Using Genetic Algorithms to Study the Effects of Topology on Spectrum Based Diagnosis

Cuiting Chen, Hans-Gerhard Gross and Andy Zaidman
Delft University of Technology, the Netherlands
e-mail: {cuiting.chen; h.g.gross; a.e.zaidman}@tudelft.nl

Abstract

Spectrum-based fault localization (SFL) is a statistical fault diagnosis technique that infers diagnoses from runtime observations. It works by monitoring system transactions, and comparing activity information with pass/fail observations. SFL requires the monitors, which recover the activity data, to be organized to produce optimal information for the diagnosis. This organization is termed topology.

Optimality of monitoring topology for diagnosability represents a search or optimization problem amenable to be addressed by meta-heuristic algorithms. In order to study the effects of topology on the production of diagnoses through SFL, we use genetic algorithms (GA) to generate topologies that lead to improved diagnosability. We illustrate how monitoring topologies affect the diagnosability of systems, and how GA can help to study these effects. We derive general characteristics of topologies to facilitate SFL-based diagnoses.

1 Introduction

Spectrum-based fault localization (SFL) is a lightweight statistics-based automatic diagnosis approach that can be applied to identify misbehaving system parts [5]. It works by automatically inferring a diagnosis from symptoms [1]. The diagnosis is a ranking of potentially faulty system components and the symptoms are observations about component involvement in system activation, plus pass/fail information for each activation [8]. The activation of the system is expressed in terms of a binary activity matrix representing for each component whether it has been involved in a transaction. The pass/fail information is expressed in terms of a binary output vector. A diagnosis is determined by calculating the similarity between each component's activation vector and the output vector. A component whose activity vector is more similar to the output vector is more likely faulty than other components, and ranked higher as suspect.

The application of SFL creates a particular challenge, i.e. the placement of the monitors for gathering component involvement information. We refer to this placement as the *monitoring topology* of the diagnosis system. In principle monitors may be placed anywhere in the monitored system. However, the places should be selected carefully to yield the best results in terms of calculating correct diagnoses. Typical places are in or around the system components, or collections of system components, or between them. Finding monitoring topologies that lead to high diagnosability represents a difficult optimization problem amenable to be solved by meta-heuristic algorithms, such as genetic algorithms. This brings us to the formulation of the following research questions:

RQ1: How can genetic algorithms be used to determine better diagnosable topologies?

RQ2: What are characteristics of topologies that are better diagnosable?

One contribution of this paper is the application of GA, including the definition of adequate fitness functions, in order to study the optimality of topologies for better diagnosability. Another contribution is the formulation of general characteristics of topologies that improve SFL-based diagnoses. Optimization of topology is a well-known problem domain to be addressed by genetic algorithms, e.g. [4], however, the use of GA in spectrum-based software fault localization is novel, in particular the formulation of the fitness introduced.

The remainder of this article is organized as follows. Section 2 introduces SFL and how it is affected by topology. Section 3 illustrates how GA can be applied for SFL topology optimization. Section 4 outlines our experiments performed, and Section 5 presents the discussion of their results, and lessons learned. Finally, Section 6 lists the related work, and Section 7 summarizes and concludes the paper and gives an outlook on future work.

2 Background and Scope

Spectrum-based fault localization calculates a diagnosis ranking of potentially faulty components from observing their activity and pass/fail outcome [8]. Activity is expressed in terms of block-hit-spectra [12], producing per transaction a binary coverage spectrum [21][23] and a verdict. Component activity and verdicts are derived through dedicated monitors. This is demonstrated in [5].

Table 1 illustrates SFL with a system made of components $C_0 - C_{10}$. It is activated with 6 transactions $t_1 - t_6$, leading to the corresponding component activations in the activity matrix. Four transactions

Table 1: Illustration of SFL

Cmp	Character counter	t_1	t_2	t_3	t_4	t_5	t_6	SC_o
	def count(string)	[Activity Matrix]						
C_0	let = dig = other = 0	1	1	1	1	1	1	0.82
C_1	string.each_char { |c|	1	1	1	1	1	1	0.82
C_2	if c===/[A-Z]/	1	1	1	1	0	1	0.89
C_3	**let += 2**	1	1	1	1	0	0	1.00
C_4	elsif c===/[a-z]/	1	1	1	1	0	1	0.89
C_5	let += 1	1	1	0	0	0	0	0.71
C_6	elsif c===/[0-9]/	1	1	1	1	0	1	0.89
C_7	dig += 1	0	1	0	1	0	0	0.71
C_8	elsif not c===/[a-zA-Z0-9]/	1	0	1	0	0	1	0.58
C_9	other += 1 }	1	0	1	0	0	1	0.58
C_{10}	return let, dig, other	1	1	1	1	1	1	0.82
	end							
	Output vector (verdicts)	1	1	1	1	0	0	

