多故障的相似系数经验评价

Empirical Evaluation of Similarity Coefficients for Multiagent Fault Localization

Lúcio S. Passos, Rui Abreu, *Member, IEEE*, and Rosaldo J. F. Rossetti, *Member, IEEE*

***Abstract*—Detecting and diagnosing unwanted behavior in multiagent systems (MASs) are crucial to ascertain correct operation of agents. Current techniques assume *a priori* knowledge to identify unexpected behavior. However, generation of MAS models is both error-prone and time-consuming, as it exponentially increases with the number of agents and their interactions. In this paper, we describe a light-weight, automatic debugging-based technique, coined extended spectrum-based fault localization for MAS (ESFL-MAS), that shortens the diagnostic process, while only relying on minimal information about the system. ESFL-MAS uses a heuristic that quantifies the suspiciousness of an agent to be faulty. Different heuristics may have a different impact on the diagnostic quality of ESFL-MAS. Our experimental evaluation shows that 10 out of 42 heuristics (namely accuracy, coverage, Jaccard, Laplace, least contradiction, Ochiai, Rogers and Tanimoto, simple-matching, Sorensen-dice, and support) yield the best diagnostic accuracy (96.26% on average) in the context of the MAS used in our experiments.**

***Index Terms*—Fault diagnosis, multiagent systems (MASs), reliability, spectrum-based fault localization (SFL).**

I. Introduction

PREVIOUS approaches to ascertain nominal behavior of multiagent systems (MASs) assume *a priori* knowledge (i.e., model) to diagnose observed failures (see [1], [2]). This knowledge can only be properly specified when designers fully understand the environment upon which agents act as well as agents' state space.

However, in practice, due to: 1) the complexity of MASs; 2) dynamism of the environment; and 3) presence of legacy systems, MAS and/or agent models are rather laborious to build. As a consequence, building the model is an error-prone task. Any knowledge not included in the model may therefore influence the capability of model-based fault diagnosis to effectively recognize faults. The more realistic the problem becomes, the more complex it gets to minutely encompass all characteristics of related entities in the model. Diagnosis techniques for MASs thus should not completely rely on models built *a priori* to pinpoint unexpected behaviors.

To address this issue, this paper considers an approach that relies on minimal prior knowledge to pinpoint behavioral faults in MASs. Spectrum-based fault localization (SFL) is a promising technique that does not rely on an explicit model of the system under analysis and has been shown to yield good diagnostic accuracy for software systems [3]–[5].

The diagnosis process of SFL is based on the analysis of the differences in the so-called spectrum [6] (abstraction over program traces) for passed and failed runs. SFL isolates the faulty component, using a similarity coefficient as heuristic, whose activity mostly correlates with observed failures. More importantly, SFL can be applied to resource-constrained environments due to its relatively low computational overhead [4], [7]. Such properties suggest that SFL is a well-suited technique for MASs. This paper aims to study the suitability of SFL as a diagnosis approach in the specific context of MASs, proposing the necessary extensions to cope appropriately with the intrinsic characteristics of such a domain.

Traditional uses of SFL have two fundamental limitations when directly applied to MASs. First, the usual abstraction of the spectrum as the involvement of software components generates a uniform spectrum (with no useful information to discover the fault), since agents are time-persistent entities and are constantly perceiving and acting upon the environment. The second limitation of SFL regards the agent's autonomic facet. To localize the faulty component, SFL assumes that components output the same results for the same set of inputs; this assumption does not hold for MASs as agents might act differently while facing the same set of events.

The extended spectrum-based fault localization for MAS (ESFL-MAS) solves the first limitation by giving a performance-oriented view over the spectrum by tracing agent's behavior expectancy during a run to provide useful diagnostic information and to solve the time-dependency problem. Concerning the second limitation, we have proposed a simple yet elegant solution, which is to execute the monitored MAS several times in order to catch multiple instances of agents' behavior for different circumstances.

Manuscript received June 8, 2015; revised October 2, 2015; accepted November 13, 2015. Date of publication April 22, 2016; date of current version April 13, 2017. This work was supported in part by the Fundação para a Ciência e a Tecnologia, in part by the Portuguese Agency for Research and Development under Grant SFRH/BD/66717/2009, in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, and in part by the Brazilian Agency for Research and Development under Grant BEX 9382-13-5. This paper was recommended by Associate Editor W.-K. V. Chan.

L. S. Passos and R. J. F. Rossetti are with the Department of Informatics Engineering, Faculty of Engineering, University of Porto, Porto 4200-465, Portugal, and also with the Artificial Intelligence and Computer Science Laboratory, University of Porto, Porto 4200-465, Portugal (e-mail: lucio.san.passos@gmail.com; rossetti@fe.up.pt).

R. Abreu is with the Department of Informatics Engineering, Faculty of Engineering, University of Porto, Porto 4200-465, Portugal, also with the HASLab/INESC TEC, Braga 4710-057, Portugal, and also with the Palo Alto Research Center, Palo Alto, CA 94304 USA (e-mail: rui@computer.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2016.2523905

Additionally, we have suggested an optimization, named MAS filter, which increases diagnostic accuracy by filtering low-entropy spectrum's rows.

Literature has shown that there is no standard similarity coefficient that yields the best result for SFL [3], [5], [8]. Empirical evaluation is, therefore, essential to establish which set of heuristics excels for the specific context to which SFL is being applied. To the best of our knowledge, SFL has not as yet been applied to diagnose behavioral faults in MASs; there is hence the need to empirically evaluate different formulas using known faults to compare the performance yielded by several similarity coefficients. This paper empirically determines the best similarity coefficients for SFL including the necessary extensions in the MAS context.

Our experiments study an exhaustive list of similarity coefficients. As the diagnostic accuracy of the coefficients may be similar, we have done a clustering analysis to understand which set of coefficients are better for MASs. The impact on the diagnostic accuracy was also studied regarding the quantity of available data to ESFL-MAS as well as the of error detection quality. Based on the empirical results, accuracy, coverage, Jaccard, Laplace, least contradiction, Ochiai, Rogers and Tanimoto, simple-matching, Sorensen-dice, and support coefficients excel by showing stability while varying the number of passed and failed time steps and reaching high diagnostic accuracy for low error detection precision (EDP).

This paper makes the following contributions.

1) We discuss the limitations of applying SFL with commonly used types of spectrum to time-persistent entities such as agents.
2) We describe the ESFL-MASs to diagnose agent behavioral faults when testing the system as a whole.
3) We present an experimental study on the impact of 42 heuristics in the ESFL-MAS diagnostic accuracy using the well-known and real-world representative pick-up and delivery problem (PDP) as test suite.
4) We show that for ESFL-MAS the accuracy, coverage, Jaccard, Laplace, least contradiction, Ochiai, Rogers and Tanimoto, simple-matching, Sorensen-dice, and support outperform the remainder coefficients across the entire quantity and quality data space (yielding 96.26% diagnostic accuracy) in the specific conditions of our test suite.

This paper consolidates conference papers [9] (Best Paper Award) and [10] by the authors.

The remainder of this paper is organized as follows. Related works are discussed in Section II. Section III discusses main concepts of SFL and presents an illustrative example for sequential software. Section IV introduces definitions used throughout this paper; furthermore, the diagnosis problem for MASs is defined. Section V describes the constraint of SFL and the proposed changes in order to use SFL for MASs. Section VI explains both the created test suite and the data collection process. In Section VII, we evaluate the effects of different similarity coefficients in the diagnosis through varying the amount of available data and the precision of the error detection phase that precedes ESFL-MAS. Finally, Section VIII concludes this paper and suggests future work.

II. Related Work

There is a wide set of approaches to increase the reliability of MASs. Formal verification, along with model checking and theorem proving (see [2]), has received great attention by the community. While exhaustive and automated, such approaches are computationally costly, despite employing a number of reduction techniques, as well as rely solely on the model of the system under test to certify correct functioning.

In the scope of fault diagnosis in MASs, the related literature is twofold: 1) fault-based diagnosis and 2) model-based diagnosis [11]. The former relies on experts to model all known faults in a given domain; conversely, the latter starts from a system's model (i.e., its structure and behavior) of expected functioning.

