Sensitivity Analysis of Spectrum-based Fault Localisation for Multi-Agent Systems

Lúcio S. Passos[1] and **Rui Abreu**[1] and **Rosaldo J. F. Rossetti**[1]
[1]University of Porto, Porto, Portugal
e-mail: {pro09026*, rossetti}@fe.up.pt, rui@computer.org

Abstract

Diagnosing unwanted behaviour in Multi-Agent Systems (MASs) is a crucial task to assure the correct operation of a system. A light-weight technique inspired by the software-engineering-oriented techniques, the we have coined Extended Spectrum-based Fault Localisation for Multi-Agent Systems (ESFL-MAS) can be used to shorten the diagnose cycle by reducing the testing effort. As the technique relies on minimal information about the system, its diagnostic accuracy is inherently limited. In this paper, we study the impact of different similarity coefficients that are applied to the system execution spectra as we try to assess the correct operation of an agent-based application. The studied coefficients are proposed in the literature and we study the impact of them on the accuracy of spectrum-based fault localisation applied to multi-agent systems. Our experimental evaluation shows that three out of 42 (Jaccard, Ochiai, and Sorensen-Dice) have the most effective and stable performance throughout the variation of the number of passed and failed events.

1 Introduction

Multi-agent systems (MASs) are built of autonomous entities that, ideally, are inherently robust due to their "loosely coupled" feature that characterises these type of systems. However, they are susceptible to unforeseen situations, which may cause unexpected and, most of the times, undesirable behaviours. Such unexpected behaviour may affect the system's availability and reliability at the operational level. From this fact arises the necessity to endow multi-agent systems with fault-tolerant characteristics. A crucial phase in treating faults in any system is the process of recognising and localising an unexpected event, known as diagnosis.

Current advances of fault-tolerance schemes applied to MASs offer a limited level of fault diagnosis when dealing with an agent-based system able to change its overall behaviour over time. Although promising results have been registered in the literature, most of the suggested approaches are based on some *a priori* knowledge or information about the system. This condition cannot always be ensured in real-world deployment of complex MASs because of the dynamism of the environment, the presence of legacy systems, and so forth. Additionally, there are a few techniques that test the MAS as a system running at the target operating environment [1]. We, in this first stage, focus on developing a technique that diagnoses behavioural faults in order to be applied in the *testing-system level*.

The similar drawback of relying on *a priori* information to detect and diagnose faults is being addressed by the software-engineering community aiming to shorten the test-diagnose-repair cycle in many domains. Spectrum-based fault localisation (SFL) is a promising technique that has been shown to yield good results in terms of diagnostic accuracy in software [2; 3]. The diagnosis process in SFL is based on the analysis of the differences in program spectra (an abstraction over program traces) for passed and failed runs. *Passed* runs are executions of a program that were correctly completed, whereas *failed* runs are executions in which an error was detected. A program spectrum is characterised by an execution profile that indicates the active parts of a program during a run. Then, spectrum-based fault localisation entails identifying the part of the program whose activity most correlates with the detection of errors.

An advantage of SFL is that it does not rely on a model of the system under investigation and can be applied in resource-constrained environments due to the relatively low computational overhead [2]. These aspects elevate the technique as a good candidate for being implemented in MASs deployment. To the best of our knowledge spectrum-based techniques have not been employed within the MAS fault-diagnosis research, apart from our recent preliminary attempt published in a non-archival workshop, yet here we discuss further results [4].

In this paper, we follow a paradigm shift and propose an extension of the SFL technique called, *Extended Spectrum-based Fault Localisation for Multi-Agent Systems* (ESFL-MAS) in order to diagnose agents' behavioural faults when testing the overall system operation. Similar to the original approach, our proposal was used for shortening the diagnosis cycle for identifying agents that jeopardise the overall performance of a MAS-based application. We study the limitation of the current SFL and thus augment some of its features to support diagnosis in a multi-agent system. Moreover, a *similarity coefficient* ranks potential faulty agents that most contribute to the failure of the whole system. As

*This project has been partially supported by FCT, the Portuguese Agency for R&D (PhD Scholarship grant SFRH/BD/66717/2009) and CAPES, the Brazilian Agency for R&D (PhD Scholarship grant 9382-13-5).

the similarity coefficient influences the ESFL-MAS performance, we study the impact of different similarity coefficients in the MAS context. We evaluate 42 similarity coefficients on a comprehensive example to demonstrate the approach by instantiating a well-known toy-problem in the area of MAS known as the gold miners problem within a grid environment [5]. The gold-miners scenario encompasses key characteristics of both a logistic problem and a non-hierarchical MAS. The main contributions of our work are the following: (i) we discuss the limitation to use traditional SFL in agent-based systems and propose an extension to this technique to comply agents' features, (ii) and show that for the purpose of multi-agent systems the Jaccard, Ochiai, and Sorensen-Dice similarity coefficients outperforms the other coefficients when varying the number of passed and failed observations.

The remainder of the paper is organised as follows. The related work is discussed in Section 2 to reinforce our research challenges. In Section 3 we introduce an illustrative case of the Spectrum-based Fault Localisation technique. Section 3 introduces a formalisation of the spectrum-based diagnosis extension to be applied in MAS and, moreover, Section 4 describes the experimental setup and, in Section 5, we present and analysis the obtained results. At last, in Section 6 we conclude this work and suggest future research directions.