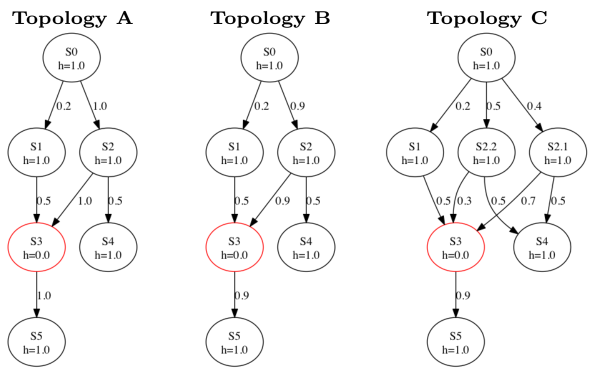

Figure 1: Different diagnosis topologies of an example service-based system

are failing, two are passing, noted in the output vector. The Ochiai similarity coefficient (SC_o) is determined for each component activation vector (a_i) and the output vector (o_i). SC_o is based on three counters $n(1,1)$, $n(1,0)$, $n(0,1)$, representing the respective numbers of occurrences that a_i and o_i form the combinations $(1,1),(1,0),(0,1)$, and it is defined by: $SC_o = n_{11}/\sqrt{(n_{11}+n_{01})\cdot(n_{11}+n_{10})}$. This denotes the likelihood of a component being faulty and determines its position in the ranking. Any SC may be used; however, the Ochiai SC has been found to work best [2]. By applying SC_o in Table 1, C_3 is correctly identified as the faulty component in this example system (faulty component marked bold in Table 1).

Table 2 illustrates how monitoring topology affects SFL. The three topologies shown are comprised of six components, $C_1 - C_6$. In Topology A and B, every component represents a monitor collecting component activation information. In Topology C, component C_3 is split to represent two monitors, i.e. $C_{3.1}$ and $C_{3.2}$. Component C_4 is faulty with health h=0.0, all other components are healthy (h=1.0). The invocation probabilities between the components are represented by the numbers noted down at the arrows. Topology A shows a particular system characteristic, with components C_1, C_3, C_4, and C_6 being tightly coupled, indicated by the 1.0 invocation probabilities. This represents a specific inhibiting factor for the calculation of the diagnosis as shown by the poor component ranking in the corresponding activity matrix. Components C_1, C_3, C_4 get attributed the same ranking, rendering this diagnosis ambiguous. Topology B represents a relaxation of the tight couplings between components C_1, C_3, C_4, and C_6 to a lower value of 0.9. This leads to a better SC_o calculation and a correct diagnosis of the faulty service for Topology B. The same effect is achieved, if a component is split up into two observation points (C3.1 and C3.2 in Topology C). The calculation of the SC_o for Topology C is also correct and unambiguous. This simple example suggests that topology has a major influence on the calculation of a diagnosis with SFL. The goal of this paper is to study these effects with the help of GA, and derive general characteristics of topologies resulting in high diagnosability.

Table 2: Activity, SC for the topologies in Fig. 1

Topology A

Service	Activity for Topology A	SC_o
S5	00000000000000000000000000000000000000000	0.000
S1	10001111010000001100001010010110000000100	0.592
S4	10011111011111001101100010101110001001000	0.742
S3	11111111111111111111111111111111111111111	1.000
S2	11111111111111111111111111111111111111111	1.000
S0	11111111111111111111111111111111111111111	1.000
Output	11111111111111111111111111111111111111111	

Topology B

Service	Activity for Topology B	SC_o
S5	00000000000000000000000000000000000000000	0.000
S1	00000000000000100001000000011000010000100	0.340
S4	00111010011000000011101010111001001010001	0.668
S0	11111111111111111111111111111111111111111	0.949
S2	11111011111110111111111111111111111111111	0.973
S3	11111011111110101111111111111101111111111	1.000
Output	11111011111110101111111111111101111111111	

Topology C

Service	Activity for Topology C	SC_o
S5	00000000000000000000000000000000000000000	0.000
S1	10100000000000000000100000010000000000000	0.344
S4	00010000000110001101110010100111000000000	0.445
S2.2	10010010000000111001011110110111001110101	0.601
S0	11111111111111111111111111111111111111111	0.689
S2.1	10011001001110011101101110101100001001100	0.851
S3	10010001000010111101101110100100011001100	1.000
Output	10010001000010111101101110100100011001100	

3 GA for Topology Optimization

We favor genetic algorithms over other optimization heuristics, since they are adequate for our problem domain [4] and easy to apply. GA represent a group of optimization techniques, loosely related to the mechanisms of natural evolution with reproduction and selection [7]. The parameters of the optimization problem are encoded as binary string (chromosome). Each chromosome represents an individual in a pool of solutions (population). During reproduction, pairs of individuals are selected for recombination and some parts of their chromosomes form a new individual. This is termed crossover and controlled by the crossover operator according to probability P_c. After recombination, individual bits of the new chromosome are mutated by a mutation operator according to a low mutation probability P_m. The resulting new individuals are assessed with the fitness function. This measures how well an individual solves the original problem. Fitter individuals have a higher chance to reproduce. This is controlled by the so-called selection operator. The fittest individuals remain in the population and build the basis for the next generation.

In SFL, the topology is represented in the activity matrix. It expresses for every observation point (monitor), whether it has been activated in a transaction or not. The coverage of the topology can be expressed as one binary string, making a mapping to a GA-chromosome straightforward. Every line in the activity matrix becomes a substring of the chromosome.

