Evaluation of testing assignment for system level self-diagnosis

Viktor Mashkov[1][0000-0001-9817-3388], Jiri Fiser[1][0000-0002-5404-1727], Volodymyr Lytvynenko[2][0000-0002-1536-5542], Maria Voronenko[2][0000-0002-5392-5125]

[1]University J.E. Purkyne, Usti nad Labem, Department of Informatics, Usti nad Labem, Czech Republic,
viktor.mashkov@ujep.cz, jf@jf.cz
[2]Kherson National Technical University, Department of Informatics & Computing Technology, Kherson, Ukraine
immun56@gmail.com, mary_voronenko@i.ua

Abstract: The research concerns a task of comparing and assessing testing graphs in context of system level self-diagnosis. System level self-diagnosis aims at diagnosing systems composed of units, with the requirement that they are able to test each other by exchanging information through available links. A set of tests performed in a system can be represented as a testing graph. The obtained testing graph can be assessed based on the quality of diagnosis that it allows to achieve. In the research, we suggest the method allowing different testing graphs to be assessed and compared. The method uses the characteristic numbers. We have shown how such numbers can be computed.

Keywords: complex systems, self-diagnosis, probabilistic algorithm, decision rule

1 Introduction

Research in the area of system level self-diagnosis started in the late 60s of the last century. Since that time there has been done a great amount of research. Achievements in theoretical research gave impulse to practical implementation and enabled a broadening of the application domains of system level self-diagnosis. Initially, system level self-diagnosis was applied in complex multiprocessor systems and then it gradually spread to distributed systems, different types of networks, multi-agent systems [1, 2, 3, 4] etc. There are four main issues, which form the context of system level self-diagnosis [5].

The first issue is the system testing assignment that defines the possible set of tests among the system units. The second issue is the assumptions underlying the diagnosis algorithms. Particularly, these assumptions tackle the possible faulty sets and the test results. These assumptions have direct impact on achievable quality of diagnosis (i.e., correctness and completeness).

The third issue is the organization of test performance. It concerns the tasks of test scheduling, test repetition, test performing (random or deterministic), etc. The fourth issue relates to the problem of determining the unit(s), which will perform a diagnosis algorithm and/or will provide environment with the information about system state.

Usually, the first and second issues are considered together, and they can be viewed as theoretical ground of system level self-diagnosis. In the theory of system level self-diagnosis, three main problems were formulated. They are characterization, diagnosability and diagnosis problems [6].

The characterization problem (i.e., the problem of testing assignment) was the first one which was considered in context of system level self-diagnosis. The work of Preparata at al. [7] studied the requirements to the system testing assignment. In this research, the authors have introduced the diagnosability measure which, to a great extent, depends on the system testing assignment. Particularly, in this research, they proved that for correct diagnosis of at most *t* faulty units there must be satisfied the following two requirements: a system must have *n* units, where $n \geq 2t+1$*;* each system unit must be tested by at least *t* distinct other units. When these requirements are not satisfied correct diagnosis is not guaranteed. Nevertheless, there is a probability (sometimes great enough) that in this case it is also possible to achieve correct diagnosis. In this research, we suggest new diagnosability measure which allows comparing system testing assignments and evaluating the formed testing graphs obtained after performing a set of tests.

2 Formal problem statement

System level self-diagnosis is based on the results of tests performed by system units. System units test each other either according to predefined schedule or randomly.

One of the ways of how to express a testing assignment consists in presenting it as a testing graph. Testing graph is a basis for a model of test performance. This model can be used for simulation of tests execution and obtaining a syndrome. Syndrome is an input for a diagnosis algorithm (see Fig. 1).

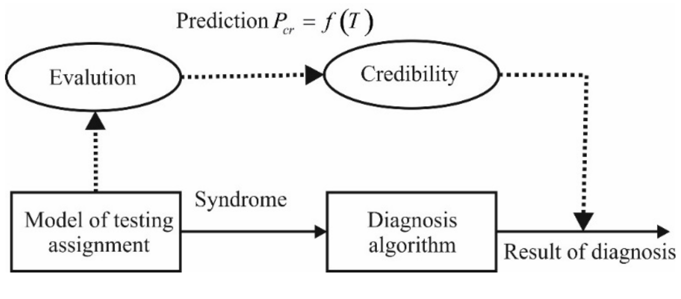

Fig.1. The role of task of evaluation of testing assignment

Different testing assignments, T (i.e., testing graphs) allow obtaining different credibility of diagnosis result (i.e., probabilities of correct diagnosis result, P_{CT}). To predict the credibility of possible diagnosis results it is needed to evaluate the available testing graph. It can be expressed as $T \Rightarrow P_{CT}$. The problem of computing a credibility of diagnosis results consists in determining the function f, where $P_{cr} = f(T)$. Characteristic numbers C_k can be used for computing f.