Albeit fault-based diagnosis is not recommended for agent-based applications because the synergy among agents is unpredictable, the literature presents architectures based on such an approach. Horling *et al.*'s [12] diagnosis system conducts identification and recovery processes through a predefined failure-causal graph. Dellarocas and Klein [13] designed a domain-independent exception handling for agent-based systems using a sentinel-based approach.

Regarding plan diagnosis, de Jonge *et al.* [14] proposed a multilevel diagnostic system that pinpoints the set of erroneous actions and the causes for such a failure. Moreover, Micalizio and Torasso [15] combined the agent plan diagnosis with recovery processes. Plan-diagnosis techniques rely on given agents' plans restricting their usage at testing-system level.

Multiagent diagnosis has a different branch called social diagnosis that is concerned with determining coordination failures within a team of agents. Kalech and Kaminka [16] achieved this by modeling hierarchy of behaviors while agents refresh their beliefs akin team-mates. In a recent endeavor [17], they allow nonbinary constraints and improve efficiency to large-scale teams. These efforts focus on the coordination failures rather than their effects on the MAS overall performance.

Scalability is an important issue for fault detection and diagnosis in large-scale MASs. Reference [18] is seminal in dealing with this issue and devise overseer, which monitors large distributed application by overhearing agent communications. Previously cited endeavors by Micalizio and Torasso [15] and Kalech [17] claim to be scalable to large teams, but have performed experiments with a few agents only. Conversely, Serrano *et al.* [19] described a debugging tool that selectively monitors agent interactions and performs post-mortem analysis to detect emergency of unacceptable collaborations. These approaches are constrained to collaborative MASs.

SFL has been applied to pinpoint concurrency faults in code blocks [20]. However, still no work implementing SFL in

TABLE I
FAULTY JAVA METHOD FOR BINARY SEARCH. THE INPUT FOR TEST CASES IS COMPOSED BY SET = {1, 2, 3, 4, 5} AND TARGET

Java Source Code	S	*target*						
public boolean search (int[ ] *set*, int *target*) {		null	[ ]	1	2	3	4	5
if (*target* == null) {	1	1	1	1	1	1	1	1
return false; }	2	1	0	0	0	0	0	0
int *low* = 0, *high* = *set*.length −1;	3	0	1	1	1	1	1	1
while (*low* <= *high*) {	4	0	1	1	1	1	1	1
int *ix* = (*low* + *high*)/2;	5	0	0	1	1	1	1	1
int *rc* = *target*.compareTo(*set*[*ix*]);	6	0	0	1	1	1	1	1
if (*rc* < 0) {	7	0	0	1	1	1	1	1
high = *ix* − 1;	8	0	0	1	1	0	0	0
} else if (*rc* > 0) {	9	0	0	1	1	1	1	1
//Bug: sign '−' instead of '+'	-	-	-	-	-	-	-	-
low = *ix* − 1;	10	0	0	0	1	0	1	1
} else {	11	0	0	1	0	1	0	0
return true; }	12	0	0	1	0	1	0	0
return false; }	13	0	1	0	0	0	0	0
}	*e*	**0**	**0**	**0**	**1**	**0**	**1**	**1**

∗ S: Statement; *e*: error vector.

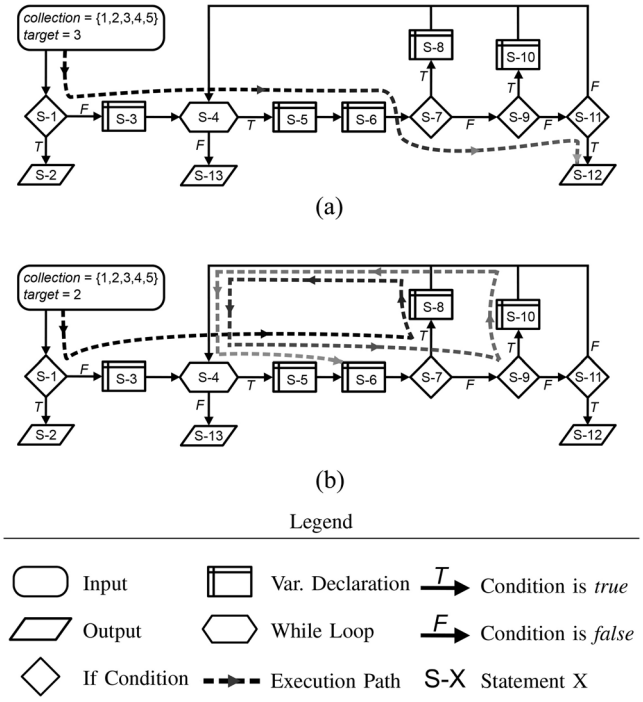

Fig. 1. Work-flow of the Java code in Table I with the execution flow of two test cases. It outputs the expected result in (a), whereas it has an unexpected output in (b). Execution path for inputs (a) set = {1, 2, 3, 4, 5} and target = 3 and (b) set = {1, 2, 3, 4, 5} and target = 2.

MASs has been reported yet. Therefore, all the MAS community's efforts assume some kind of *a priori* model, such as the correct (or faulty) behavior of agents including possible actions and plans. As a consequence, these diagnosis techniques very much rely on the level of details encompassed in the model, which is error prone as well. This paper goes further and proposes a spectrum-based fault diagnosis capable of collecting dynamic information about the MAS and does not depend on predefined models.

III. SPECTRUM-BASED FAULT LOCALIZATION

SFL is a dynamic program analysis technique, which requires minimal information about the system to be diagnosed. The binary search algorithm implemented in Java (first column on the left in Table I) will be considered to illustrate the SFL procedure throughout this section. Briefly, this method finds the integer target given an ordered set of integers as an array. Binary search divides the sorted set in half until the sought-for item is found, or until it is determined that the item cannot be present in the smaller set. Let us assume that this Java source code has a bug in the sign of statement 10[1] where the sign "−" in the equation low = *ix*−1 should be a "+" sign. This fault can be latent in the system and only leads to errors under certain conditions, but, when it is activated, it leads to an array-out-of-bound exception in the method execution.

We first need to introduce the concept of spectrum to correctly understand SFL, which is a set of runtime data of the dynamic behavior exhibited by a piece of software. Literature shows that exist different forms of recording such a spectrum (see [6]). Regardless its form, a spectrum can always be abstracted in terms of two general elements.

[1]The commented line is not accounted for in SFL since the Java compiler excludes any comment from the binary version.

1) *Component:* It is an system's element that, for diagnosis purposes, is considered to be atomic. We consider a statement as a spectrum's component in the example.
2) *Transaction:* It is a specific information about the component. In our illustrative example, this specific information is the involvement of a particular statement.

The next step is to gather runtime profiles containing the specific information about each system's component from test case runs. The result of a test case can be either nominal ("pass") or error ("fail"). This information of in which test case the software failed constitutes the error vector. Back to our example, let us suppose the method was tested for seven test cases where the input set = {1, 2, 3, 4, 5} is immutable while the target input has values according to presented in Table I. The dynamic behavior of the search method through the statement-hit spectrum was collected and it is shown in the right side of Table I. One must bear in mind that this form of spectrum indicates whether or not a certain statement was involved given a test case.

Fig. 1 presents the work-flow of the Java code in Table I as well as the execution paths for test cases target = 3 [Fig. 1(a)] and target = 2 [Fig. 1(b)]. The process to build the spectrum's column of a test case is: as the execution path progresses through the work-flow, positions of involved statements are set to 1, whereas positions of noninvolved statements are set to 0.