2 Related Work

Our work concerns diagnosis techniques in general, and, more specifically, diagnosis of faulty agents in a system. The diagnosis process is triggered by a detected deviation from the expected behaviour of a given system; this process aims to identify the origin of the failure, i.e. to discover all contributors for the unexpected event. Due to their anthropological and social features, the multi-agent community has proposed specific approaches to diagnose different types of fault in multi-agent systems.

Our analysis of related work in this area is twofold: *fault-based diagnosis* and *model-based diagnosis* [6]. The first relies on experts to model all known faults in a given domain; conversely, the second starts from a system's model (i.e. its structure and behaviour) while any abnormality is classified as a fault.

Albeit fault-based diagnosis is not recommended for multi-agent systems because synergy among agents is unpredictable, the literature presents architectures based on such an approach. Horling et al.'s [7] diagnosis system conducts identification and recovery processes through a predefined failure-causal graph; when the system fails, a suitable node is triggered and therefore, based on the failure's features, others correspondent nodes are activated, and thus the agent further reorganises its own tasks to recover to a non-faulty state. Similarly, Dellarocas and Klein [8] designed a domain-independent exceptions handling for agent-based systems using a sentinel-based approach. It introduces a knowledge base of generic exceptions. Agents, however, must also register their model so faults can be known in advance. The diagnosis process uses heuristic classification to search in a predefined tree for possible causes based on detected symptoms. Hence, in those proposals, agents very much depend on specific run-time conditions and actions previously taken. Unlikely to black-box approaches, fault-model techniques require a previous knowledge of possible faults.

The model-based diagnosis approach has also been applied to agent-based systems. For instance Roos et al. [9] address diagnoses where models are spatially distributed; each single agent has a local model which combines knowledge of all agents as well. However, this method trades off diagnosis precision over the capability of local-minimal diagnosis. Authors focus on how to effectively merge knowledge and co-ordinate components, whereas our work concentrates on how to minimise the knowledge usage.

A few approaches more tackle execution, monitoring, and diagnosis of plans in multi-agent systems. In [10], authors introduce the notion of *plan diagnosis* which considers the plan performed by agents as the system to be diagnosed; in their follow-up work, Jonge et al. [11] complement the plan diagnosis with the definition of secondary diagnosis. More precisely, the primary plan diagnosis clears the set of actions that leads to failure, whereas the secondary plan diagnosis underlies causes for such a failure. Towards the same direction, Micalizio [12] proposes an extended model of actions, yet demonstrating further interest in the whole process (from identification to recovery) with partial observable plans; the author has recently integrated the agent plan diagnosis with recovery processes through a conformal planner which guarantees a backup plan whenever it is possible [13]. Conversely to diagnosing failures in agent plans that are defined in advance, faulty behaviours depend on the context of each agent (or the system) to be detected and thus, besides the plan analysis, all the surroundings should contribute to the diagnosis, which the above works have not considered.

Multi-agent diagnosis has a different branch called *social diagnosis* that is concerned with determining coordination failures within a team of agents. Kalech and Kaminka [14; 15] achieve such by modelling hierarchy of behaviours while agents refresh their beliefs akin teammates. Through a matrix-based representation of coordination among agents, the system detects a fault and therefore diagnosis uses the aforementioned model to determine the abnormal agent or set of agents. In recent endeavours [16], they allow non-binary constraints and improve efficiency to large-scale and more realistic teams. This set of works is robust and scalable in collaborative MASs; however, their matrix-based model focused on the coordination failures rather than on how these influence the MAS overall performance.

The closer that the SFL has been applied to agent-based systems were in the work [17] where authors propose a spectrum-based technique to pinpoint concurrency faults in code blocks. Yet, to the best of our knowledge and after a careful review of the current literature, no work implements spectrum-based diagnosis specific to MAS. Additionally, all the community's efforts in the MAS community assume some kind of *a priori* model, for instance, the correct (or faulty) behaviour of agents including their possible actions. Consequently, these diagnosis techniques very much rely on how precise their knowledge is, thus increasing its dependency on the design process that is error-prone as well. To the contrary, we propose a spectrum-based fault diagnosis for multi-agent that collects dynamic information about the system rather than about predefined models, therefore decoupling the fault diagnosis from the design phase.

3 Spectrum-based Fault Localisation

Before describing our modifications to the SFL to localise faults in MASs, we illustrate the related concepts considering an implementation of the arc-tangent function in Table 1. It is meant to calculate the arc-tangent with two arguments, which are used to output the phase of a complex number. Analysing the signs of the input values, the function tests a sequence of cases and returns the angle in radians. For the sake of illustration, a fault is injected in the sign of one condition in instruction 6 of code in Table 1: the $arctan(x/y)$ is subtracted instead of being summed to π. An error occurs after the code inside the conditional statement is executed while $y >= 0$ and $x < 0$. Note that this fault can be latent in the system and only leads to errors under certain conditions.

The SFL collects data that provide a specific view of the dynamic behaviour of a monitored software [18]. When applied to procedural software, this data is collected at runtime and consists of a number of *hit flags* of a specific component; in our case, each statement of the code has an associated hit. To illustrate the concept of a program spectrum, observe Table 1; running this program results in the spectrum matrices where $\bullet$ (and $\circ$) represent that statement was (and was not) executed by the respective test case.