3 Literature review

System level self-diagnosis (SLSD) was introduced by Preparata at al. [7] and has been deeply investigated in literature. There are four main issues that form the context of SLSD [8]. The first issue is the testing assignment that defines the possible set of tests among the system units. The second issue is the assumptions underlying the diagnosis algorithms. The third issue is the organization of tests performance. The fourth issue relates to the problem of choosing the diagnostic nucleus. The first and second issues can be considered as theoretical ground of SLSD. In the theory of SLSD, three basic problems were formulated [6, 9]. These are characterization problem, diagnosability problem and diagnosis problem. The problem of testing assignment (i.e., characterization problem) was the first task that was considered in context of SLSD. The work of Preparata at al. [7] determined the requirements to the testing assignment of a system. Diagnosis problem concerns the task of determining a fault set from an allowable family, for a given testing assignment, fault model, and syndrome [8].

There have been developed many algorithms allowing to identify a fault set uniquely (when some assumptions are made about the faulty units). The main task while developing these algorithms is to reduce their complexity. Among the most efficient algorithms there could be named algorithm proposed by Dahbura and Masson [10] which has $O(N2.5)$ time complexity and $O(t3+|E|)$ algorithm suggested by Sullivan [11]. Both algorithms were developed for t-diagnosable systems under the symmetric invalidation model and when permanent fault are allowable only. There are many special classes of t-diagnosable systems that support more efficient diagnosis techniques than above mentioned ones, and this is reason to believe that an $O(|E|)$ diagnosis solution exists for all t-diagnosable systems. Preparata et al. defined the $D\delta.t$ structure in which unit u_i tests uj if and only if $j-i=\delta m(\bmod n)$, where $m=1,2,\dots,t$. Meyer and Masson [12] gave $O(nt)$) solution to the case of $\delta=1$.

In real complex systems, units are not necessarily homogeneous and can operate under different conditions. Therefore, units can have different levels of their reliability. This fact can be accounted for by assigning probabilities to the unit states. Probabilistic approach to system level self-diagnosis doesn’t deal with such problems as t-diagnosability and testing assignment. Among the first who investigated the probabilistic algorithms were H. Fujiwara and K. Kinoshita [13]. Probabilistic algorithms are based on the computing of the posterior probabilities of system unit states, upon which the decision about the system state is made.

4 Testing graph

Tests in a system can be performed:

- either in accordance with a preset schedule (i.e., defined a priori)
- or in an adapted manner when, at the beginning, the tests are performed in accordance with defined a priori testing assignment.

Once a unit is diagnosed as fault free, the tests it performs are considered reliable, and therefore, any other units should only be tested ones by this fault-free unit to correctly determine its status. Thus, the testing assignment is adapted such that units diagnosed as fault-free perform all the testing in the system [14]:

- or entirely randomly (i.e., from the beginning to the end of testing);
- or adaptively randomly.

At the beginning, all units are engaged in tests performing. Tests are performed randomly. Once a test reset takes the value of 1, the units participated in this test (so-called suspected pair) should only be tested by other system units (i.e., should not perform tests on other units). The choice of each pair of units for testing is performed randomly.

Testing graph is a convenient form for presenting the tests performed in a system.

Testing graph is a directed graph $G=(V,E)$. Each vertex $v_i = V$ of G represents unit u_i and edge $e_{ij} \in E$ represents the test which is performed by unit u_i on unit u_j.

Having obtained the testing graph, we can investigate the diagnosis properties which this graph possesses. Based on these diagnosis properties, it is possible to compare different testing graphs and find the best one from a certain criteria.

It is easy to show that diagnosis result depends on number of faulty units. Let t be the total number of faulty units that are allowable in the system. For some values of t, it is always possible to construct the testing graph, which will ensure the correct and full diagnosis whichever syndrome is obtained.

Syndrome is a set of test results. Test result is represented by binary variable r_{ij}, such that $r_{ij}=1$ if unit u_j has not passed the test, and $r_{ij}=0$ otherwise.

Correct diagnosis means that the detected faulty units are indeed faulty units. Full diagnosis means that every faulty unit is identified.

For the testing graphs that ensure unique diagnosis [5] of at most t faulty units (so called t-diagnosability) the following conditions should be satisfied [7]: each vertex of the graph should have at least t incoming edges and there should not be multiple edges.