As an example, let us build the spectrum's column for both test cases in Fig. 1. Following the execution path in Fig. 1(a), the first involved statement is $S-1$ and consequently the first position of the column is set to 1; then, the next involved statement is $S-3$ that, at this point of the run, results in [1 0 1]

(observe the noninvolvement of $S-2$ is represented by 0 in the respective vector's position). Setting as 1 every involved statement along the execution path, which are: $S-1$, $S-3$, $S-4$, $S-5$, $S-6$, $S-7$, $S-9$, $S-11$, and $S-12$, the final spectrum's column for test case target = 3 is [1 0 1 1 1 1 1 0 1 0 1 1 0] as shown in Table I. In Fig. 1(b), one can observe the complete different execution path that involves statements: $S-1$, $S-3$, $S-4$, $S-5$, $S-6$, $S-7$, $S-8$, $S-9$, and $S-10$ (the last two activated after another while cycle); using the same building process, the spectrum's column is [1 0 1 1 1 1 1 1 1 1 0 0 0] as presented in Table I.

Building the error vector requires to compare the expected output and actual output of the software; if the output is the expected one, the error vector is set to 0 in the test case position, otherwise, it is set to 1. For Fig. 1(a), the expected output is true because the target element 3 is indeed present in the set $\{1, 2, 3, 4, 5\}$. Observing the last executed statement $S-12$ and, given the Java code, one can see that the test case execution returned the expected result. Therefore, the error vector is set to 0 in the respective position. Similarly, for Fig. 1(b), the expected output is also true due to the presence of the target element 2 in the set; however, one can see that the execution path finishes in $S-6$, which is a variable declaration. The fault activation in $S-10$ results in a negative value of variable ix; ix is then used as index of set in $S-6$ causing an array-out-of-bounds exception given the attempt to access a negative index in an array. Therefore, the test case has an unexpected output and the error vector is set to 1 in the respective position (see Table I).

SFL benefits from both spectrum and error vector to localize the faulty statement. Generically, SFL assumes the hypothesis that closely correlated components are more likely to be relevant to an observed failure. In practice, the basic idea is that comparing transactions over multiple runs and then computing the suspiciousness values of components can indicate which of these is the most likely to be the faulty one. Resemblances between binary vector (e.g., error vector) and nominally scaled data (e.g., spectrum) are quantified by means of similarity coefficients. These coefficients measure similarity essentially using dichotomy terms, which for our illustrative example refers to the involvement of statement j in the test case i (A_{ij}) and result of test case i (e_i), formally defined as

$$c_{pq}(j) = \left|\left\{i | A_{ij} = p \wedge e_i = q\right\}\right| : p, q \in \{0, 1\}. \quad (1)$$

Considering $S-8$'s column ([0 0 1 1 0 0 0]) and error vector ([0 0 0 0 0 1 1]), we illustrate the mechanism of filling dichotomy terms. First, second, and fifth positions of both vectors have value 0; therefore $c_{00} = 3$. In the third position, the involvement column has value 1 whereas error vector has value 0, producing $c_{10} = 1$. Forth position of both vector has value 1, thus $c_{11} = 1$. Finally, $c_{01} = 2$ because, in sixth and seventh positions, involvement column has value 0 and error vector has value 1. Table II shows the dichotomy terms of our example.

Several SFL methods use different formulas of similarity coefficients to compute these suspiciousness values. In this paper, an exhaustive list of 42 heuristics (also known as similarity coefficients) [3], [21] has been studied, focusing on the context of fault localization in software agents. For the sake of organization, Table VIII (in the Appendix) presents the formulas for all similarity coefficients.

TABLE II
VALUES OF DICHOTOMY TERMS FOR SFL EXAMPLE

Dichotomy Element	Statement												
	1	2	3	4	5	6	7	8	9	10	11	12	13
c_{11}	3	0	3	3	3	3	3	1	3	3	0	0	0
c_{10}	4	1	3	3	2	2	2	1	2	0	2	2	1
c_{01}	0	3	0	0	0	0	0	2	0	0	3	3	3
c_{00}	0	3	1	1	2	2	2	3	2	4	2	2	3

TABLE III
JACCARD SIMILARITY COEFFICIENT VALUES AND RANK FOR SFL EXAMPLE

S	1	2	3	4	5	6	7	8	9	10	11	12	13
Value	0.43	0.0	0.5	0.5	0.6	0.6	0.6	0.25	0.6	1.0	0.0	0.0	0.0
Rank	8	10	6	7	2	3	4	9	5	1	11	12	13

Let us finish the diagnosis process of the example by using the Jaccard coefficient (C_{16}) to compute suspiciousness values of each statement (see Table III). We substitute dichotomy values of $S-8$ into Jaccard formulas as follows:

$$C_{16} = \frac{c_{11}}{c_{11} + c_{10} + c_{01}} = \frac{1}{1+1+2} = 0.25. \quad (2)$$

For the illustrative example, by computing suspiciousness values and ranking statements with respect to them (see Table III), SFL (correctly) identifies statement 10 as the most likely location of the fault. Clearly, this example has a small number of test cases and statements, nonetheless it fully illustrates the manner SFL works.

IV. CONCEPTS AND DEFINITIONS FOR SFL IN MAS

An MAS is a computational system composed by a set of agents. Each of them (agents) is able to autonomously reason as well as sense and act upon a certain environment, leading to the satisfaction of its own (and sometimes shared) goals. Agents are time-persistent entities, that is, they are continuously operating to fulfill their goals.

Definition 1 (Multiagent System): An MAS consists of a set of agents AGS = $\{Ag_1, \ldots, Ag_m, \ldots, Ag_M\}$ that are situated in an environment E. Thus, an MAS is a tuple MAS = $\langle E, \text{AGS}\rangle$ [22].

The MAS runs during a limited time interval. We consider discrete time (represented by nonnegative integers) because: 1) continuous time can be sampled and 2) we intend to work within the boundaries of finite-dimensional events.[2] Throughout this paper, we refer to an unit of time either by using time frame or time step.

Example 1: In order to explain the basic concepts and the application of SFL in MASs, we make use of a running example. Fig. 2 shows this MAS, which is borrowed from our experimental setup that will be thoroughly described

[2]A finite-dimensional event depends only on the values of the process at finitely many time points [23].

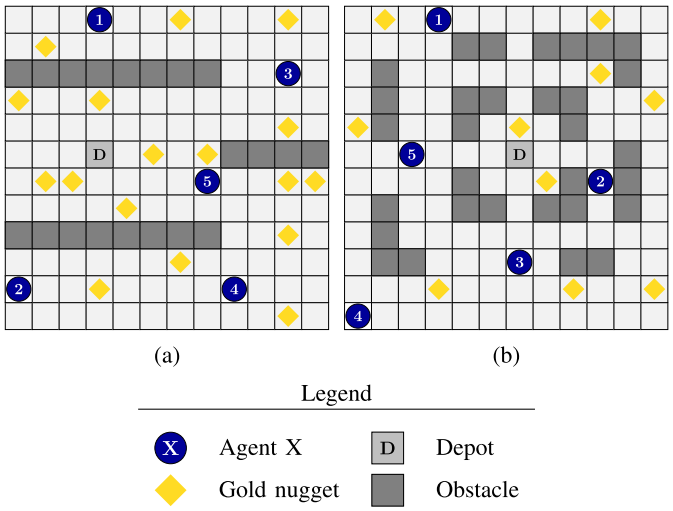

Fig. 2. Goldminers example used as the running example. (a) Test case 1. (b) Test case 2.

in Section VI. For the sake of clearness, we have reduced the number of agents and size of the environment of this example. In a nutshell, agents find themselves exploring an area searching for the gold nuggets spread over the environment. They aim at collecting as much gold nuggets as they can and delivering them to a depot where the nuggets are safely stored. Let us assume that agent Ag_5 erroneously compute its distance from a gold nugget due to an unforeseen bug in the reasoning process unintentionally left by the designer/programmer. As a result of this bug, Ag_5 has lower performance in some specific situations than it should have. Throughout this paper, it is shown how to use SFL to pinpoint the faulty agent.

After defining the basic MAS elements, it is needed to assess its performance by measuring observable variables of both the environment and the set of agents. For instance, an MAS aiming to control traffic lights uses environment variables such as traffic flow to measure MAS performance. Likewise, utility-based MASs measure the global utility (which is commonly used as a system performance measure) by summing the utility of all agents, that is, applying a function over observable variables of agents. In our running example, the MAS performance can be measured by the amount of gold nuggets in the depot.