Table 3: Fitness A: high overall SC

```
# Fitness A: high overall SC
def f_high(chrom, act)
  # genotype -> phenotype transfer
  activity = Array.new
  while (a=chrom.take(act)) != [] do
    activity << a
    chrom = chrom.drop(act)
  end
  # SC calculation
  sc = Array.new
  activity.each do |output_vec|
    activity.each do |activity_vec|
      sc<<ochiai(activity_vec, output_vec)
    end
  end
  # fitness: sum up sc values
  fitness = sc.inject{|sum,x| sum + x}
  return fitness
end
```

The fitness distinguishes good from poor solutions, and it represents the adequacy of a topology to support the calculation of a diagnosis. Diagnosability can be expressed in terms of the extent to which all diagnoses carried out on an activity matrix coming from that topology, are correct diagnoses. In other words, if a topology is organized such that every faulty component can be identified correctly, the topology may be referred to as highly diagnosable. This can be achieved by consecutively setting all components used in the activity matrix to be faulty, and then calculating the similarity coefficient for each fault scenario. This yields a value representing how well a topology facilitates the discovery of faults in components. Topologies leading to higher fitness values will lead to better pinpointing of all faulty components.

The ruby-method `f_high` (Fitness A in Table 3) represents the basic fitness function yielding high overall SC. First, in the so-called *genotype-phenotype transfer*, the GA chromosome is translated into the problem domain, i.e. the binary gene-string is transformed into a binary activity matrix. Second, each component activation vector is set to be the output vector, and the SC is calculated. Third, the SC values are summed up.

4 Experiments

We performed a number of experiments in order to have GA generate highly diagnosable topologies, and then to derive general characteristics for diagnosable topologies. The genetic algorithm used for these experiments can be downloaded.[1] It uses the following rudimentary operators.

Two individuals are selected for recombination based on tournament selection [17]. This chooses N_t individuals from the population randomly, and returns the fittest in this tournament. The actual recombination is done according to the uniform crossover operator [22]. It determines for every bit in the chromosome, according to a probability P_c, whether the value for the new individual (offspring) is taken from the first or from the second parent.

The other GA-parameters depend on the complexity of the particular problem size to be solved. The population size N_p, and the tournament size N_t are set to different values in the different experiments, reflecting the chromosome size of the respective problem, i.e. according to the size of the activity matrix (or based on experience). Bigger activity matrices represent larger search spaces and require bigger populations for better sampling of the search space. Experiments with large topologies are possible but would require more space for presentation. Therefore, the topologies shown are limited to five components. Experiments with larger numbers of components yield similar results. The GA maintains and evolves the N_p fittest individuals. Crossover probability P_c is set to 0.5 in all experiments, and mutation probability P_m is set to a low value of 0.001. These were determined through initial experiments and found to provide acceptable results. Every experiment was repeated 20 times. There may be better GA implementations or operators to chose from, however, the ones introduced here are sufficient to produce usable results.

Assessing the Setup.

The first experiments performed serve as assessment in terms of whether or to which extent the GA is able to generate highly diagnosable activity matrices. We assume a diagnosable topology is represented by high overall SC values. This can be tested by iteratively setting the output vector in the fitness function equal to each component's activation vector (Fitness A in Table 3). Each component is set to be faulty in the calculation of the SC (single fault case), resulting in $SC_o = 1$ for this comparison, and we expect the GA to produce activity matrices in which all component activations are alike. An example is shown in Table 4, above. The first activity matrix (fitness=16.75) represents the best random individual from the first generation. The second activity matrix (fitness=24.88) represents the fittest individual after 200 generations. The success of this optimization example is quite obvious. All component activity vectors are highly similar, representing a highly diagnosable activity matrix expressed by the calculation of high overall SC. In fact, the most optimal solution in this example is fitness=25, when all combinations of component activity vector and output vector yield a 1.0 as SC value, i.e., when they are identical. In this example, the fittest individual is only 1 bit flip away from the optimal solution, i.e. in the penultimate spectrum of C_1.

Even though, this experiment is successful in terms of assessing our experimental setup, it is useless in diagnosis, because the activity matrix represents a topology in which all components are tightly coupled. If C_1 is invoked, all other components will also always be invoked, leading to components C_1 to C_5 being assigned the same ranking (SC = 1.0; compare with Topology A in Table 2), and resulting in an ambiguous diagnosis. As a consequence, we have to extend the adequacy criterion for topologies: "A topology is diagnosable, if it facilitates the detection of all faults in a system, *and their unambiguous identification*," i.e. it must not generate duplicate top SC_o.

Topologies for Discriminable Diagnoses.

In this experiment, the fitness function from the previous setup is adjusted to award topologies higher fitness, which result in high overall SC, but also lead to *discriminable* diagnoses, thereby addressing ambiguity. The fitness function `f_discrim` (Fitness B in Table 5) illus-

[1]https://github.com/SERG-Delft/rusiga

Table 4: Assessment of the experimental setup

100 Generations, 40 Activations N_p=120, N_t=6, P_c=0.5, P_m=0.001	
	best random individual (fitness=16.75)
C_1	1010101001111001011111000110101110100100
C_2	0011101100110000101101101010110011110101
C_3	1001010011011111111000011110110001011101 1
C_4	0101101110011001100101101011100010110001
C_5	0101111111001001010101001110101010101001
	best final individual (fitness=24.88)
C_1	0111101110111111101111101111111111011010 1
C_2	0111101110111111101111101111111110110111
C_3	0111101110111111101111101111111110110111
C_4	0111101110111111101111101111111110110111
C_5	0111101110111111101111101111111110110111

trates this extension. It awards individuals that lead to one top ranked component, and a number of lower-ranked components. Moreover, it can be configured to minimize (`diff=:low`) or maximize (`diff=:high`) the difference between the top ranked and all lower-ranked components. Table 6 shows examples for both optimization goals.