For example, a testing graph may satisfy the requirements for t-diagnosability for $t=1$ and for $t=2$, but fails to be t-diagnosable for $t=3$. The maximum value of t for which a testing graph is t-diagnosable is denoted as t_{max}. Many t-diagnosable testing graphs (for $t \le t_{max}$) can exist. Thus, the task arises to determine those testing graphs which have the least number of edges for providing t-diagnosability.

Definition: Testing graph is t-optimal if it contains minimal number of edges and at the same time it ensures t-diagnosability.

For the value $t = t_{max}$, t-optimal testing graph could be simply named as optimal. The number of edges of t-optimal testing graph can be easily computed as follows:

$$l = tN \tag{1}$$

Thus, the number of edges of optimal testing graph is equal to:

$$l = t_{max} N = \left\lfloor \frac{N-1}{2} \right\rfloor N \tag{2}$$

Whichever testing graph that contains more than l edges is considered as redundant testing graph. It is also possible for the given value t to construct instances of testing graphs which provide certain diagnosis properties, e.g., capability to detect certain number of faulty units. For example, testing graph (see Fig. 1.a) is t-optimal for $t = 1$ since it provides unique diagnosis of any faulty unit, and it has minimal total number of edges which is needed for such diagnosis.

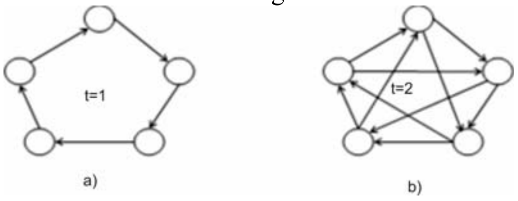

Fig. 2. Optimal testing graphs

Testing graph in Fig. 1.b has $t = 2$ and provides unique diagnosis of any two faulty units. Moreover, this graph is optimal since $t = t_{max}$.

All testing graphs depicted in Fig. 2 provide unique diagnosis of the same number of faulty units (particularly, $t = 1$) but have different total number of edges.

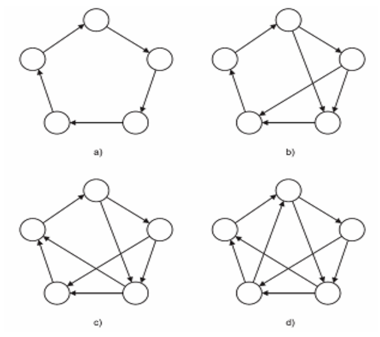

Fig. 3. Examples of the testing graphs with $t = 1$

5 Assessment of testing graphs

The method suggested by Preparata et al. [7] for assessing testing graphs does not allow comparing the testing graphs and determining more precisely their diagnosis properties. Only parameter t and its maximum value are taken into account. Besides parameter t, there also exist further criteria for evaluating diagnosis properties of testing graphs, which make it possible to assess and compare the testing graphs.
For example, to each testing graph there could be assigned the value of probability, which will reflex the fault detection property of testing graph [15, 16, 17]. It can be a probability that a syndrome (obtained after performing all tests, which are depicted in testing graph) is sufficient to detect all possible faulty units, PFD. It is only assumed that a fault-free unit is always able to detect a faulty unit (i.e., test covering is 100%). This probability can be also interpreted as probability that a system is fault-free when obtained syndrome contains only zero test results.

The probability PFD can be calculated as a sum of probabilities of the events when a fault-free units test all the other units. In the testing graph, it means that vertices which correspond to the fault-free units have edges directed to all remaining vertices.

The computation of these probabilities can be explained with help of a simple example. Let the testing graph be the one as shown in Figure 3.

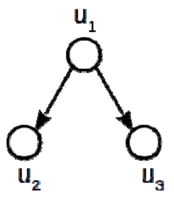

Fig. 4. Exemplary testing graph

For this testing graph there is seven $\left(=2^3-1\right)$ possible combinations of faulty-free units.

The combination without faulty-free unit occurring with probability P_U^3, where P_U is probability of unit faulty state (it is assumed that all system units have the same probability), is irrelevant because empty set of fault-free units cannot test the other system units.

More useful are three situations with one fault-free unit. In Fig. 4, this faulty-free unit is depicted by white vertex:

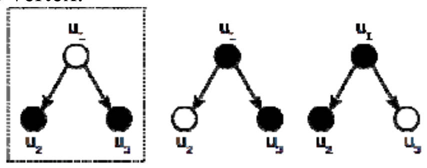

Fig. 5. Exemplary situations with one faulty free unit

Each of these situations occurs with probability $(1-P_U)P_U^2$, but only one of them corresponds the condition that fault-free units test all the other system units. Two other situations do not allow to perform correct diagnosis of system units.