The set of test cases generates two types of programme spectra: ones with detected errors (called *failed* runs) and others without detected errors (called *passed* runs). These program spectra can be used for fault diagnosis by comparing both types of spectra and then analysing their differences. Throughout the observation of outputs, the process generates a vector of errors for each test case. We will demonstrate the approach through our *arc tangent* example.

Suppose we apply the inputs: $t_0 = \langle -4, -1 \rangle$, $t_1 = \langle 5, 0 \rangle$, and $t_2 = \langle 2, -3 \rangle$. The first and second inputs trigger statements without any fault and the program will pass these test cases. Yet, the last input will result in a failure represented by the error vector. The complete spectra for all test cases are as portrays Table 1. The difference between all spectra correctly identifies statement *result = arctan(y/x) -* π as the most likely location of the fault; this occurs because the previous statement is executed when the error is detected.

Clearly, this example has small number of test cases and statements; however it fully illustrates the manner SFL works. In a nutshell, the spectrum is a binary matrix whose columns represents the test case by which a (for our example) statement is activated. The error vector constitutes information about failure events in execution of test cases. Fault diagnosis consists in identifying the statement whose spectrum most resembles the error vector. Such a comparison is measured by the similarity coefficient, which will be studied in this paper and represented in this example by the *Jaccard Value* column.

Not explored in our work, but definitely interesting to pursue, is a hybrid approach combines SFL with model-based diagnosis (using generic models that can automatically be inferred from the application) in order to further improve diagnosis [19; 20].

Table 1: Jaccard Scores of Statements

	Statements	t_0	t_1	t_2	Jaccard Value
1	if(x $==$ 0 and y $==$ 0)	●	●	●	0.333
2	printf("Undefined");	○	○	○	0
3	else if(x $>$ 0)	●	●	●	0.333
4	result = arctan(y/x);	○	○	○	0
5	else if(y $>=$ 0 and x $<$ 0)	●	●	●	0.333
	/*Bug: sign '-' instead of '+'*/				
6	result = arctan(y/x) - π;	○	○	●	1.000
7	else if(y $<$ 0 and x $<$ 0)	●	●	○	0
8	result = arctan(y/x) - π;	●	○	○	0
9	else if(y $>$ 0 and x $==$ 0)	○	●	○	0
10	result = (π/2);	○	●	○	0
11	else if(y $<$ 0 and x $==$ 0)	○	○	○	0
12	result = -(π/2);	○	○	○	0
13	print("%d\n", result);	●	●	●	0.333
	Test Results	✓	✓	×	

4 Extending SFL for Multi-Agent Systems

The spectrum-based diagnosis is a simple, yet effective technique that demands for minimal information concerning the system to be diagnosed. All the aforementioned features motivate its use in an agent-based application; as a matter of fact, multi-agent diagnosis should not rely on *a priori* knowledge related to either its structure or the undergoing interactions. This happens because one of the most interesting characteristics of MASs is the dynamism of the system and its ability to adapt to new conditions.

We inspired by works in the area of testing and debugging of software, propose this extension of the spectrum-based fault localisation to tackle testing issues for MASs. Note that here we use the spectrum to diagnose agents in a higher level, i.e. in their behaviour level. We see the agent as a black box where the diagnosis technique does not have access to the source code. This higher level enables the ESFL-MAS to test off line the overall operation in the system exposed to different circumstances. The designer might recreate relevant scenarios and, compared with a standard one, the ESFL-MAS pinpoint the possible faulty agent. Thus, before to deploy the MAS, we can test different scenarios ensuring minimal performance. Below we discuss the three main points to detail the implemented modifications of the original SFL, presented in form of question.

Is one set of test cases enough to analyse MAS behaviour? There are two facets to be considered in this matter. First, normal object-oriented software produces an output after a single execution whereas the MAS must run throughout several time steps so we can truly see its emerging behaviour. That is, for each test case, we must run the MAS during a time frame to take into consideration its evolution. Second, a single execution per test case also limits our view of the system's behaviour as agents are autonomous and their activation paths (i.e. choices) might differ in different executions of the same test case. Therefore, we must execute several rounds of the same test case to cover as many paths as possible to assure robustness.

In a higher level of abstraction, what would be the information in a spectrum? This is crucial information as it discloses the activation states (dynamic behaviour) of the agents in the system. Based on this information the faulty agent is identified. However, the types of spectrum so far used in object-oriented and procedural software development do not meet agent-oriented software requirements. For instance, if we choose a block-hit spectrum and apply it to our problem, the X matrix is going to be full of 1's because agents are persistent entities and are always active. As so, we introduce a metric-based spectra, i.e. all agents are assessed by a metric and the 0's and 1's values indicate whether or not the agent is above a performance threshold, respectively.

How to simply assess the agent and overall performances? The technique should assess whether or not the MAS-based application works under its normal operational levels. We understand that such an assessment must rely on how effectively the overall system fulfils its requirements; in other words, how it performs the tasks it was built to execute. A direct solution for evaluating performance might use a domain-based metric that sets a threshold (e.g. a multi-agent system controlling an automotive shop-floor should produce 10 cars per hour). When the productivity is above this threshold the value expresses that the run has *passed*, otherwise it is deemed to have *failed*.

