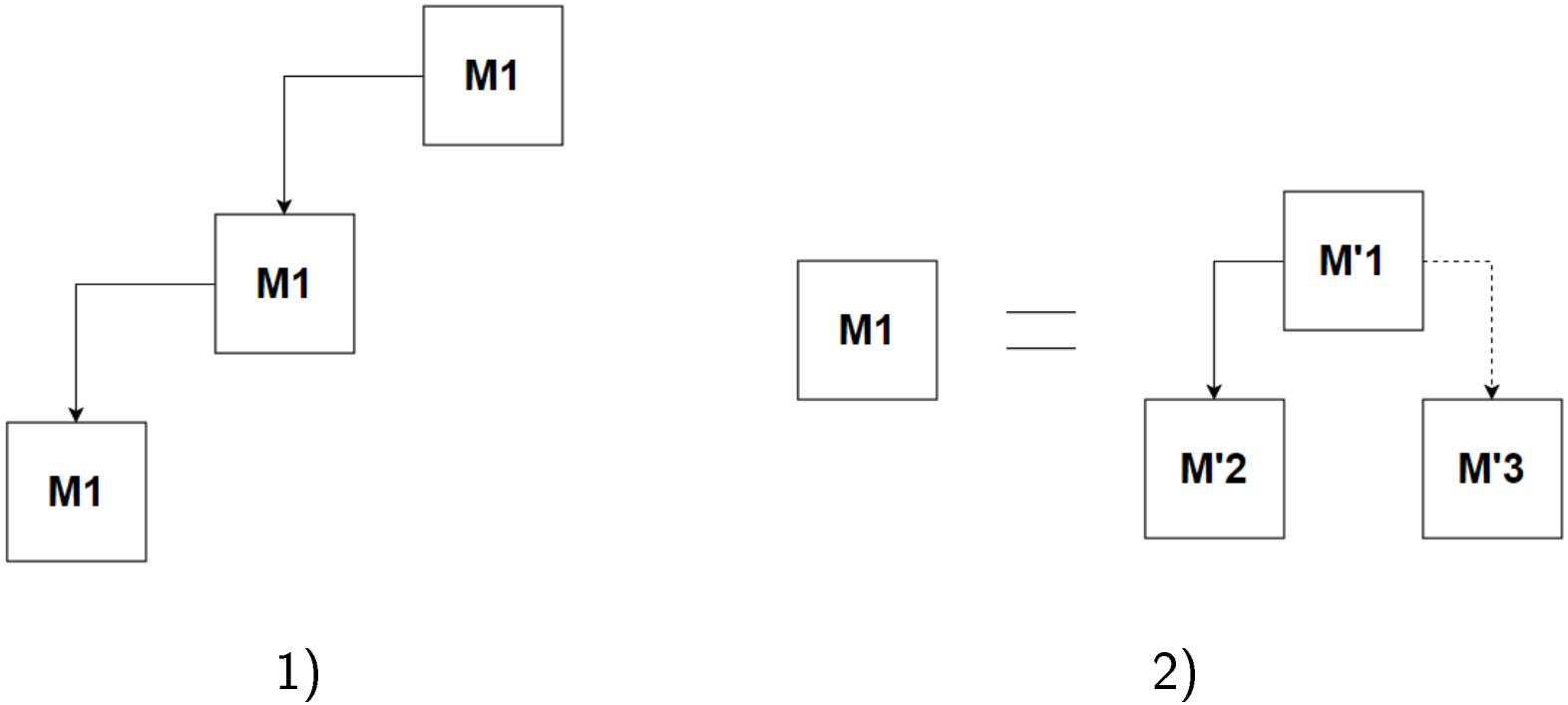
# Алгоритм для случая 4

```
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 ← add(x), p = p_x * p
        if(F=1)
            Traversing(H)
            MCS ← add(-x), p = (1 − p_x) * p
            if(G=0)
                Traversing(K)
            else
                Module3_Traversing(G, K)
        else
            Module2_Traversing(F, H)
            MCS ← 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.