The probability of all satisfactory situations with one fault-free unit is:

$$P(A_1) = (1-P_U)P_U^2 \cdot 1 \tag{3}$$

The probability of each situation with two faulty-free units is $(1-P_U)^2 P_U$. There are three such situations (see Fig. 5):

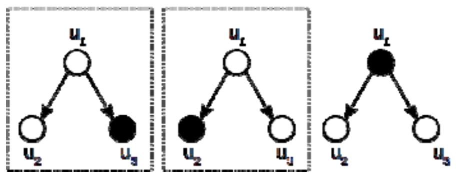

Fig. 6. Situations with two fault-free units

In the first situation, fault-free units u_1 and u_2 test the remaining unit (i.e., unit u_3). Therefore its probability $(1-P_U)^2 P_U$ should be considered as one of the summands for computing probability P_{FD}. In a similar way, the probability of the second situations (units u_1, u_3 test unit u_2) should be taken into account. The third situation (units u_2, u_3 are fault-free) does not guarantee correct diagnosis of the remaining units (and, therefore, it is not a summand for computing probability P_{FD}).

The probability of all satisfactory situations with two faulty free units is:

$$P(A_2) = (1-P_U)^2 P_U \cdot 2 \tag{4}$$

Now, there is only one situation left. Particularly, the situation when all units are faulty-free. This case also leads to correct diagnosis. The probability of this situations is $(1-P_U)^3$ and it is always included in computing probability P_{FD} regardless of testing assignment:

$$P(A_2) = (1-P_U)^2 \tag{5}$$

The total probability PFD can be generally expressed by the following sum:

$$P_{FD} = \sum_{k=1}^{N} P(A_k) = \sum_{k=1}^{N} (1-P_u)^k P_U^{N-k} C_k \tag{6}$$

where C_k is the number of options to choice the subgraph with k vertices from which all the remaining vertices are achievable (C_N is always 1).

For the considered testing graph $C_1 = 1$ (only one subgraph is satisfactory for correct diagnosis) and $C_1 = 2$ (two subgraphs are satisfactory). Therefore, the total probability is:

$$P_{FD} = (1-P_U)P_U^2 \cdot 1 + (1-P_U)^2 P \cdot 2 + (1-P_U)^2 \qquad (7)$$

For given $P_U = 0.1$, we receive $P_{FD} = 0.9$.

Let's make some changes in this testing graph. Particularly, edges from u3 to u2 and u1 are added, which makes testing graph 1-diagnosable (see Fig. 6).

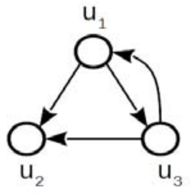

Fig. 7. Exemplary testing graph with t=1

For this graph, $C_1 = 1$ (both units u1 and u3 test all remaining units) and $C_2 = 3$ (all sets with two units test remaining unit). Therefore, for $P_U = 0.1$ we receive probability $P_{FD} = 0.9 \cdot 0.1^2 + 0.9^2 \cdot 0.1 \cdot 3 + 0.9^3 = 0.99$.

The graph in Figure 7 has six edges (every unit tests all remaining units).

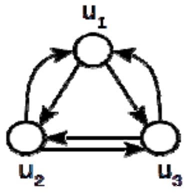

Fig. 8. Testing graph with six edges

In this graph, number C_1 is increased up to 3 (from each vertex of the testing graph now it is possible to directly reach all the remaining vertices of the graph). Clearly, number C_2 is also equal to 3. The value of P_{FD} is 0.9 · 0.12 · 3 + 0.92 · 0.1 · 3 + 0.93 = 0.999.

This graph has the same value t (particularly $t = 1$) and provides the same 1-diagnosability. However, from the point of probability PFD, it is evident that the graph in Figure 6 has better diagnosis properties $(P_{FD} = 0.999)$ than the graph in Figure 5 $(P_{FD} = 0.99)$.

For comparing the probabilities P_{FD}, we need to have the numbers C_k, $k = 1, 2, \ldots, n$ to be computed in advance. These numbers is called as characteristic numbers.

Definition: Characteristic numbers $C_k, k = 1, 2, \ldots, n$ are the numbers of choices of k vertices (resp. sub-graphs) from the testing graph so that all the remaining vertices of the graph are directly reachable.