Figure 1 presents the information of the multi-agent system P handled by the ESFL-MAS and Algorithm 1 introduce the proposed algorithm. The definitions are

- $n \in \mathbb{N}$ is the number of agents in the multi-agent system and Ag_l is an agent l where $1 \leqslant l \leqslant n$.
- $\mathcal{T}$ is a set of test cases: $\mathcal{T} = \{T_1, \cdots, T_m | m \in \mathbb{N}\}$, where m is the number of test cases.
- Test case T_i considers a set of $q \in \mathbb{N}$ environmental variables that all agents can perceive.
- $C \in \mathbb{N}$ is the number of executions per test case and $0 < j \leqslant C$ is the current execution index.
- $K \in \mathbb{N}$ is the total number of time step in a given execution and $0 < k \leqslant K$ is the current time step index.
- $X_{T_i,j}$ is the spectrum matrix for the test case T_i and execution j. The whole tensor X has dimensionality of $m \times C \times K \times n$. A element $x_{T_i,j,k,l}$ represents the performance of the agent Ag_l at time step k and its value is defined by the function
  $$x_{T_i,j,k,l} = \begin{cases} 0 & \text{if } p_{k,Ag_l} \geq thr_{Ag_l} \\ 1 & \text{if } p_{k,Ag_l} < thr_{Ag_l} \end{cases}$$
  where p_{k,Ag_l} is the measure performance of the agent Ag_l at k and thr_{Ag_l} is a expected performance for the agent.
- E is a set of boolean elements where each element e_k represents the performance of the system at time step k and its value is defined by the function
  $$e_k = \begin{cases} 0 & \text{if } p_{k,MAS} \geq thr_{MAS} \\ 1 & \text{if } p_{k,MAS} < thr_{MAS} \end{cases}$$
  where $p_{k,MAS}$ is the measure performance of the MAS at k and thr_{MAS} is the expected performance of the MAS.

In a nutshell, the algorithm first initialises the variables and vectors with proper values (line 2-11); then it runs the MAS for the set of test cases and executions which builds the space of information and the error vector (line 12). For each agent and spectrum, the algorithm analyses the combination of agent's and system's performances to add in their respective counters (lines 13-29). ESFL-MAS produces a tuple of counters $\langle a_{ef}, a_{ep}, a_{nf}, a_{np} \rangle$ for each block of the multi-agent system P such that: a_{ef} and a_{ep} represent the number of test cases that covered the block and returns a failed and passed testing results, respectively; whereas a_{nf} and a_{np} denote the number of test cases that do not execute the block and return a failed and passed testing results, respectively. At last, the similarity of each agent is calculated through the counter vector. The algorithm returns a sorted list with ascending order of the most probable faulty agents.

The set of information of $\mathcal{T}$ test cases and C executions per test case constitute a 4-dimensional tensor ($m \times C \times K \times n$), where the metric-based spectrum is a matrix $X_{T_i,j}$. Also, its columns and lines correspond respectively to n different agents of the multi-agent system and

$$\begin{array}{ccc} \mathcal{T} & & X_{T_i,j} & E_{T_i,j} \\ \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_m \end{bmatrix} & \times C & \begin{array}{c} \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1K} & x_{2K} & \cdots & x_{nK} \end{bmatrix} \\ \begin{matrix} Ag_1 & Ag_2 & \cdots & Ag_n \end{matrix} \end{array} & \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_K \end{bmatrix} \end{array}$$

Figure 1: Information of MAS P to ESFL-MAS

Algorithm 1: ESFL-MAS Algorithm

```
   Input: Multi-Agent System $P$, set of test cases $\mathcal{T}$,
          number of executions per test case $C$, and
          similarity coefficient $s$
   Output: Diagnosis report $D$
1  begin
2      $m \longleftarrow |\mathcal{T}|$
3      $n \longleftarrow$ Get_NumOfAgents($P$)
4      $D \longleftarrow \emptyset$
5      for $j = 0$ to $n$ do
6          $a_{11}(j) \longleftarrow 0$
7          $a_{10}(j) \longleftarrow 0$
8          $a_{01}(j) \longleftarrow 0$
9          $a_{00}(j) \longleftarrow 0$
10         $S[j] \longleftarrow 0$ // Similarity $s$ of agent $j$
11     end
12     $(X, E) \longleftarrow$ Run_MAS($P$, $\mathcal{T}$, $C$)
13     for $i = 0$ to $m$ do
14         for $j = 0$ to $C$ do
15             $K \longleftarrow$ Get_NumTimeSteps($X[i,j]$)
               for $k = 0$ to $K$ do
16                 for $l = 0$ to $n$ do
17                     if $x[i,j,k,l] = 1 \wedge e[i,j,k] = 1$ then
18                         $a_{11}(l) \longleftarrow a_{11}(l) + 1$
19                     else if $x[i,j,k,l] = 0 \wedge e[i,j,k] = 1$ then
20                         $a_{01}(l) \longleftarrow a_{01}(l) + 1$
21                     else if $x[i,j,k,l] = 1 \wedge e[i,j,k] = 0$ then
22                         $a_{10}(l) \longleftarrow a_{10}(l) + 1$
23                     else if $x[i,j,k,l] = 0 \wedge e[i,j,k] = 0$ then
24                         $a_{00}(l) \longleftarrow a_{00}(l) + 1$
25                     end
26                 end
27             end
28         end
29     end
30     for $i = 0$ to $n$ do
31         $S[i] \longleftarrow s(a_{11}(i), a_{10}(i), a_{01}(i), a_{00}(i))$
32     end
33     $D \longleftarrow$ Sort($S$)
34     return $D$
35 end
```

K time step per execution as we have assumed a discrete time. The information concerning the time step in which an under-performance of the system was detected constitutes another vector named E. This vector represents a hypothetical part of the MAS that is responsible for observing all failure events.