Table 5: Fitness B: discriminable SC

```
# Fitness B: discriminable SC
def f_discrim(chrom, act, diff=:high)
  # genotype -> phenotype transfer
  # same as f_high()
  ...
  # SC calculation
  # same as f_high()
  ...
  # fitness: discriminiable SC
  highest_sc = (sc.sort!)[-1]
  pivot = sc.find_index(highest_sc)
  low_sc = sc[0..pivot-1]
  top_sc = sc[pivot..-1]
  sum_top = top_sc.inject {|sum,x| sum+x}
  sum_low = low_sc.inject {|sum,x| sum+x}
  return sum_top - sum_low if diff==:high
  return sum_low - sum_top if diff==:low
end
```

Adjusting `diff` to `:high` leads to a large number of '0's in the final activity matrix compared to a random activity matrix from the early generations, representing a lot of unique component activation. This means that discriminable diagnoses, indeed, can be supported by the topology of the system, and that inactivity of the components, indicated through the many zeroes, supports this. In other words, high diagnosability can be achieved through inactivity observations, or through activation of components in isolation, which is the opposite of tight component coupling. This is an interesting result, because for the topology it means, that having components which may be activated individually rather than in combination with other components, helps separating system executions, and thus, improves the diagnosability of the system. This comes from how the SC_o calculates similarity. Completely inactive spectra are ignored by the SC_o, but spectra with fewer activations provide more useful information for SFL than spectra with more activations. For example, a spectrum with $a_i = [0, 0, 0, 0, 1]$ is more useful than another one with $a_j = [1, 1, 1, 1, 0]$, because if the transaction a_i fails, this will result in the only one activated component in a_i being blamed more. This outcome may seem like "the bleeding obvious," but, because complete decoupling of all components is not realistic in real systems, in the future, we will have to assess whether or to which extent a GA may be able to generate optimal monitoring locations that help to exploit this property, at least to a certain extent.

Table 6: Examples for discriminable diagnoses

30 Generations, 40 Activations N_p=50, N_t=3, P_c=0.5, P_m=0.001; **diff=:high**	
	fitness=-2.98 (best random individual)
C_1	1110100010001000101100111001110100100011
C_2	1000101000100100100110100000000011001110
C_3	1111010000100101100001000100100101110000
C_4	1001001111110111100111110111011000010100
C_5	0100110111000010010110110011010000101001
	fitness=4.606 (best individual after 30 gen.)
C_1	0000100000000100000000000011000000011100010
C_2	0110000000000010000000101000010001000011 00
C_3	0000000001000010001000000001000000000000 1
C_4	0001001100100001000100000000000000000010000
C_5	1100010110010000110001010000111000000000 0
30 Generations, 40 Activations N_p=50, N_t=3, P_c=0.5, P_m=0.001, **diff=:low**	
	fitness=7.092 (best random individual)
C_1	1001111101010000100011101111100100011101
C_2	1011011000010110111100111101110110011010
C_3	1100111001010110101100101111111010101110
C_4	0101101110001000101010101101101001110011
C_5	1010001001000110101100011001111101101011
	fitness=12.978 (best individual after 30 gen.)
C_1	1011111001011111100010111111100111111111
C_2	1111111011011111110111011111111001111111011
C_3	1111111011011111110100011111111111111011
C_4	0111101011011100101010101111111100101111111
C_5	1111100011011111110100011111111001111111011

Setting `diff` to `:low` shows different results. Even though the activity matrix contains many '1's, indicating tight coupling between the components, conclusive diagnoses can be calculated, if the topology can provide just enough discriminative information, e.g. some '0's in some spectra. Looking only at the failing spectra in which each component was activated, would lead to ambiguous diagnoses (comparable with Topology A in Table 2). Because there is slight variation in other spectra to compensate for the tight coupling, the information contained in the activity matrix is just diverse enough in order for the diagnosis algorithm to come up with an unambiguous ranking. An increase in observation diversity can be achieved by adding observation points. One approach could be the inclusion of observations representing the invocation links between the components. Another approach is the instrumentation of the components themselves in order to acquire more diverse observations. This second approach has been demonstrated to improve diagnosis considerably for service-based systems [6]. In any case, both approaches also raise the question of the optimal number of observation points for high diagnosability w.r.t. low monitoring overhead, to be addressed in future work.

Topologies for Intermittent Fault Behavior.
In the previous experiments, activation of a faulty component always lead to a failure. Here, we would like to assess to which extent topology influences the quality of the diagnosis when components exhibit intermittent fault behavior. Intermittency, i.e. a component fails occasionally, is quite common in software, and it is not attributable to random faults (as in hardware). Even though, software exhibits deterministic fault behavior, intermittency comes from the mismatch between the monitoring granularity and the activation granularity (basic block level). Hence, intermittency presents a monitoring topology issue.