For more complex graphs the characteristic numbers can be computed from the modified adjacency matrix of the testing graph.

Adjacency matrix of a testing graph $M = \left(a_{ij}\right)$ has the following entries:

$$a_{ij} = \begin{cases} 1 & \text{if unit } u_i \text{ tests unit } u_j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Modified adjacency matrix is derived from the adjacency matrix by way of setting the values of 1 alongside the main diagonal. An algorithm for computing the characteristics numbers for particular testing graph have to account all the combinations of choices a unit(s), from which all the remaining ones are directly reachable, according to the following expression:

$$C_k = \sum_{c \in C(k,n)} \left[\prod_{j=1}^{n} \left(a_{c_1 j} \vee a_{c_2 j} \vee \ldots \vee a_{c_k j} \right) \right] \tag{9}$$

where $C_k, C(k,n)$ is set of all k-combination i.e. all subsets of set $\{1,2,\ldots,n\}$ with k distinct element.

6 Computation of characteristic numbers

The effectiveness of calculation of probability P_{FD} for a testing assignment strongly depends on implementation of computation of characteristic number C_k for this testing assignment.

Modern programming languages make possible straightforward computation of equation 3.2. For example in Julia programming language [18], the program is almost identical to its mathematical form (where parameter a is a modified adjacency matrix).

```
function C(a::Matrix{Int}, k::Int)
  n = size(a, 1) # size in the first = second dimension
  return sum(prod(reduce(|, a[c,j]) for j = 1:n)
          for c in combinations(1:n, k))
end
```

The outer summation iterates over sequences of every k-combinations of set $1,2,\ldots,n$ (set is generated by range 1:n) produced by function combinations. The individual combination (denoted as c) is vector of integers. The computation of inner logical summation $\left(a_{c_1 j} \vee a_{c_2 j} \vee \ldots \vee a_{c_k j}\right)$ of selected items in columns is provided by discontinuous indexing (index is vector of integers) and by reduction (folding) using bitwise or operation.

This initial and naïve implementation is depicted in Figure 9 (example of computation of characteristic number C_2 for system with tree units).

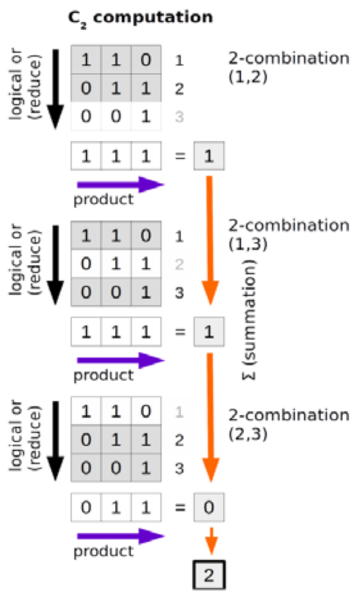

Fig. 9. Straightforward computation of character number

This implementation is compact but it has some shortcomings such as:

- effective iterator over all *k*-combination is not available in most programming languages (e.g. *Matlab* or *Java*);
- it does not use any form of parallelism (including bit-wise parallelism which is supported by all processors);
- the computation does not utilize already calculated values (outputs).

The first two shortcomings can be eliminated by iteration over k-combinations which are represented by bit patterns. These patterns are formed by string of bits containing k 1's and n-k 0's. Sequences of all k-combinations are special cases of combinatorial Gray codes [19] (generalization of commonly used and well-known Gray code) and they are realizable by elementary bitwise operations at processor level (e.g. bit complement, bitwise OR, AND and shifts).

In our implementation we use simple generator [20] which utilizes only basic bitwise operations as well as operation of counting trailing zeros. This operation is often directly and efficiently supported by current CPUs (for example, trailing_zeros function is directly translated to BSF instruction in x86-64 platform [21] by Julia programming language).

The generator of k-combinations Gray codes in Julia has a form of iterator function. For pattern v it returns pair (v,w) where w is next patterns.

```
function Base.next(c:: Combinations {Int}, v:: Int)
    t = v | (v - 1)
```

```
  w = (t + 1) | (((~t & -~t) - 1) >> ( trailing_zeros (v) + 1))
  return (v,w)

  end
```

At every step of iteration over bit patterns, only relatively simple operations are performed. The parameter pa is a modified adjacency matrix packed as array of integer values each of which corresponds to one column of the original matrix. For example sample matrix from Figure 5 is represented as array [6, 8, 5] i.e. [100, 110, 011] in binary notation. Parameter gcode is again bit pattern (integer) representing a selection of modules of diagnostic graph (the so-called subgraph).

```
function cover_test (pa:: Vector {Int64}, gcode :: Int64)
  for column in pa #iteration  over packed column
    if column & gcode == 0
      return false
    end
  end
  return true #OK subgraph covers all remaining vertices
end
```

Bitwise AND operator in this code replaces slow discontinuous indexing of the original code. The reduction by logical OR is substituted by one instruction of integer equality (operator ==). Product is performed by for each-loop which realizes short circuit evaluation (the first occurrence of zero exits loop).