Spectrum-based diagnosis essentially consists in identifying the agent whose column resembles the under-performance vector the most. The similarity coefficient quantifies these resemblances and, assuming the high similarity with the under-performance vector, indicates a high probability for the corresponding agent to have caused the detected "failure;" in other words, it ranks the agents with respect to their likelihood of containing faults. However, there are also other aspects of MASs that are not fully taken into account by ESFL-MAS, such as: interacting agents, cascading effects of a faulty agent, and so forth. To handle all these elements at once is too much for one work; though, we are aware of them and will study solutions for all these issues.

5 Experimental Setup

To assess the effectiveness of the various similarity coefficients in the context of spectrum-based fault localisation in multi-agent systems, we are evaluating the ranking of the faulty agent computed using ESFL-MAS with several similarity coefficients. Our paper is inspired by works which compares the fault localisation capabilities of 42 similarity coefficients for C programs and spreadsheets respectively [21; 22]; however, in contrast to them, our focus is on multi-agent systems. For the sake of clarity and due to the lack of space, we refer the interested reader elsewhere [21] for more details about these coefficients as this paper will not present all of them.

We consider a widely-used scenario for agent, used in Agent Contest 2007, in order to experiment and evaluate the impact of the coefficient in the ESFL-MAS. This is the gold-miners scenario and to explain it we quote their description:

> "Teams of gold miners find themselves exploring the same area, avoiding trees and bushes and competing for the gold nuggets spread around the woods. The gold miners of each team coordinate their actions in order to collect as much gold as they can and to deliver it to the trading agent located in a depot where the gold is safely stored." [1]

Our test suite uses only one team of gold miners per scenario as, for now, we do not want to introduce any dynamic environmental changes.

There are two main reasons to choose the gold miners scenario. First, though in a first look the chosen scenario seems simple, it has most features of pick-delivery problems present in domains such as transportation, supply-chain, and networks; additionally, from the multi-agent structure, it complies non-hierarchical solutions commonly used to manage peer-to-peer networks and traffic control. Second, all components of the system (including model of the world, agents, and interaction protocols) were tested and validated by the MAS community; and this is important to our case because we have higher confidence that only our injected faults will contribute to our results.

[1]https://multiagentcontest.org/2007. Page accessed at 2014-08-04

We use the agents implementation based on the JASON agent-oriented platform for MAS development [23] to create our test suite. Some modifications were made in the original application so we were able to inject a faulty agent in the scenario. The faults focus on behavioural facets of agents and they may have different origins; for instance, the agent misinterprets a perception and thus takes an erroneous action. The system is composed by 20 agents and 400 gold nuggets. We randomly generate 5 test cases as they are part of the ESFL-MAS input (see Section 4). Each of the test cases corresponds to a set of initial positions for: all agents, the depot, and all gold nuggets; also the test cases were executed 75 times each, with time limit of 1000 steps per execution.

Additionally, we collect information about agents' performance in order to build the run-time profile of the system, i.e. the spectrum. These spectra reflect the dynamic behaviour of the agents in the system and how they fulfil their tasks. The gold-miners example might be seen as rudimentary version of a logistic problem and thus, the average time of completed shipments (e.g. delivered gold nuggets) for each individual agent can be used as metric for such domain. Indeed it is able to disclose evidence of agents' performance as it maintains the generality and simplicity of the proposed logistic problem [24]. Likewise, the metric used to establish the performance of the system, according a given threshold, is the average time between two shipments to the depot. However, we can use whatever other metric that reflects the system's "productivity."

Using the original implemented version of the gold-miners as the correct version, we use its execution output as error detection baseline for each test case. The time step is labelled as "failed" if the output is below the corresponding threshold established in the correct version, and as "passed" otherwise. With this strategy, we detect when the system underperforms deviating from the expected (desired) behaviour. That is, we provide indications to the system designer how to evaluate the agent's blueprints.

We run the proposed fault localisation approach and obtain a list of probable faulty agents, sorted by the likelihood to generate an under-performance event (that is, the computed value of the similarity coefficient, currently being used). Since agents with a high probability of exhibiting a fault behaviour are more likely to be first in the ranking, we use this indication to measure the quality of the diagnosis, also known as *accuracy*. As it is reported in [25], let $d \in \{1, \cdots, N\}$ be the index of the block that we know to contain the fault. For all $j \in \{1, \cdots, N\}$, let s_j denote the similarity coefficient calculated for block j. Then the ranking position is given by

$$\tau = \frac{|\{j|s_j > s_d\}| + |\{j|s_j \geq s_d\}| - 1}{2} \tag{1}$$

Abreu et al. have elsewhere [25] defined accuracy or quality of the diagnosis as the effectiveness to identify the faulty block. This metric represents the percentage of blocks that need not be considered when searching for the fault by traversing the ranking. It is defined as

$$q_d = \left(1 - \frac{\tau}{N-1}\right) * 100\% \tag{2}$$

In the remainder of this paper, values for q_d will be expressed as percentages.