Fitness function `f_randinterm` (Fitness C in Table 7) realizes intermittency through removing all '1's from each output vector except for a number of randomly chosen ones (e.g. 3 random failure observations). This

Table 7: Fitness C: random intermittency and Fitness D: constant intermittency

```
# Fitness C: random intermittency
def f_randinterm(chrom, activ, diff=:high)
  # genotype -> phenotype transfer
  # same as f_high()
  ...
  # SC calculation
  sc = Array.new
  activity.each do |output_vec|
    output_vec.remove_all_ones_except_rand(3)
    activity.each do |activity_vec|
      sc<<ochiai(activity_vec,output_vec)
    end
  end
  # fitness: discriminable
  # same as f_discrim()
  ...
end
# Fitness D: constant intermittency
def f_constinterm(chrom, activ, diff=:high)
  # genotype -> phenotype transfer
  # same as f_high()
  ...
  # SC calculation with const. output vector
  output_vec = [0,0,0,1,0,0,0,0,0,1,0,0,0,...]
  activity.each do |activity_vec|
    sc<<ochiai(activity_vec,output_vec)
  end
  # fitness: discriminable
  # same as f_discrim()
  ...
end
```

yields similar results as presented in Table 6, with `diff` set to `:high` and `:low`, respectively, so we omitted an example. Consecutively using each activation vector as output vector, leads the optimization to be focused only on the generation of high/low differences between top ranked activations and the lower ranked activations, thereby ignoring the intermittency target.

Amending the fitness function by focusing on only one faulty component, leads to a more differentiated outcome (through Fitness D, in Table 7). Table 8 shows two examples with five constant failures seeded into the output vector, and with `diff` set to `:high` and `:low`, respectively. Looking at the two examples, the solution of the GA to the intermittency problem is both cunning and ironic: *"in an optimal topology, faulty components should only be executed when they are guaranteed to fail,"* which avoids intermittency altogether and is not very useful. Further, when `diff` is set to `:high`, it becomes apparent that when the failing component, C_5 in this example, is activated, none of the other components is activated, suggesting again, that the ability to activate components in isolation is advantageous. And when `diff` is set to `:low`, ambiguous diagnoses can be resolved through additional observations, i.e. through the very few additional '1's in the bottom activity matrix. This confirms our previous observations. Intermittency cannot be addressed with this kind of experiment.

Freely Evolved Topologies.

Up to this point, we have had the GA evolve topologies based on a predefined output vector with seeded faults. That way, we could define the interesting error scenarios, and have the GA generate optimal activity matrices. In this experiment, we let the GA not only evolve the activity matrices, but also their corresponding output vectors. It means, we have no control over the number of failure observations generated in the output vector, and we cannot tell whether the diagnosis is correct, because we cannot seed any particular faults. For these experiments, a 6th component is added to the GA chromosome representing the output vector, and Fitness E in Table 9 is used for evaluation of the individuals. The fitness function is slightly different compared to the earlier ones, because of the output vector taking part in the evolution. Setting `diff` to `:low` results in a selective pressure favoring many failure observations to be produced as shown in the example activity matrix on the top right hand side of Table 10. Because the number of failing transactions is unrealistically high for real software systems, we set `diff` to `:high`, resulting in much lower number of failure observations. This is shown in the example activity matrix on the bottom right hand side of Table 10.

Table 8: Examples for fault intermittency

	200 Generations, 40 Activations N_p=200, N_t=3, P_c=0.5, P_m=0.001; **diff=:high**	
	fitness=0.377 (best random individual)	
C_1	0000110011100110100000001000101100110001	
C_2	0001011111000100001001010111011110101111	
C_3	1100101010101110110101001101000010110001	
C_4	1001100110111100101100010000011011111110	
C_5	1010010010110111110111000010011001111001	
O	0111000000000000000000000000000000000110	
	fitness=1.0 (best individual after 200 gen.)	SC_o
C_1	0000101111000000110100111011111000110000	0.00
C_2	1000010101010001101001000111110100010000	0.00
C_3	0000010011111011011110100011000110110001	0.00
C_4	0000111111110010011000101110000001111001	0.00
C_5	0111000000000000000000000000000000000110	1.00
O	0111000000000000000000000000000000000110	
	200 Generations, 40 Activations N_p=200, N_t=3, P_c=0.5, P_m=0.001; **diff=:low**	
	fitness=1.105 (best random individual)	
C_1	1111110000110000100011111100010011110010	
C_2	1111100100010001000111100001100000010011	
C_3	0111100111101010101101000110000011110101	
C_4	0111110100100011101000111000111100000101	
C_5	0111010010010100100100101010001111110110	
O	0111000000000000000000000000000000000110	
	fitness=2.652 (best individual after 200 gen.)	SC_o
C_1	0111000000001000000000000000000000000110	0.91
C_2	0111100000000000000000000000000000000110	0.91
C_3	0111000001000000000000000000000000000110	0.91
C_4	0111000000000000000000000000000000000110	1.00
C_5	0111000010000000000000000000000000000110	0.91
O	0111000000000000000000000000000000000110	