This new implementation is illustrated on Figure 10 (only one of summation is depicted).

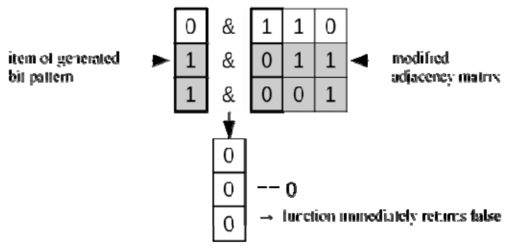
item of generated bit pattern
modified adjacency matrix
0 & 1 1 0
1 & 0 1 1
1 & 0 0 1
0
0
0
-- 0
→ function immediately returns false

Fig. 10. Fragment of suggested computation of characteristic number C_2

Further acceleration of algorithm is possible by utilization of trivial assertion: if vertices of subgraph of a testing graph cover all the remaining vertices of this testing graph then the vertices of any graph which include this subgraph (the so-called supergraph) also cover the remaining vertices.

The improved algorithm computes characteristic numbers all together concurrently i.e. it returns vector of characteristic number $\left[c_1, c_2, \ldots, c_n\right]$.

The first tested subgraph in this algorithm contains one unit and it is represented by bit pattern with 1 on the leftmost position (e.g. bit pattern 1000 for system with four unit). If this subgraph covers the remaining vertices (i.e. function cover_test returns value true) then c1 is incremented and its supergraphs containing 2 to n vertices (e.g. 1100, 1101, 1111) are taken in account in increment of values $c_2,\dots,c_n$ (number of subgraphs is determined by appropriate binomial coefficient). These supergraphs need not be consequently iterated and tested in the following phases of computation. Otherwise, the next subgraphs with two units represented by 1's on the leftmost position is tested (e.g. pattern 1100).

When all supergraphs of the first one-vertex subgraph are considered (i.e. directly tested or automatically included in bulk) the next one-vertex subgraph is tested (eg. 0100).

This implementation requires additional memory for generation of sequences of bit patterns but generated combinatorial Gray codes take into consideration partial ordering of combination by supergraph relation i.e. codes are hierarchical in some sense.

Optionally the precomputed table of binomial coefficients can be used. The asymptotic space complexity of algorithm is O(k) which is acceptable because k is relatively small (time complexity is still exponential). The code is more complex but it is still usable in small devices.

```
function Cvec(a:: Matrix {Int}, sorting :: Bool)
  pa = pack(a) # packing of matrix to vector of bit patterns (i.e integer values)
  if sorting # sort by 1's count (i.e. by outdegree of given vertex), see next paragraph
     sort!(pa , by= count_ones)
  end
  n = length (pa) # number of vertices
  binom = binomials(n) # populating of matrix of binomials coefficients
  p = zeros0(Int , n + 2)  #vector of indexes of the leftmost positions of ones (indexed
from zero)
  gc = zeros0(Int64 , n + 1) # vector of subgraph bit patterns (zero based indexing)
  c = zeros(Int , n) # vector of characteristic numbers  (one based indexing)
  p[0] = n + 1 # p[0] is only formal stop position
  k = 1 # number of vertices in subgraph (= number of 1's in patterns)
  while k > 0
    p[k] += 1 # shift leftmost 1-bit position to the right
    if p[k] == p[k -1] # if it is not possible
               p[k] = 0
      k -= 1 # return to iteration over (k-1)-vertex subgraph
   else
      gc[k] = gc[k -1] | (1 << (p[k] - 1)) # bit pattern of k-vertex subgraph

if cover_test (pa , gc[k])
        c[k] += 1 # it counts this subgraph
        for j=1:p[k]-1 # and all its super-graphs
          c[k+j] += binom[p[k] - 1, j]
        end
      else
        k += 1 # or test next subgraph
      end
    end
  end
  return c
end
```

According to the above mentioned assertion the algorithm is rather asymmetrical. The subgraphs which are represented by 1-bits positioned on the left side of a bit pattern take advantage of the assertion on greater extent (the generator fills bit strings from the leftmost bit). The (one vertex) subgraph represented by the least significant bit does not utilize assertion anymore.