6 Observation of the ESFL-MAS Sensibility

One, when analysing the ESFL-MAS, instinctively recognizes the main influence that both the similarity coefficient and the metrics have in the technique's performance. Nevertheless, we start the ESFL-MAS sensibility analysis from the similarity coefficients because there are previous works in the software-engineering domain that could verify, at some degree, the correctness of our results.

Sensitivity analysis intends to estimate the influence of the number of passed and failed time steps on the accuracy of the fault diagnosis. To do so, we evaluate q_d while varying the number of passed (N_p) and failed time steps (N_F) that are encompassed in the diagnosis process. Since the ratio of failed and passed time steps is very small (nearly 0.02) and no previous experiment analysing the sensitivity of ESFL-MAS have been performed, we study the impact the number of executions has on the diagnostic accuracy q_d throughout all the range of available data; this means we have chosen to vary the number of passed and failed time steps in the range of 5-100% (with steps of 5%)of the total number of passed and failed runs per spectrum.

We have observed that, even though similarity coefficients have been proposed for different purposes, in our experiments some of them have similar responses. Figure 2 and 3 show two representative examples of sets of coefficients with the best performances. We plot q_d according to the several coefficients for N_P and N_F varying from 1-100%. For each entry of these graphs, we average q_d over 100 randomly selected combinations of N_P passed runs and N_F failed runs, where we verify that the variance in the measured values of q_d is negligible.

The Figure 2 portrays the response of Jaccard [26], Ochiai [27], and Sorensen-Dice [28] coefficients. All three outperform all others coefficients regarding the stability of their response throughout the whole range of N_P and N_F. On the other hand, the group composed by Accuracy [29], Least Contradiction [29], Rogers and Tanimoto [30], and Simple-Matching [31] has the best maximum value for q_d; however, as it can be seen (Figure 4), the performance of the aforementioned coefficients very much depends on both N_P and N_F. This fact makes them poor candidates to be applied when the effectiveness of either the fault detection or the generated test cases is unknown, because the accuracy can drop in average 20% when varying N_P and almost 30% when varying N_F.

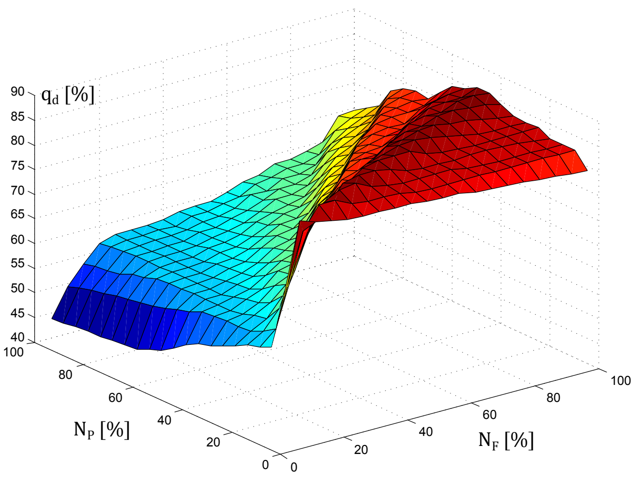

Figure 3: Observation of sensibility of Accuracy, Least Contradiction, Rogers and Tanimoto, and Simple-Matching

Figure 4 depicts the impact of N_F in the quality of diagnosis for $N_P = 50\%$. Inspecting the results for different similarity coefficients confirms the stable response of Jaccard and Ochiai coefficients, with a slightly better result from Jaccard. In addition, the Accuracy coefficient equals both Jaccard's and Ochiai's performance near 70% of N_F, which means in average 25 time steps with underperformance against 490 passed time steps. Across the entire test bed , we found that some coefficients lose performance while increasing the number of failed time steps. An example is the Collective Strength [32] depicted in Figure 4.

From our experiments on the impact of the number of executions on the accuracy q_d we can draw the following conclusions. As the technique is highly influenced by the number of passed time steps, the inclusion of more failed time steps not always guarantee a better quality of diagnosis. The best strategy to improve accuracy is to run the MAS with test cases that could induce underperformance more frequently. This supports how crucial are the test-cases selection for