Definition 2 (Measurable Space): A measurable space is a set Z, together with a nonempty collection $\mathcal{Z}$, of subsets of Z, satisfying the following two conditions.

1) For any X, Y in the collection $\mathcal{Z}$, the set $X \cap Y^c$[3] is also in $\mathcal{Z}$.
2) For any $X_1, X_2, \ldots, \in \mathcal{Z}$.

The elements of $\mathcal{Z}$ are called measurable sets of variables.

Definition 3 (Measure of MAS): Let $(E, \mathcal{E})$ and (AGS, $\mathcal{AGS}$) be two measurable spaces. A measure of MAS performance consists of a two nonempty subsets $M_E \subset \mathcal{E}$ and $M_{\text{AGS}} \subset \mathcal{AGS}$ together with $\mu : f(\langle \mathcal{E}, \mathcal{AGS} \rangle, n) \rightarrow \mathbb{R}$ that maps the performance of an MAS at time n to a real number. It uses as arguments the measurable set of variables of the tuple $\langle E, \text{AGS} \rangle$.

[3] $X \cap Y^c$ is the set of all values of X that are not in Y.

There is a strong connection between MAS and environment (Definition 1) termed by Ferber [24] as "the agent/environment duality." Thereby, we formally define our understanding of an environment, which introduces the necessary components to define test cases for MASs.

Definition 4 (Environment): An environment E is described as a tuple $E = \langle S, s_0, \mathcal{A} \rangle$, where:

1) $S = \{s_0, s_1, \cdots\}$ is a countable set of environment states with an initial state s_0;
2) $\mathcal{A} = \times_{Ag_m \in \text{AGS}} \mathcal{A}_{Ag}$ denotes all joint actions (of agents) that can be performed in the environment.

Definition 4 does not explicitly introduce sets of environment objects other than agents, since all (observable) information of these objects are considered to be incorporated into the set of environment states S. In our example, the depot and positions of gold nuggets are objects that are modeled within the environment state set.

A test case, in ordinary sequential programs, comprises input values and excepted output values. Defining the input and output of MASs is straightforward by means of the environment and the measure of MAS.

Definition 5 (Input, Output): Given an MAS situated in an environment E and measured by μ, then the inputs comprise the initial state s_0 of E and initialization parameters for agents (ags_0). The outputs comprise the measure μ of MAS.

With this definition of input and output, we are able to define a test case for an MAS and its evaluation.

Definition 6 (Test Case): Given an MAS, then a tuple $\langle I, O \rangle$ is a test case for MAS if and only if:

1) I is a set of values for each object specifying the environment state s_0 and a set of initialization parameters ags_0 for every agent $Ag_m \in \text{AGS}$;
2) O is a set of values specifying expected MAS outputs when measured by μ.

In our setting, test case evaluation works as follows. First, the environment is initialized with state s_0 and agents with parameters ags_0. Subsequently, the MAS is executed. The outputs computed by μ are compared with the expected values stated in the test case. If the computed output value is not equivalent to the expected value, the MAS fails the test case during that specific period. Otherwise, the MAS passes the test case during a specific period. Unlike in "traditional" software, running an MAS with a test case yields a vector of errors where each element is the passed/failed MAS status at time n.

Example 2: A test case for our running example from Fig. 2(a) is I = {depot = $(4, 6)$, gold = $\{(1, 4), (2, 2), (2, 7), (3, 7), (4, 4), (4, 11), (5, 8), (6, 6), (7, 1), (7, 10), (8, 6), (11, 1), (11, 5), (11, 7), (11, 9), (11, 12), (12, 7)\}$, obstacle = $\{(1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3), (8, 3), (1, 9), (2, 9), (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 6), (10, 6), (11, 6), (12, 6)\}$, $\text{ags}_0 = \{(4, 1), (1, 11), (11, 3), (9, 11), (8, 7)\}$, and $O = (1 * \text{collectedgold})/n$. The MAS has to be executed during a period so it is possible to establish when it passes and/or fails the test case.

Avižienis *et al.* [25] defined fault-related terms as: a failure is an event that occurs when delivered service deviates from

correct service; an error is a system state that may cause a failure; and a fault is the cause of an error in the system. In this paper, we consider that faults in agents' behavior depend on a given context, i.e., on how each agent interprets that particular situation [26]; and, mainly, these faults are a systemic matter as it might affect the overall performance [27]. More specifically, the term "faulty agent" is used to refer to an agent that either is not healthy and needs to be repaired or has been induced to a failure state (known as cascading effect).

Diagnosis is the task of pinpointing the faulty component that led to symptoms of malfunctioning (failure/error). In software, the set of components can be defined at any granularity level: a class, a function, or a block. The lower the granularity level gets, the more focused is the diagnosis; even though such low granularities require more computational effort [28]. Hence, following Reiter's formalism [29], the diagnosis problem for MASs can be defined as follows.

Definition 7 (Multiagent Diagnosis Problem): Given an MAS that has a set of agents AGS and a set of observations OBS for test case $\langle I, O\rangle$, then the diagnosis problem is to find the faulty agent which is responsible for the mismatch between the expected MAS performance and the observed one.

The multiagent diagnosis problem is defined with granularity at the agent level and thus considering agents as black boxes. This is a fair assumption when different parties implement agents reasoning and do not completely share their knowledge and/or architecture. On the one hand, the technique proposed in this paper is not able to identify the specific bug inside the code of the faulty agent. On the other hand, however, it has the advantage of being agnostic to programming languages and agent architectures.

V. Extending SFL for Multiagent Systems

In this section, we describe our approach, called ESFL-MAS. As stated by Definition 7, this paper deals with agent-level diagnosis, because it focuses on testing the MAS as a whole ensuring that agents' performance is nominal when facing several environmental conditions. We envision that, at run-time, the agent-level is the most advisable one as faulty agents are simply removed from the running MAS to avoid performance degradation. The first step is to map MAS concepts into the aforementioned SFL elements of SFL and, due to the previous discussion, such SFL components are the agents themselves. As for transactions, since MASs must run during some period (several time frames) to observe their emerging behavior, ESFL-MAS considers the error status of an agent's behavior at time n as a transaction.

The hit spectrum is the far most commonly used type of spectrum [4]. It encodes the component's activity in terms of involved/not involved in a given test case. A limitation of the SFL approach is related to the high level of abstraction enforced by hit spectrum as it does not provide useful information about the state of the agent during an execution. Since agents are time-persistent entities, they are constantly active and acting upon the environment; this creates spectra with very low entropy. Low entropy in the spectrum means that there

n	A					e
	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5	
1	1	0	1	1	1	1
2	1	1	1	1	1	0
3	1	1	0	0	0	1
4	0	0	0	0	0	0
5	0	0	0	0	1	1

(a)

n	A					e
	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5	
1	1	1	1	1	1	1
2	1	1	1	1	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	1
5	0	1	0	0	1	1

(b)

n	A					e
	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5	
1	0	0	0	0	0	0
2	1	0	1	1	0	0
3	0	1	0	1	1	1
4	0	0	1	0	1	1
5	1	1	1	1	1	1

(c)

n	A					e
	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5	
1	1	1	1	1	0	0
2	1	0	0	1	1	1
3	1	1	0	1	1	1
4	0	0	0	0	1	0
5	0	0	0	0	0	0

(d)

Fig. 3. Collection of performance spectra for both test cases 1 and 2 ($I = 2$) with $J = 2$. (a) $(A, e)_{1,1}$. (b) $(A, e)_{1,2}$. (c) $(A, e)_{2,1}$. (d) $(A, e)_{2,2}$.

is less useful information and, consequently, decreasing SFL diagnostic quality [30]. Therefore, hit spectra is not suitable to MASs. This conclusion can be generalized to other types of spectrum that do not account for time within transactions.