The partial improvement consists of the sorting of vertices by out-degree values because the vertex with greater out-degree is better candidate for participation in subgraph covering all remaining vertices (i.e. should be placed on more significant bit of packed column).

The efficiency of algorithms is evaluated by the time required for application of algorithms on random diagnostics graphs (see Figure 11). The random testing graphs contain approximately $n/2$ edges (close to density of edges in optimal testing graphs).

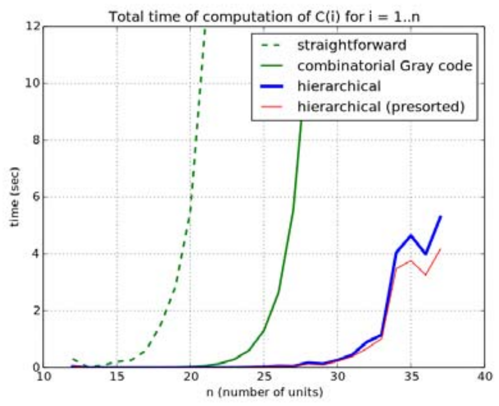

Fig. 11. Time of computation of characteristic number for different number of units

All algorithms have exponential time complexity but their application areas are different. The straightforward implementation is usable only for relatively small number of units (<17 for subseconds execution times on PC range CPU). The algorithm using bit-level parallelism is usable to greater number of units (approximately 25). The algorithm with hierarchical bit patterns makes possible computation with subseconds delays for system with 34 unit. The pre-sorting of vertices leads to some speed-up but it is in most real cases practically negligible.

The hierarchical algorithm also depends on specific structures of testing graph (details are subject to further research).

Figure 12 shows effectiveness of algorithms for almost t_{max}-optimal testing graphs i.e. or graphs with a large number of tests. For t-optimal testing graphs $(t > t_{max})$ the situation may be very different. Plot in Figure depicts the time of computation for simple t-optimal graphs for system with 25 units (t_{max} is 12) and for both algorithms based on bit patterns (straightforward implementation is useless for system with 25 units).

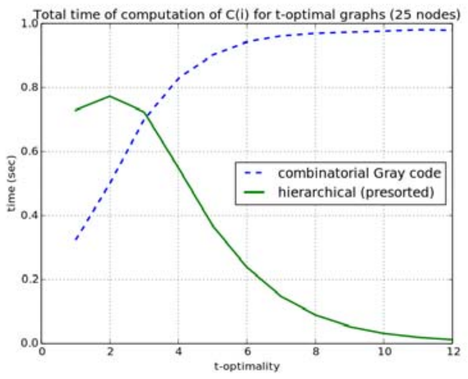

Fig. 12. Computational time for t-optimal testing graphs

For testing graphs with a small number of edges (tests) the simple (non-hierarchical) algorithm is better compared to t_{max} optimal testing graphs because iteration is simpler and short circuit evaluation of product operation is performed almost immediately (only two direct instruction are executed per subgraph). On other hand the hierarchical approach is more complicated (tens of direct instruction per subgraph) and it cannot utilize acceleration of bulk increments (small subgraphs do not cover its remaining vertices). But this acceleration outperform short-circuit for 3-optimal graph and for nearly t_{max} –optimal graphs the difference is in order of tens.

The computation of probability PFD using equation 3.1 requires floating point unit because it uses multiplication of rational numbers (computation of characteristic number is limited to integer arithmetic with simple operation without multiplication). Fortunately, the relative comparison of probabilities (i.e. evaluation of testing assignment) does not depend on probability of a unit faulty state P_U (if it is assumed that all system units have the same probability P_U). Therefore we can choose any probability PU from interval [0,1]. For probability 0.5 (although it is unrealistic) the equation 3.1 is simplified to form:

$$\sum_{i=1}^{k} \frac{C_k}{2^N} \tag{10}$$

Comparison of probabilities for concrete system requires only (integer) summation of characteristic number (value 2^N is constant). When we need to compare testing graphs with differed number of units the computation can be performed by way of bit shift operation on fixed point representation of probability values.

7 Conclusions

Tests performed in a system can be represented as a testing graph. Analysis of the obtained testing graph aims at checking whether all system units have been tested or whether the formed testing graph belongs to predefined subset of testing graphs. It depends on the value of required credibility of system self-diagnosis. Testing graph can be also used as input data for diagnosis algorithm. For different testing graphs the obtained diagnosis results will have different credibility.