Table 9: Fitness E: Freely evolved topologies

```
# Fitness E: freely evolved
# with output vector
def f_discrout(chrom, act, diff=:low)
  # genotype -> phenotype transfer
  # same as f_discrim()
  ...
  # SC calculation
  # ouput -> last comp act vector
  output = activity_matrix[-1]
  activity_matrix.delete_at(-1)
  sc = Array.new
  activity_matrix.each do |activ|
    sc << ochiai(activ, output)
  end
  # fitness: discriminable SC
  top_sc  = (sc.sort!)[-1]
  top_cnt = sc.count(top_sc)
  low_sc  = sc[0..-2]
  sum_low = low_sc.inject {|sum,x| sum+x}
  return (top_sc - sum_low) / top_cnt if diff==:low
  return (sum_low - top_sc) / top_cnt if diff==:high
  #return (sum_low - top_sc) / (top_cnt + output.count(1))
  # favor. low num. of failures
end
```

Two noteworthy results can be observed in this second case. First, as noted earlier, being able to activate components individually supports the diagnosis. Second, intermittency can be dealt with. The faulty

Table 10: Examples for Freely evolved topologies

	200 Generations, 40 Activations N_p=200, N_t=3, P_c=0.5, P_m=0.001; **diff=:low**	
C_1	1111111111111111111111111111111101111111	0.97
C_2	1011111111111111110111111111111111111111	0.98
C_3	1011111111111111111111111111011111111111	0.98
C_4	1011111111111111111111111111111111111111	1.00
C_5	1011111111111111111111111111111101111111	0.98
O	1011111111111111111111111111111111111111	
	200 Generations, 40 Activations N_p=200, N_t=3, P_c=0.5, P_m=0.001; **diff=:high**	
C_1	1100011100000000110010101100010101010010000	0.00
C_2	0100000001101000000001000000101011100000001	0.00
C_3	0101100001000100001000100010010001100000	0.80
C_4	0111000000100001001000000100010000010010111	0.00
C_5	1000001001100001110100001110001010011000	0.00
O	0000100000000001000010001000000010001100000	

Table 11: Freely evolved topologies, fewer faults

	200 Generations, 40 Activations N_p=100, N_t=3, P_c=0.5, P_m=0.001; **diff=:low**	
	fitness=0.081 (best random individual)	
C_1	0011010100111001111111111001010001000111	
C_2	0011110101001011000100111010010101110111	
C_3	0011111100001100111100001110000100110111	
C_4	0110010010001111010000011001100000100111	
C_5	0110101101000001000100110110110001111001	
O	0010111000110111000000100101010001101111	
	fitness=0.44 (best indiv. after 200 gen.)	SC_o
C_1	0010100100100100000000110000010000001100000	0.316
C_2	0000111010101100001100100000001000000100000	0.289
C_3	0000101000010011011100000000000000100000110	0.302
C_4	0010100001100000000000000111000000001100011	0.302
C_5	0110100010000001000100100011000000000100010	0.302
O	0000100000000000000000000000000000000000000	

component, C_3 in this example, is sometimes invoked without resulting in a failed transaction. This leads to the $SC_o < 1.0$. In fact, when C_3 is invoked in isolation, it fails, if it is invoked in combination with any of the other components it passes. This is a clear indicator of a missing observation point, either in the component C_3 itself, or in one of its peers which is invoking it individually. Translated to a real system it means, that if $C3$ is invoked from an external (unmonitored) system, it is exercising a different internal route, than if it was called from one of its monitored peers. In future work, we will assess to which extent missing observation points can inhibit proper diagnosis.

Freely Evolved Topologies with Fewer Failures. Even when `diff` is set to `:low`, we can get useful results, when the fitness function favors individuals with low number of failures. We can amend the Fitness E shown in Table 9 by adding a scaling factor, i.e. `output.count(1)`, shown in the last line of the fitness function's code, penalizing individuals with high numbers of failures. Table 11 shows an example with a an activity matrix containing only one failure.

Remarkable, again, is the high number of zeroes indicating that monitoring of inactivity is advantageous for high diagnosability. This is in line with our earlier results. However, the result shown in Table 11 also hints to another interesting topological issue. Even though, all components are activated together in case of the failing transaction, representing tight interaction of the components in the fault case, the activity matrix contains enough discriminative information in order to reach an unambiguous diagnosis. This is, again, a strong indicator that the ability of a monitoring topology to observe various combinations or patterns of component invocations will help in reaching unambiguous diagnoses. The other occasional activations lead to sufficiently diverse information in order to being able to separate the tight coupling of the components on failure.

From this observation, we can deduce that not only diverse coverage benefits diagnosability, but moreover, also distinct coverage. In other words, topologies with more diverse execution routes, covering distinct components, facilitate diagnosability. In the topology, this can be achieved through monitoring not only activation or non-activation of a particular entity, i.e. the fact that something has been used, but also through monitoring the context in which something has been used, i.e. incoming and outgoing combinations of activations. In other words, the fact that something has been covered through various routes, or in particular sequences, which can be monitored, has an influence on the diagnosability of a topology. In the future, we will take a closer look at the influence of the traces leading to an activity matrix, rather than merely the activations themselves.