The solution proposed to overcome this limitation is to encode each agent's performance in terms of expected/not expected during a time step. An agent's health state is either expected (when it is performing as expected) or unexpected (when it is not performing as expected). A detection of symptoms (also called error detection) phase is responsible to infer any behavioral violation from system observations [31]. Specifically for agents, this can be done using several methods from monitoring agent's utility to applying anomaly detection techniques. Detecting an unexpected behavior does not necessarily mean that one has identified the agent that is causing the system failure to occur [32]. ESFL-MAS pinpoints the faulty agent so the designer is able to fix it, which is also essential to improve reliability of MASs. Given this performance-oriented perspective, we propose the performance spectra, where the error detection phase generates the set of data composing the spectrum. It is worthwhile mentioning that error detection mechanisms are outside the scope of this paper.

Definition 8 (Performance Spectrum): Let N denote the number of passing and failing time frames. Let N_f and N_p, $N_f + N_p = N$, denote the number of fail and pass sets, respectively. The performance spectrum abstraction encodes the agents' activity in terms of expected/unexpected behavior at the nth time frame and the error vector (i.e., transactions' correctness) in terms of MAS performance has passed/failed the test case at nth time frame. Using the performance spectrum, A and e are defined as

$$A_{nm} = \begin{cases} 1, & \text{if agent } m \text{ performed an unexpected} \\ & \text{behavior at time } n \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$e_n = \begin{cases} 1, & \text{if MAS had an unexpected output} \\ & \text{at time } n \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

Given that the pair (A, e) highly depends on both the environment settings and the agents' autonomy, SFL is limited to catch multiple instances of both dependencies. This limitation is addressed as follows. First, to solve the problem of multiple environment settings, one must run the MAS for different environment and agent settings; thus, the ESFL-MAS collects performance spectra referring to several test cases. Second, to solve the agent's autonomy problem, one must execute the MAS J rounds of the same test case to ensure that the collected spectra cover as many agents' activation paths (i.e., choices) as possible.

Definition 9 (Collection of Performance Spectra): A collection of performance spectra for a test case i is denoted by $\text{PS}_i = \{(A, e)_{i,1}, \ldots, (A, e)_{i,j}, \ldots, (A, e)_{i,J}\}$, which has been executed J rounds. Additionally, $\text{PS} = \{\text{PS}_1, \ldots, \text{PS}_i, \ldots, \text{PS}_I\}$ denotes the collection of all performance spectra, where I is the number of available test cases.

Example 3: We execute two rounds of the MAS of Example 1 for test cases 1 and 2 [see Fig. 2(a) and (b)], i.e., $I = 2$ and $J = 2$. Assuming that there is an error detection mechanism able to detect unexpected behavior in the agents, the collection of performance spectra $\text{PS} = \{(A, e)_{1,1}, (A, e)_{1,2}, (A, e)_{2,1}, (A, e)_{2,2}\}$ presented in Fig. 3 (ignoring the highlighted columns that will be explained later on) is built.

We have observed that agents are constantly monitored over time but do not consistently fail. For this reason, the collected performance spectra have several time frames in which either every agent performed an expected behavior or every agent performed an unexpected behavior. Both events contain no information for SFL because only the variability of transactions in the spectrum contributes toward improving diagnostic quality. A proposed optimization to ESFL-MAS, named MAS filter, recognizes these aforementioned events and filters them from the performance spectra to increase quality of the diagnosis process. Conceptually, when excluding the nonuseful time frames, the entropy value of the spectrum tends to its optimal value and consequently increases diagnostic accuracy. The MAS filter is defined as follows:

$$\text{MAS filter}(A, e) = (A_x, e_x) : x = \left\{n \mid \exists i, j : A_{ni} \neq A_{nj}\right\}. \quad (5)$$

The filter is based on the set x, which is the set of all spectrum's rows for which exist elements with both values 1 and 0. Intuitively, it filters time steps in which all agents have expected (or unexpected) behaviors detected.

Example 4: Following our running example, the MAS filter excludes the highlighted rows of the four pairs (A, e) shown in Fig. 3, creating a filtered version for each of them that has higher entropy than the original spectrum.

For each agent, a dichotomy matrix is then created (see Table IV). One-dimension of this matrix is related to the amount of time steps in which the agent had an unexpected behavior detected, and the other is the passed/failed MAS status determined by the test case expected output. Concisely, c_{11} is the number of time steps in which an agent performed an unexpected behavior and MAS was detected as failed, c_{10} is the number of time steps in which an agent performed an unexpected behavior and the MAS was considered correct, c_{01} is the number of time steps in which an agent did not perform an unexpected behavior and the MAS was found faulty, and c_{00} is the number of time steps in which an agent did not perform an unexpected behavior and the MAS was detected correct. Formally

$$c_{pq}(m) = |\{n | A_{nm} = p \wedge e_n = q\}| : p, q \in \{0, 1\}. \quad (6)$$

Each element in the table is a counter that will be used to calculate suspiciousness values in ESFL-MAS.

Example 5: Fig. 4 presents the dichotomy tables of our running example. Each dichotomy term is computed only taking into account the nonhighlighted rows of the performance spectra in Fig. 3.

SFL abstractly models each agent using a weak-fault model, meaning that the spectrum-based diagnosis essentially consists in identifying the agent whose transactions resembles the error vector the most. The suspiciousness values calculated by a similarity coefficient quantify these resemblances, under the assumption that a high similarity with the error vector indicates a high suspiciousness for that agent to have caused the detected failure. SFL process results in the diagnostic report.

Definition 10 (Diagnostic Report): A diagnostic report is an ordered set D of agents sorted with respect to their suspiciousness of being the true fault explanation.

Example 6: Let us continue to identify the faulty agent of our running example. ESFL-MAS uses the information of

TABLE IV
DICHOTOMY TABLE FOR PERFORMANCE SPECTRUM

MAS status	Behaviour of Ag_m: Unexpected ($A_{nm} = 1$)	Excepted ($A_{nm} = 0$)
Failed ($e_n = 1$)	c_{11}	c_{01}
Passed ($e_n = 0$)	c_{10}	c_{00}

	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
c_{11}	2	1	1	1	2
c_{10}	0	0	0	0	0
c_{01}	1	2	2	2	1
c_{00}	0	0	0	0	0

(a)

	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
c_{11}	0	1	0	0	1
c_{10}	1	2	1	2	0
c_{01}	1	0	1	1	0
c_{00}	1	0	1	0	2

(b)

	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
c_{11}	0	1	1	1	2
c_{10}	1	0	1	1	0
c_{01}	2	1	1	1	0
c_{00}	0	1	0	0	1

(c)

	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
c_{11}	2	1	0	2	2
c_{10}	1	1	1	1	1
c_{01}	0	1	2	0	0
c_{00}	1	1	1	1	1

(d)

Fig. 4. Dichotomy tables for the running example. (a) $(A, e)_{1,1}$. (b) $(A, e)_{1,2}$. (c) $(A, e)_{2,1}$. (d) $(A, e)_{2,2}$.

TABLE V
JACCARD SIMILARITY COEFFICIENT VALUES AND RANK FOR THE RUNNING EXAMPLE

Agent	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
Coefficient Value	0.36	0.36	0.18	0.33	0.78
Rank (D)	2	3	5	4	1

dichotomy tables from Fig. 4 to compute suspiciousness values using the similarity coefficient. In this example, we choose the Jaccard coefficient (C_{16}). ESFL-MAS computes the suspiciousness value by inserting the information of dichotomy terms into the formula, e.g., for agent Ag_5

$$C_{16} = \frac{c_{11}}{c_{11} + c_{10} + c_{01}} = \frac{7}{7 + 1 + 1} = 0.78. \quad (7)$$

The process is repeated for every agent until it obtains the values shown in Table V. Afterward, the list of agents is sorted in descending order with respect of suspiciousness values. As expected, the faulty agent Ag_5 was the highest ranked by ESFL-MAS in the end of the process.

Algorithm 1 programmatically describes the ESFL-MAS procedure. The goal of the algorithm is to identify the most probable faulty agent in given MAS based on their detected behavioral violations. It takes as input a collection of performance spectra PS and returns a diagnostic report D. In contrast to the SFL algorithm, the set of performance spectra is built using an error detection mechanism that is independent of ESFL-MAS and it is outside the scope of this paper as well.