In the research, we have considered the probability that all system units can be correctly diagnosed after performing all tests. This probability can be used as a diagnosability measure which will allow comparing system testing assignments and evaluating obtained testing graphs. This probability is computed by using characteristic numbers. In the research, we suggested relatively effective method for computing characteristic numbers and developed the algorithm (based on bitwise operations with integer values). Efficiency of the developed algorithm was also evaluated for different scenarios of testing assignments.

References

1. Mashkov, V.: Task allocation among agents of restricted alliance. *Proc. of IASTED ISC'2005 conference,* Cambridge, MA, USA, 2005, pp.13-18 (2005)
2. Mashkov, V.: Restricted Alliance and Coalitions Formation. Proc. of IEEE/WIC/ACM International Conference on Intelligent Agent Technology/ Beijing, China, 2004, pp.329-332 (2004)
3. Qin, L., He, X., Zhou, D.: A survey of fault diagnosis for swarm systems. Systems Science and Control Engineering. Vol. 2, 2014, pp. 13-23 (2014)
4. Mahapatro, A., Khilar, P.: Fault diagnosis in wireless sensor networks: a survey IEEE Commun Surv Tutorials. Vol. 15, 2013, pp. 2000-2026 (2013)
5. Mashkov, V., Mashkov, O.: Interpretation of diagnosis problem of system level self-diagnosis. *Int. Journal „Mathematical Modeling and Computing",* Vol.2, No.1, 2015, pp.71-76 (2015)
6. Barborak, M., Malek, M., Dahbura, A.: The consensus problem in fault-tolerant computing. *ACM Computing Surveys.* Vol.25, No.2, 1663, pp.171-220 (1663)
7. Preparata, T., Metze, G., Chien, R.: On the connection assignment problem of diagnosable system. *IEEE Transactions on Electronic Computers.* Vol.EC-16, No.12, 1967. pp. 848-854(1967)
8. Mashkov, V.: Selected problems of system level self-diagnosis. Lviv: Ukrainian Academic Press, 2011, 184 pages (2011)
9. Somani, A.: System Level Diagnosis: A Review. (1997) [online]. Available from www: <http://citeseerx.ist.psu.edu/viewdoc/ summary?doi=10.1.1.52.9488>.

10. Dahbura, A., Masson, G. :An O($n^{2.5}$) fault identification algorithm for diagnosable systems. IEEE Trans. Comput., Vol.C-33, 1984. pp.486-492 (1984)
11. Sullivan, G.: An O(t^3 + |E|) fault identification algorithm for diagnosable systems. IEEE Trans. Comput., Vol.C-37, 1988. pp.388-397 (1988)
12. Meyer, G., Masson, G.: An efficient fault diagnosis algorithm for symmetric multiple processor architectures. IEEE Trans. Comput., Vol.C-27, 1978. pp.1059-1063 (1978)
13. Fujiwara, H., Kinoshita, K.: Some existence theorems for probabilistically diagnosable systems. IEEE Trans. on Comp. Vol.C-27, No.4, 1981. pp.297-303 (1981)
14. Bianchini, R., Buskens R.: An adaptive distributed system-level diagnosis algorithm and its implementation. In the *21st International IEEE Simposium on Fault-tolerant Computing*. New York (USA), 1991, pp.222-229 (1991)
15. Mashkov, V., Barabash, O.: Self-checking and self-diagnosis of module systems on the principle of walking diagnostic kernel. Engineering Simulation, Vol.15, 1998, pp. 43-51 (1998)
16. Jarrah, H., Sarkar, N., Gutierrez, J.: Conparison-based system-level fault diagnosis protocols for obile ad-hoc networks: A survey. Journal of Network and Computer Applications. Vol. 60, 2016, pp. 68-81 (2016)
17. Weber, A., Kutzke, A., Chessa, S.: Energy-aware test connection assignment for the self-diagnosis. J Braz Comput Soc. Vol. 18, 2012, pp. 19-27 (2012)
18. Bezanson, J., Karpinski, S., Shah, V., Edelman, A.: Julia: A Fast Dynamic Language for Technical Computing. 2012, ARXIV (2012)
19. Savage, C.: A Survey of Combinatorial Gray Codes. *SIAM Review* [online]. 1997, 39(4), 605-629 (1997) [cit. 2016-04-16].
20. Anderson, S.: Bit Twiddling Hacks [online] (2005) htps://graphics.stanford.edu/~seander/bithacks.html
21. *AMD64 Architecture Programmer's Manual Volume 3: General Purpose and System Instructions3* . AMD. 2011. pp. 204–205, (2011)