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

# Проблемы алгоритма




1) 2)

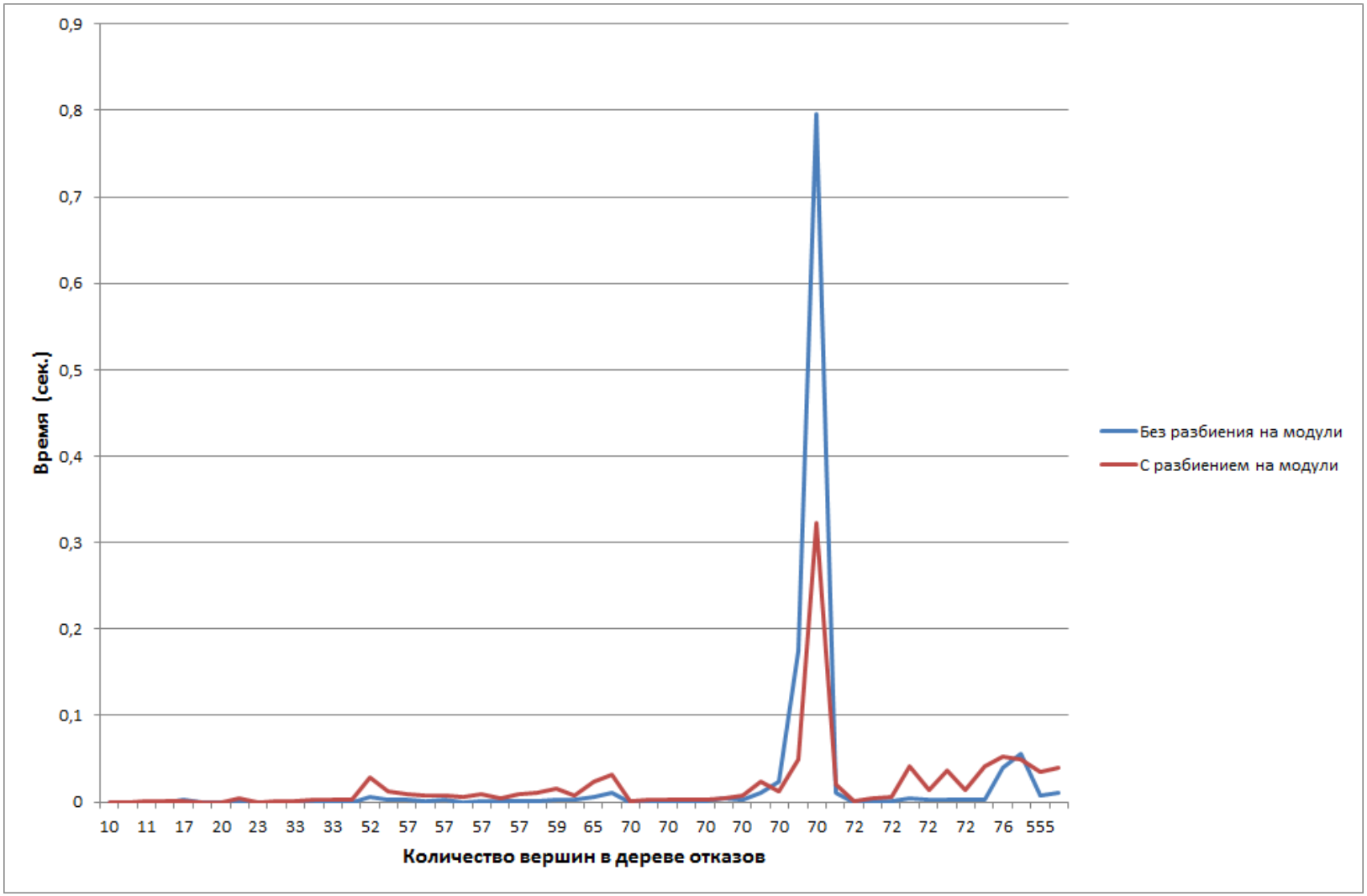
1. 3 и более последовательно соединенных модуля (рис. 1).
2. Внутри модуля могут быть другие BDD, которые тоже могут содержать (рис. 2).

# Мой алгоритм

Traverse(T, modNum) where T = ite (x, H, L)

1. if(T.IsZero) zeroMCSs[modNum] $\leftarrow$ mcs
2. else if(T.IsOne) oneMCSs[modNum] $\leftarrow$ mcs
3. else if(T.IsModul)
4. {
5. Traverse(BDDs[x], x)
6. foreach z in zeroMCSs[x] : mcs $\leftarrow$ z; Traverse(L, modNum)
7. foreach o in oneMCSs[x] : mcs $\leftarrow$ o; Traverse(H, modNum)
8. } else {
9. Traverse(L, modNum)
10. mcs $\leftarrow$ x; Traverse(H, modNum)
11. }

# Результаты поиска минимальных сечений




На графах с большим количеством связей (например, где 70 вершин) лучше справляется алгоритм с разбиением на модули.

# Список литературы


- 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