In practice, the algorithm works by iteratively selecting an spectrum (A, e) from PS (line 3) and then filtering the nonuseful rows through MAS filter. It iterates over the number of time steps N and the number of agents M (lines 4–6) and check whether A_n has a difference among its elements (line 7). If it has, the control flag skip is set to false (line 8) and the search for a different element is interrupted. Finally, it passes to the next spectrum's row if skip is true (lines 10 and 11). Next, it iterates over the row to feed the dichotomy terms (lines 12–15), which are the input to the similarity coefficient s (line 16). ESFL-MAS returns the diagnostic report with the most probable faulty agent based on the suspiciousness values of each agent (lines 17 and 18).

Algorithm 1: ESFL-MAS

```
   Input: Collection of Performance Spectra PS
   Output: Diagnostic report D
1  begin
2      c = [0]_{2×2×M}
3      for (A_{N×M}, e_{N×1}) ∈ PS do
4          for n ∈ {0, ··· , N − 1} do
5              skip ← true
6              for m ∈ {0, ··· , M − 2} do
7                  if A_nm ≠ A_{n(m+1)} then
8                      skip ← false
9                      break
10         if skip then
11             continue
12         for m ∈ {0, ··· , M − 1} do
13             p = A_nm
14             q = e_n
15             c_pqm = c_pqm + 1
16     d ⟵ s(c)
17     D ⟵ Sort(d)
18     return D
```

VI. EXPERIMENTAL SETUP

The empirical assessment herein presented aims to discover the set of similarity coefficients that yields the best results in terms of diagnostic quality. This section describes the test suite used in the experiments, the data extraction process, and the metric used to evaluate the ESFL-MAS performance.

A. Test Suite

We use an instance of the PDP [33] to test our approach because: 1) it is well-known and 2) it is a real-world representative problem. Problems of this kind are highly dynamic and decisions have to be made under uncertainty and incomplete knowledge. Briefly, it consists of mobile vehicles retrieving and delivering a set of items. The PDP is a well-studied, NP-hard problem and, given its inherent distribution and decentralization, MASs offer an interesting solution for PDP [34]. Examples of PDP solved through agents include ride-sharing services [35], application of mobile robots in missions [36], and automated guided vehicle employed in many industrial environments [37].

The second edition of the multiagent programming contest (MAPC)[4] [38] provides an instance of PDP known as the GoldMiners scenario in which its multiple agents work under uncertainty and responsive situations, i.e., new destinations become available in real-time and are immediately eligible for consideration. GoldMiners implements fundamental concepts of MASs, such as autonomy, team-work coordination, high-level interaction, as well as partial and local perception of the environment. Another reason to use the MAPC's implementation of GoldMiners is that all components composing the system (which include world model, agents' reasoning, and interaction protocols) was previously tested and validated by the MAS community. For our purposes, this assure (in a higher degree of confidence) that only our injected faults will contribute to dysfunctional behaviors.

From several MASs available in the MAPC, we chose the one programmed in AgentSpeak [39], an agent-oriented programming language, while agents were run using Jason [40], an interpreter for an extended version of AgentSpeak. The choice was made given our previous experience using AgentSpeak [41] and the fact that the Jason team won the second edition of MAPC.

The MAS aims to solve the PDP by finding a schedule that delivers as many items as possible at the lowest cost

[4]The MAPC is an annual competition that, since 2005, aims at providing suitable test suite offering key problems for the community to test agent-oriented programming approaches.

while cooperating in a dynamic environment. The environment is a grid-like world where agents can move to a neighbor cell. Agents explore the environment avoiding obstacles and collecting gold nuggets. They also can communicate and coordinate their actions in order to collect as much gold nuggets as possible and to deliver them to the depot where they can be safely stored. In addition, agents have only a local view of the environment, their perceptions might be incomplete, and the executed action may fail.

Research on MASs encompasses different aspects of their artifacts (agents, environment, and so forth) and functioning (organization, type of social behavior, knowledge representation, and so forth). As aforementioned, ESFL-MAS is able to diagnose the faulty agent regardless its architecture and knowledge representation. Nevertheless, MAS organization and type of social behavior (mainly cooperation) may impact on ESFL-MAS performance. First, when cooperation exists, agents work in high synergy which might contribute to very similar choices influencing the performance spectrum. Second, when there is no organization, agents do not have strict roles in the MAS and as so agents' choices might more easily jeopardize another agent's performance.

Hence, seeking completeness of the test suite and based on previous work in MAS organizations [42], we implement a modified version of the Jason implementation of MAPC's GoldMiners. Specifically, the original Jason implementation relied on a twofold strategy: first, *a priori* allocation of agents' search quadrants and, second, a team-work coordination aiming to find and carry gold nuggets to the depot. Modified MASs vary both in the coordination and in the spatial organization (resource allocation) dimensions resulting in the following types of MAS.

1) *Noncoordinated and Nonorganized (NCNO):* Agents work individually (not cooperatively) and do not receive a search quadrant (loose spatial organization).
2) *Noncoordinated and Organized (NCO):* Agents work individually but each of them has an assigned search quadrant.
3) *Coordinated and Nonorganized (CNO):* Agents coordinate the gold-nuggets search, yet there is no allocated quadrant.
4) *Coordinated and Organized (CO):* Agents coordinate the gold-nuggets search as well as have an assigned search quadrant.

As modules responsible for interaction among agents and organizational composition are independent from the agent reasoning and other MAS functions, the modified versions inherit the reliability from the original implementation.

Given these baseline (correct) versions, simulating real faults offer a more direct way to assess the proposed technique. These faults can be hand-seeded or seeded by mutation through rules (called mutation operators). We used both of these strategies for different purposes. Hand-seeded faults aim to emulate dysfunctional behaviors specifically for the aforementioned strategy implemented by the Jason Team. Moreover, faults seeded by mutation rules automatically build a set of validated faulty versions as we have used mutation operators proposed by Huang *et al.* [43]. In this paper, we have used the (as called by the authors) high-level mutation operators for Jason.

TABLE VI
DESCRIPTION OF TYPE OF FAULTS—HIGHLIGHTED ROWS REPRESENT THE HAND-SEEDED FAULTS AND THE OTHERS ARE GENERATED THROUGH MUTATION OPERATORS. USE OR NOT OF FAULTS IS REPRESENTED BY $\surd$ AND $\times$ RESPECTIVELY

Qnt.	Fault Description	NCNO	NCO	CNO	CO
1	Agent disrespects its search quadrant	×	√	×	√
1	Agent conceals nugget positions	×	×	√	√
1	Agent has a delayed response	√	√	√	√
1	Agent gets stuck in a specific goal	√	√	√	√
1	Agent gets the farthest gold nuggets.	√	√	√	√
3	Delete a belief in the agent.	√	√	√	√
3	Delete a plan in the agent.	√	√	√	√
3	Delete the condition part of a rule.	√	√	√	√
3	Replace the triggering event operator.	√	√	√	√
3	Delete the context of a plan.	√	√	√	√
3	Delete the body of a plan.	√	√	√	√
3	Delete a formula in the body plan.	√	√	√	√
3	Swap adjacent formulae in a plan.	√	√	√	√
3	Replace the operator of a goal.	√	√	√	√
3	Replace receivers in a message.	×	×	√	√
3	Replace the illocutionary force.	×	×	√	√
3	Delete a propositional in a message.	×	×	√	√

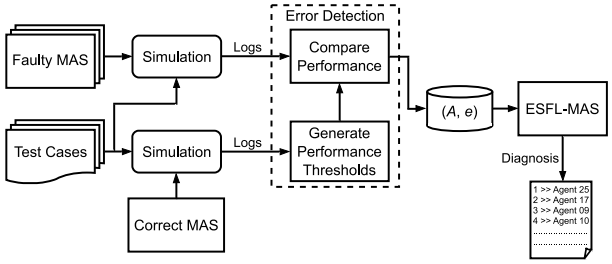

Fig. 5. Experimental phases.