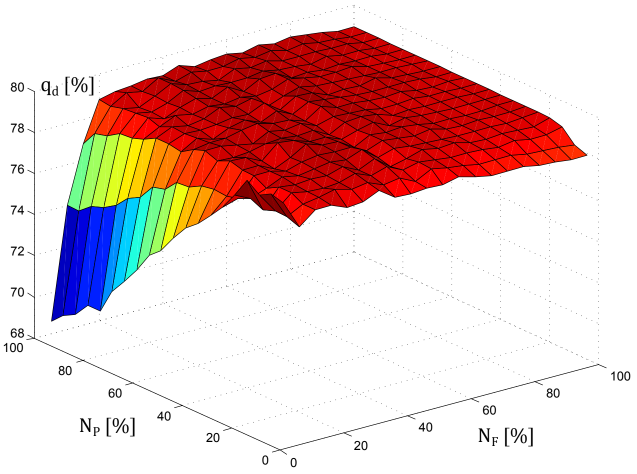

Figure 2: Observation of sensibility of Jaccard, Ochiai, and Sorensen-Dice

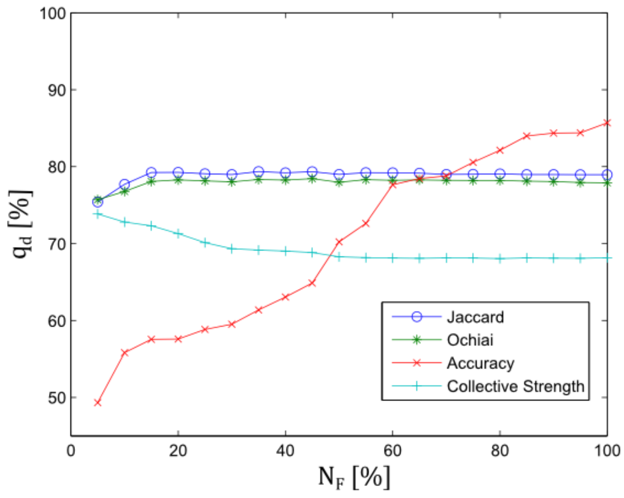

Figure 4: Impact of N_F on q_d for $N_P = 50\%$

spectrum-based diagnosis. It is important to highlight that these conclusions are within the context of our test bed, in which we do not consider possible social interaction among agents and only one agent had a fault.

7 Conclusions

Extended Spectrum-Based Fault Localisation aims to identify agents that may jeopardise the overall performance through misdirected reasoning. The first appraisal described is supposed to be used in the context of testing MAS in a system level in order ensure expected behaviour in a set of scenarios. Besides, not all aspects of MASs are covered in this version, yet we achieve prominent results, giving a good prospect for the ESFL-MAS application. Furthermore, we analysed the impact on the diagnostic accuracy of varying the number of observations (quantity) for different similarity coefficients. The gold miners scenario [5], developed in JASON agent-oriented platform, was adopted to perform all analysis.

First of all, as the technique is highly influenced by the number of "passed" time steps, the inclusion of more "failed" time steps not always guarantee a better quality of diagnosis q_d. The best strategy to improve accuracy is to run the MAS with test cases that could induce underperformance more frequent. This supports the relevance of test-cases selection for spectrum-based diagnosis and steers your work to investigate more deeply in the generation of sound test cases for MAS. We need to acknowledge, that these conclusions are within the context of the discussed test bed, in which we do not consider possible social interactions among agents and where only one agent presents a faulty behaviour.

Along with the above conclusion, results sustain that for Ochiai, Jaccard, and Sorensen-Dice coefficients the minimal of 20% of test cases should have failed executions. This fact must be taken into account by the designers when implementing scenarios to test MAS. Moreover, as plotted in the Figure 4, we see that for only 50% of "passed" time steps and 20% of "failed" executions the ESFL-MAS is able to strongly (80% accuracy) indicate the presence of unwanted behaviours in some of the agents. From the designer point of view, this is critical and useful results as he/she will experience a reduction of time spent on testing and debugging phase.

The comparison of SFL to other diagnosis approaches remains for future work. We did not compare to Kalech's [16] and Micalizio's [13] works because a direct comparison is not possible because both of them focus on a specific feature of MASs, which is team-mates for Kalech and plan of actions for Micalizio. Although we intend to compare both works in our test suite as well as in their empirical setups.

After the analysis described in this paper, future works will address the study of several aspects of the ESFL-MAS in order to establish correlations and degrees of influence of its component on the quality and precision of diagnosis. We will investigate the impact of metric-based error detection in the accuracy q_d also we intend to evaluate the evolution of the accuracy q_d as a function of detection quality, aiming to obtain a better resolution on the performance of the various similarity coefficients as well as on the agents' and system's evaluation metrics. Hence, we intend to move towards a more design-independent solution. Moreover, it is also interesting to study the sensitivity of our technique to different number erroneous time steps so we can be able to state the minimum number of errors that should be detected to assure the best expected performance. Finally, we will test the ESFL-MAS in a much more realistic scenarios with dynamic environments, more agents, more interaction between agents taking advantage of the existing benchmarks for MASs, as well as experiment with the spectrum-based reasoning technique [33; 34] which reasons in terms of multiple faults.

References

[1] CuD. Nguyen, Anna Perini, Carole Bernon, Juan Pavón, and John Thangarajah. Testing in multi-agent systems. In Marie-Pierre Gleizes and JorgeJ. Gomez-Sanz, editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 180–190. Springer Berlin Heidelberg, 2011.

[2] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J.C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780 – 1792, 2009. SI: {TAIC} {PART} 2007 and {MUTATION} 2007.

[3] Nuno Cardoso and Rui Abreu. A kernel density estimate-based approach to component goodness modeling. In *AAAI*, 2013.

[4] Lúcio Sanchez Passos, Rosaldo Rossetti, and Joaquim Gabriel. Diagnosis of unwanted behaviours in multi-agent systems. In *Proceedings of the 11th European Workshop on Multi-Agent Systems*, 2013.