5 Discussion

5.1 Revisiting the Research Questions

In the introduction, we asked ourselves *how genetic algorithms can be used to study the effects of the monitoring topology on the diagnosability of systems.* We will now address this problem by providing answers to the two research questions formulated:

RQ1: How can search-heuristics such as genetic algorithms be used to determine better diagnosable topologies? The topology of a system is represented by an activity matrix, whereby, for each observation point, it expresses whether that point has been activated. The coverage of all observation points can be expressed as a binary string, making a mapping to a GA-chromosome straightforward. For the fitness, we propose several approaches. First, a function that expresses the diagnosability of a monitoring topology, i.e., the extent to which all diagnoses carried out on an activity matrix coming from that topology, are correct diagnoses. In the fitness function, each component is set to be faulty per diagnosis. Then, the fitness function calculates to which extent all similarity coefficients combined from all runs represent correct and distinguishable diagnoses. This yields a value representing how well a topology facilitates the discovery of each potential fault in every component. This basic fitness function can be amend in order to address the different optimization criteria required in the different experiments, e.g. favor high or low differences in the SC_o, or favor output vectors with low number of failures.

RQ2: What are characteristics of topologies that are better diagnosable? According to the fitness function, a topology is a "good" or a diagnosable topology, if it facilitates the detection of all faults in a system in an unambiguous manner. The application of GA hints at a number of routes to satisfying this fitness goal. Our results show that

- being able to invoke components in isolation is beneficial for diagnosability, because it helps separate component involvement in system executions better.

- adding observation points (monitors) in the system, and including the monitoring of inactivity, helps separating system executions, which also facilitates the diagnosability of the system.
- including monitoring of the system context (external components from other systems, incoming and outgoing activations) can support diagnosability through incorporating different invocation routes.
- including tracing information which represents combinations or distinct patterns of component coverage, may suport SFL-based diagnosis.

All these items also raise the question of the optimal number of observation points for high diagnosability w.r.t. low monitoring overhead.

5.2 Lessons Learned

Besides the more general characteristics of diagnosable topologies stated above, the application of GA taught us a lot about the behavior of the SFL approach. It is interesting to see how a search heuristic cannot only help to provide solutions, but also point to issues, both known, and unknown.

In the initial assessment of our setup, the GA generated topologies with tightly coupled components. This was due to our poor fitness definition. We knew already that tight component interaction is bad for diagnosability of a topology, and the GA was, in fact, pointing to this issue, so that in subsequent experiments, the fitness function could be adjusted.

The fact that having fewer activations within a spectrum provides better information for SFL than more activations was not obvious initially. Creating monitoring topologies that lead to such observations is, therefore, an essential goal for future work.

Finally, from the last experiments we can deduce that the context of activity is an important factor in the calculation of a diagnosis. In other words, if a component is activated, which route did this activation take? We knew already that introducing more information into the calculation of the SC yields better diagnoses. But this points to very particular information to be included, i.e., the activation paths through the system. In future work, we will derive the activation sequences from the traces generated by the monitors and encode this in the activity matrix.

5.3 Threats to Validity

In this initial application of GA to studying the effects of topologies on diagnosability, we have used activity matrices instead of real topologies. An activity matrix represents component involvement in system transactions and must be regarded as a simplification of a topology. It does not explicitly express the links between components. We can, therefore, only infer very general characteristics of potentially diagnosable topologies.

In the experiments, we have only looked at a low number of observation points (monitors), and at a low number of observations (spectra). We are aware of the fact that the number of observations and observation points affect the achievable results, but we decided to treat the generation of variable numbers of observation points as a problem in its own right, to be addressed in the future.

6 Related Work

Literature describing the application of genetic algorithms to the optimization of topologies is abundant. For instance, Kumar et al. [14] propose a general approach based on GA to design network topologies for distributed systems, in order to achieve network reliability; Madeira et al. [16] develop a computational model to optimize topologies of linear elastic structures with GA; the authors of [9] use GA to optimize the topology of hardware circuit against parallel flows.

In software engineering, the authors of [10], [19], and [20] propose to apply multi-objective GA to automatically synthesize software architectures. The architectural patterns are used for mutations and the quality metrics are used as fitness function to assess each architecture. Their research results conclude that their approach of architecture synthesis based on GA is able to produce a set of reasonable architectural solutions. However, only two quality attributes, i.e. modifiability and efficiency, were considered in their approach to generate software architectures. Lutz [15] use metaheuristics to evolve good hierarchical decompositions. Decomposition is related to our problem of placing monitors at strategically optimal locations.

Harman [11] states that "metrics are fitness functions too". We acknowledge this by defining fitness functions for diagnosability. Kim and Park [13] propose the application of reinforcement learning in self-managing systems. In particular, they mention software architecture. Our approaches are intended to contribute to self-adaptive and self-managing systems.

Piel et al. [18] apply spectrum-based fault localization techniques together with online monitoring to recover health information and pinpoint problematic components for self-adaptive systems. Abreu et. al. [3] present a diagnosis approach combining spectrum-based fault localization and model-based diagnosis techniques, which is able to locate multiple faulty components with relatively low cost.

7 Summary, Conclusions and Future Work

In this paper, we outlined how genetic algorithms can be used to study the effects of monitoring topologies on SFL-based diagnoses. We defined a simple one-to-one mapping between the chromosome of a genetic algorithm and an activity matrix to be used by SFL, plus several fitness functions representing diagnosability. Activity matrices were used as simplifying models for real topologies. Explorative experiments revealed a number of general characteristics of topologies that support diagnosability, and we learned to better understand how topology affects the calculation of diagnoses.