Table VI gives an overview of the faulty versions in the test suite. Each of these faulty versions contains a single injected fault. In the test suite, we were not able to use all created faults for all types of MASs. For instance, fault 2 cannot be applied to noncooperative MASs because they inherently do not broadcast any gold nugget positions.

This test suite covers homogeneous and closed MASs in which agents co-habit in an uncertain environment and may organize to optimally allocate resources and/or interact to establish a team cooperation. Several MASs with such features have been used to solve real-world problems such as those aforementioned in the MASs for PDPs [35]–[37] and we can add traffic control [44] and shop-floor management [45].

B. Data Acquisition

A two-step process (shown in Fig. 5) generates the spectra required by the experiments. In the first stage, we collect simulation logs, while in the second, we train an error detection mechanism. Then, such simulation logs are used to generate the required performance spectra.

1) Collecting Logs: An MAS initially configured according to a given test case is executed to obtain logs from both agents

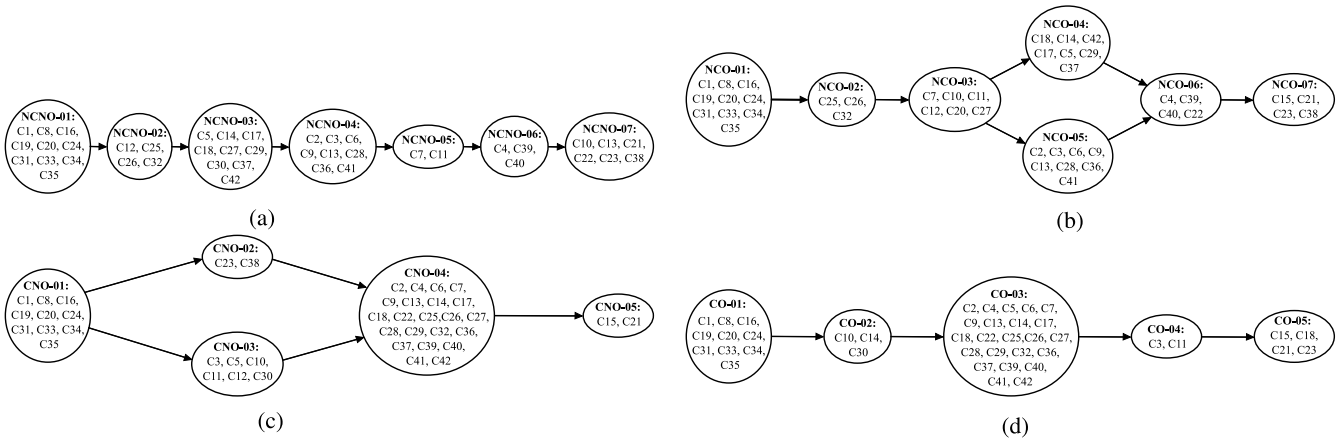

Fig. 6. Similarity coefficients grouped by their quality of diagnosis: each node corresponds to a group; edges indicate relationships between groups such that A → B means "group A requires less effort to diagnose than group B"; those with the same vertically alignment present less than 1% difference in the mean quality of diagnosis. (a) NCNO MAS. (b) NCO MAS. (c) CNO MAS. (d) CO MAS.

and the overall system. These logs were collected for every test case and they contained the amount of gold nuggets carried by each agent and the total amount of nuggets in the depot for each time step of a simulation.

For the experimental setup, we randomly generated five test cases and each of them corresponded to a set of initial positions for: all agents, the depot, and all 400 gold nuggets. Aiming to collect information to generate spectra, the MAS (composed by 25 agents) was executed 75 times for each test case during 1000 time steps.

2) Expected MAS Performance and Error Detection: Both test cases and performance spectra demand for measurements to verify the overall MAS and agent correctness. Agents and the MAS are, respectively, measured by: 1) the amount of carried gold nuggets in each time frame and 2) the amount of gold nuggets in the depot. However, we still need to define the expected MAS performance and error detection mechanism.

As Goldminers can be seen as an instance of PDPs, the MAS expected output is the average number of gold nuggets in the depot for each time frame. We run the correct MAS version and compute the performance baseline. While assessing faulty versions, time steps with performance values above the baseline are marked as passed ($e_n = 0$) and below as failed ($e_n = 1$).

ESFL-MAS does not directly get spectra from the involvement of agents in a test case and thus an error detection phase is necessary in order to generate the performance spectra. Thus, we emulated an error detection phase in our experiments to assess ESFL-MAS, even though these mechanisms are not within the scope of this paper. Error detection for miner agents is done similarly to calculation of expected MAS performance. We compute the average amount of gold nuggets carried by each agent in a certain time frame and used this value as baseline to detect whether the agent is performing as expected ($A_{nm} = 0$) or not ($A_{nm} = 1$) for time n, therefore mapping collected logs to performance spectra.

C. Evaluation Metric

As ESFL-MAS returns a list of agents sorted by their suspiciousness values, diagnostic performance is expressed in terms of diagnostic quality (also referred as accuracy) that evaluates how many agents need to be inspected before the faulty agent is found. If other agents have the same similarity coefficient as the faulty agent, we use the average ranking position for these agents. Diagnostic quality is defined as

$$Q = \left(1 - \frac{|\{j|S_j > S_f\}| + |\{j|S_j \geq S_f\}| - 1}{2(M-1)}\right) * 100\% \quad (8)$$

where S_j and S_f denote the suspiciousness value for agent j and for the faulty agent, respectively, and M is the total number of agents in the system. Intuitively, the $|\{j|S_j > S_f\}|$ term represents the number of agents ranked in front of the faulty agent, whereas $|\{j|S_j \geq S_f\}|$ represents the number of agents with same or higher value than the suspiciousness value of the faulty one. This metric assumes that, on average, half of agents with same suspiciousness values will be inspected until reaching the fault.

As an example, consider the diagnostic report in Table V. Supposing that the correct diagnostic candidate (i.e., the faulty agent) is Ag_2. In order to calculate the accuracy, we start by examining Ag_5 finding that it is healthy. Due Ag_5 being healthy, we pass to the next agent in the diagnostic report. Examining Ag_2, we observe that it is the faulty agent. The diagnostic quality is therefore

$$Q = \left(1 - \frac{1+3-1}{2(5-1)}\right) * 100\% = 62.5\%. \quad (9)$$

VII. Experimental Results

As this paper investigates an exhaustive list of similarity coefficients and some of them behave very similarly to others, we decided to perform a clustering analysis. Hierarchical clustering was used to construct the dendrogram [46], together with bootstrapping analysis as no assumption about the diagnostic accuracy distribution could be made. Several hierarchical clustering approaches (with different linkage and distances metrics) yielded similar dendrograms. However, the results discussed in this paper were obtained using the weighted average as linkage [46] and the uncentered Pearson correlation as distance [47], because they generate statistically significant results with marginal standard error (SE). In order to

TABLE VII
MEAN ACCURACY FOR EACH SIMILARITY COEFFICIENT

Group	Diagnostic Quality ($\overline{Q}$ (σ) [%])			
	NCNO	NCO	CNO	CO
01	96.25 (11.27)	95.16 (12.68)	97.08 (10.35)	96.54 (10.74)
02	73.96 (31.15)	67.74 (27.45)	55.83 (39.88)	66.23 (36.46)
03	53.61 (22.98)	53.27 (27.02)	54.10 (31.77)	49.95 (1.628)
04	47.90 (12.90)	47.87 (17.81)	50.00 (0.00)	44.92 (21.33)
05	43.54 (19.61)	46.19 (14.26)	8.358 (19.67)	27.36 (38.17)
06	36.85 (21.58)	37.47 (21.39)	-	-
07	22.08 (27.95)	23.32 (31.46)	-	-

select the number of clusters by cutting the generated tree, we used the approximately unbiased test [48], which corrects the bias caused by bootstrap, with 95% confidence (0.001 SE). In this way, coefficients within the same cluster present low intercluster distances and high intracluster distances.