[5] Rafael H Bordini, Jomi F Hübner, and Daniel M Tralamazza. Using jason to implement a team of gold miners. In *Computational Logic in Multi-Agent Systems*, pages 304–313. Springer, 2007.

[6] Peter J.F. Lucas. Analysis of notions of diagnosis. *Artificial Intelligence*, 105:295 – 343, 1998.

[7] Bryan Horling, Brett Benyo, and Victor Lesser. Using self-diagnosis to adapt organizational structures. *Proceedings of the 5th International Conference on Autonomous Agents*, pages 529–536, June 2001.

[8] C. Dellarocas and M. Klein. An experimental evaluation of domain-independent fault handling services in open multi-agent systems. In *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, pages 95 –102, 2000.

[9] N. Roos, A. ten Teije, and C. Witteveen. Reaching diagnostic agreement in multi-agent diagnosis. In *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pages 1256 –1257, july 2004.

[10] Nico Roos and Cees Witteveen. Models and methods for plan diagnosis. *Autonomous Agents and Multi-Agent Systems*, 19(1):30–52, 2009.

[11] Femke Jonge, Nico Roos, and Cees Witteveen. Primary and secondary diagnosis of multi-agent plan execution. *Autonomous Agents and Multi-Agent Systems*, 18(2):267–294, 2009.

[12] Roberto Micalizio. A distributed control loop for autonomous recovery in a multi-agent plan. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1760–1765, San

Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[13] Roberto Micalizio. Action failure recovery via model-based diagnosis and conformant planning. *Computational Intelligence*, 29(2):233–280, 2013.

[14] Meir Kalech, Michael Lindner, and Gal A. Kaminka. Matrix-based representation for coordination fault detection: a formal approach. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 162:1–162:8, New York, NY, USA, 2007. ACM.

[15] Meir Kalech and Gal A. Kaminka. On the design of coordination diagnosis algorithms for teams of situated agents. *Artificial Intelligence*, 171:491 – 513, 2007.

[16] Meir Kalech. Diagnosis of coordination failures: a matrix-based approach. *Autonomous Agents and Multi-Agent Systems*, 24(1):69–103, 2012.

[17] Feyzullah Koca, Hasan Sozer, and Rui Abreu. Spectrum-based fault localization for diagnosing concurrency faults. In Husnu Yenigun, Cemal Yilmaz, and Andreas Ulrich, editors, *Testing Software and Systems*, volume 8254 of *Lecture Notes in Computer Science*, pages 239–254. Springer Berlin Heidelberg, 2013.

[18] Thomas Reps, Thomas Ball, Manuvir Das, and James Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. In Mehdi Jazayeri and Helmut Schauer, editors, *Software Engineering ESEC/FSE'97*, volume 1301 of *Lecture Notes in Computer Science*, pages 432–449. Springer Berlin Heidelberg, 1997.

[19] Rui Abreu, Wolfgang Mayer, Markus Stumptner, and Arjan JC van Gemund. Refining spectrum-based fault localization rankings. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 409–414. ACM, 2009.

[20] Wolfgang Mayer, Rui Abreu, Markus Stumptner, AJ van Gemund, et al. Prioritising model-based debugging diagnostic reports. In *Proceedings of the International Workshop on Principles of Diagnosis (DX)*, pages 127–134, 2009.

[21] Birgit Hofer, André Riboira, Franz Wotawa, Rui Abreu, and Elisabeth Getzner. On the empirical evaluation of fault localization techniques for spreadsheets. In *Fundamental Approaches to Software Engineering*, pages 68–82. Springer, 2013.

[22] David Lo, Lingxiao Jiang, Ferdian Thung, Aditya Budi, et al. Extended comprehensive study of association measures for fault localization. *Journal of Software: Evolution and Process*, 2013.

[23] Michael Wooldridge Rafael H. Bordini, Jomi Fred Hubner. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Ltd, 2007.

[24] Goknur Arzu Akyuz and Turan Erman Erkan. Supply chain performance measurement: a literature review. *International Journal of Production Research*, 48(17):5137–5155, 2010.

[25] R. Abreu, P. Zoeteweij, and A. J C Van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007*, pages 89–98, Sept 2007.

[26] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[27] Andréia da Silva Meyer, Antonio Augusto Franco Garcia, Anete Pereira de Souza, and Cláudio Lopes de Souza Jr. Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (zea mays l). *Genetics and Molecular Biology*, 27:83 – 91, 00 2004.

[28] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[29] Liqiang Geng and Howard J Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.

[30] David J Rogers and Taffee T Tanimoto. A computer program for classifying plants. *Science*, 132(3434):1115–1118, 1960.

[31] Robert R Sokal. A statistical method for evaluating systematic relationships. *Univ Kans Sci Bull*, 38:1409–1438, 1958.

[32] Charu C. Aggarwal and Philip S. Yu. A new framework for itemset generation. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 18–24, New York, NY, USA, 1998. ACM.

[33] Rui Abreu, Peter Zoeteweij, and Arjan JC van Gemund. A dynamic modeling approach to software multiple-fault localization. *Proc. DX*, 8, 2008.

[34] Rui Abreu and Arjan JC Van Gemund. Diagnosing multiple intermittent failures using maximum likelihood estimation. *Artificial Intelligence*, 174(18):1481–1497, 2010.