The vision of our research is that, eventually, we would like to be able to have a search heuristic generate the most optimal monitoring topology in terms of high diagnosability for any arbitrary existing system. In the future, therefore, we will have to look at how real topologies can be encoded for GA, instead of merely using activity matrices representing topologies. This can be done either with the help of a topology simulator[2], or with real systems. Other issues to be addressed in the future are the inclusion of context information (derived from traces) in the calculation of the diagnosis, and the inclusion of more monitors. This last aspect represents a multi-objective optimization problem in its own right, i.e. generate topologies for optimal diagnosability with minimal monitoring overhead.

Acknowledgements: Thanks to NWO for sponsorship, and our industrial partners Adyen and Exact.

References

[1] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009.

[2] R. Abreu, P. Zoeteweij, and A. J. van Gemund. An evaluation of similarity coefficients for software fault localization. In *Proc. Int'l Symp. on Dependable Computing (PRDC)*, pages 39–46. IEEE, 2006.

[3] R. Abreu, P. Zoeteweij, and A. J. van Gemund. Spectrum-based multiple fault localization. In *Proc. Int'l Conf. on Automated Software Engineering (ASE)*, pages 88–99. IEEE, 2009.

[4] C. Chapman, K. Saitou, and M. Jakiela. Genetic algorithms as an approach to configuration and topology design. *Mech. Des.*, 116(4):1005–1012, December 1994.

[5] C. Chen, H.-G. Gross, and A. Zaidman. Spectrum-based fault diagnosis for service-oriented software systems. In *Proc. of the Int'l Conf. on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2012.

[6] C. Chen, H.-G. Gross, and A. Zaidman. Improving service diagnosis through increased monitoring granularity. In *7th Intl Conf. on Software Security and Reliability*, page to appear, Washington, DC, June, 18–20 2013.

[7] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison Wesley, 1989.

[8] A. Gonzalez-Sanchez, R. Abreu, H.-G. Gross, and A. J. van Gemund. Spectrum-based sequential diagnosis. In *Proc. Int'l Conf. on Artificial Intelligence (AAAI)*, pages 189–196. AAAI Press, 2011.

[9] G. Granelli, M. Montagna, F. Zanellini, P. Bresesti, and R. Vailati. A genetic algorithm-based procedure to optimize system topology against parallel flows. *Power Systems, IEEE Transactions on*, 21(1):333–340, 2006.

[10] Hadaytullah, S. Vathsavayi, O. Raiha, and K. Koskimies. Tool support for software architecture design with genetic algorithms. In *Proc. International Conference on Software Engineering Advances (ICSEA)*, pages 359–366. IEEE CS, 2010.

[2]https://github.com/SERG-Delft/sfl-simulator

[11] M. Harman and J. A. Clark. Metrics are fitness functions too. In *Proc. of the Int'l Symp. on Software Metrics (METRICS)*, pages 58–69. IEEE, 2004.

[12] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi. An empirical investigation of program spectra. In *Proc. SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (PASTE)*, pages 83–90. ACM, 1998.

[13] D. Kim and S. Park. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In *Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, volume 0, pages 76–85. IEEE, 2009.

[14] A. Kumar, R. M. Pathak, Y. P. Gupta, and H. R. Parsaei. A genetic algorithm for distributed system topology design. *Comput. Ind. Eng.*, 28(3):659–670, 1995.

[15] R. Lutz. Evolving good hierarchical decompositions of complex systems. *Journal of Systems Architecture*, 47(7):613–634, July 2001.

[16] J. A. Madeira, H. Rodrigues, and H. Pina. Multi-objective optimization of structures topology by genetic algorithms. *Advances in Engineering Software*, 36(1):21–28, 2005.

[17] B. Miller and D. Goldberg. Genetic algorithms, tournament selection and the effects of noise. Technical Report 95006, IlliGAL Report, Dept. General Engineering, University of Illinois at Urbana Campaign, July 1995.

[18] E. Piel, A. Gonzalez-Sanchez, H. Gross, and A. van Gemund. Spectrum-based health monitoring for self-adaptive systems. In *Proc. Int'l Conf. Self-Adaptive and Self-Organizing Systems (SASO)*, pages 99–108. IEEE, 2011.

[19] O. Räihä, K. Koskimies, and E. Mäkinen. Genetic synthesis of software architecture. In *Proc. of the International Conference on Simulated Evolution and Learning (SEAL)*, volume 5361 of *LNCS*, pages 565–574. Springer, 2008.

[20] O. Räihä, K. Koskimies, and E. Mäkinen. Generating software architecture spectrum with multi-objective genetic algorithms. In *Third World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 29–36. IEEE, 2011.

[21] T. Reps, T. Ball, M. Das, and J. Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. In *European Softw. Engineering Conf. & Symp. on Foundations of Softw. Engineering (ESEC/FSE)*, volume 1301 of *LNCS*, pages 432–449. Springer, 1997.

[22] G. Syswerda. Uniform crossover in genetic algorithms. In *Third International Conference on Genetic Algorithms*, pages 2–9, 1989.

[23] P. Zoeteweij, R. Abreu, R. Golsteijn, and A. J. van Gemund. Diagnosis of embedded software using program spectra. In *Proc. Int'l Conf. and Workshops on Engineering of Computer-Based Systems (ECBS)*, pages 213–220. IEEE, 2007.