The resulting groups for the NCNO, NCO, CNO, and CO types of MAS are shown in Fig. 6(a)–(d), respectively. The number of groups (and their elements) is not the same in every type of MAS. There are seven groups for NCNO and NCO whereas five are the groups for CNO and CO. The noncoordinated versions tend to have more disperse results than coordinated versions. This occurs because noncoordinated agents rely solely on their own capability to perceive the environment, where coordinated agents have an expanded perception through their peers. Furthermore, some clusters yield similar diagnostic quality, but they are statistically different, that is, they generate different diagnostic reports. These clusters appear horizontally aligned in Fig. 6.

After carefully analyzing the clusters, two fundamental aspects can explain their composition. First, from the mathematical formulas of coefficients, there is a logical causal relationship between accuracy and the dichotomy matrix: the more significance a coefficient assigns to anomalous behavior for which a system's error has been detected (represented by c_{11}), the better the diagnostic quality. Second, the test suite has reduced capacity of representing cascading effects of undesired behavior; for instance, the faulty agent sending the wrong location of a gold nugget for a correct agent inducing the latter in failure. Accordingly, coefficients that give greater emphasis on time steps where MAS was correct and an agent had an unexpected behavior detected have lower diagnostic quality.

The overall mean diagnostic quality for the test suite, along with the standard deviation (σ), are presented in Table VII. Each cell in the table corresponds to a cluster in Fig. 6; for instance, group NCO-03 is in the column NCO and row 03 of the table. Groups NCNO-01, NCO-01, CNO-01, and CO-01 yield the best diagnostic accuracy with 96.25%(11.27%), 95.16%(12.68%), 97.08%(10.35%), and 96.54%(10.74%), respectively. This means that the user has to inspect approximately 4% of the agents to find the faulty one. These groups also show low standard deviation (σ), excluding groups CNO-04 and CO-03. Actually, the results of groups CNO-04 and CO-03 are masked because similarity coefficients within these groups compute the same suspiciousness values for every

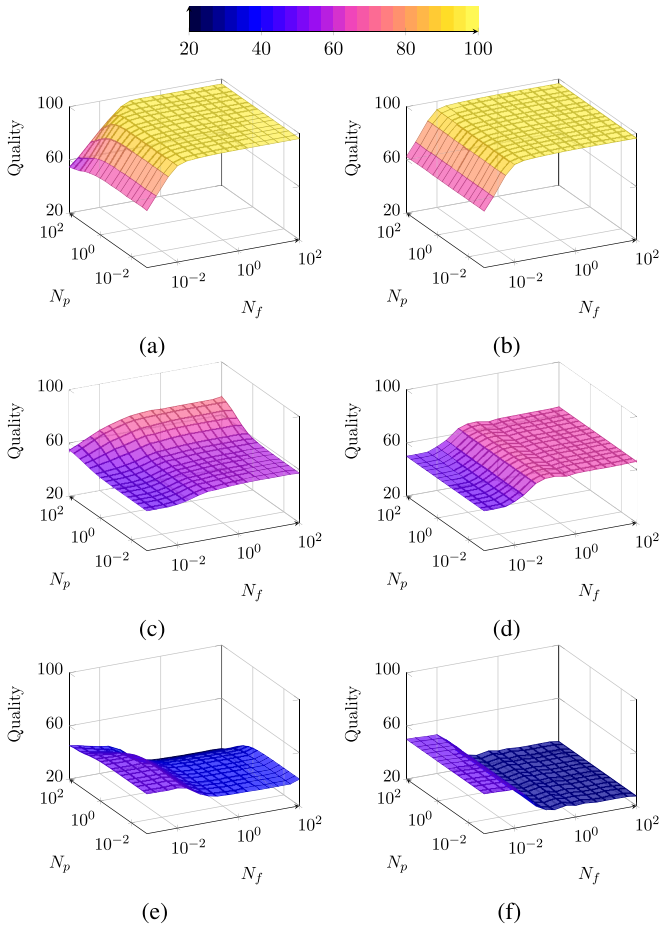

Fig. 7. Impact of observation quantity. (a) NCNO-01. (b) CO-01. (c) NCNO-02. (d) CO-02. (e) NCNO-07. (f) CO-05.

agent in the diagnostic report and, as assumed by the evaluation metric, the faulty agent is in the middle of the list, resulting in the approximate accuracy of 50% and standard deviation 0%. In practice, these results are not useful and are not considered in our analysis.

A. On the Impact of Observation Quantity

In the previous results, we have assumed that there is enough time to run the MAS several rounds under different conditions to collect a considerable amount of measurements. In practice, however, the tester works under short-time constraints or the tester does not want to extensively assess the system. To investigate the influence of the amount of available data on ESFL-MAS performance, we evaluate Q while varying the number of passed (N_p) and failed time steps (N_f) that are available. The spectrum time-line is not rearranged during the variation of both N_p and N_f; the process here is to randomly exclude nonconsecutive time steps. We did not want to compromise the consistency of our data, even though ESFL-MAS assumes that time steps are independent. Since the ratio of failed and passed time steps is very small (nearly 0.02) and no previous experiment analyzing the ESFL-MAS sensitivity have been performed, we study the influence of the quantity of available data on the diagnostic accuracy Q across

the entire range of available data. Thus, N_p and N_f are varied from 0.001% to 100% of the total number of passed and failed time steps and results are presented in logarithmic scale.

Fig. 7 shows such evaluations of groups NCNO-01, CO-01, NCNO-02, CO-02, NCNO-07, and CO-05. They allow a comparison of the ESFL-MAS behavior across the two most different MAS versions as well as of groups of the same MAS version. For each graph, we averaged Q over 1000 randomly selected combinations of passed and failed time steps until the variance in the measured values of Q is negligible.

Concerning the number of erroneous time steps N_f, we confirm from Fig. 7 that adding failed time steps improves the diagnostic quality for most case. The benefit of including more than 200 N_fs for groups NCNO-01 [Fig. 7(a)] and CO-01 [Fig. 7(b)] is marginal on average. This number increases to 2000 N_fs for groups NCNO-02 [Fig. 7(c)] and CO-02 [Fig. 7(d)]. Conversely, coefficients in groups NCNO-07 [Fig. 7(e)] and CO-05 [Fig. 7(f)] lose performance when including more failed time steps. This happens once these similarity coefficients are inversely proportional to c_{11} and c_{01}.

Concerning the number of correct time steps N_p, results for each version are quite different. As for CO MASs, N_p did not influence the ESFL-MAS accuracy. This phenomenon can be explained by the presence of coordination among agents: when agents work as team-mates in an organized manner, the MAS performs as a "well-oiled machine" and an agent that fails under these circumstances is more easily detected and therefore more easily correlated with the system failure. As for NCNO MASs, correct time steps can have effects on the diagnostic quality: 1) slightly degrade the ESFL-MAS performance for $N_f < 0.1\%$ [see Fig. 7(a)]; 2) positively influence the ESFL-MAS results across N_f dimension [see Fig. 7(c)]; and 3) decrease ESFL-MAS accuracy for $N_f > 0.1\%$ [see Fig. 7(e)]. These sparse results are consequence of the chaos created by autonomic decisions.

B. On the Impact of Error Detection Precision

Error detection is the phase that precedes diagnosis and as such it has a great impact on the ESFL-MAS' diagnostic quality. The following experiment shows how precision in detecting error affects diagnostic quality for each similarity coefficient. For any realistic system and practical error detection mechanism, there will very likely exist errors that go undetected, mainly because of two reasons. First, the faulty agent only jeopardizes system operation under specific scenario settings. For instance, let us assume that a miner agent, erroneously, is not able to perceive gold nuggets; no error is detected unless the agent gets near a gold nugget. Second, analogously to faults in software, errors induced by agents might not propagate all the way to system failures and thus go undetected.

As consequence of these issues, the number of rows in spectra, in which both faulty agent and system fail, will only be a fraction of the total rows in which the agent fails. More intuitively, this proportion represents the EDP, that is, how precisely the error detection phase is able to correlate a system

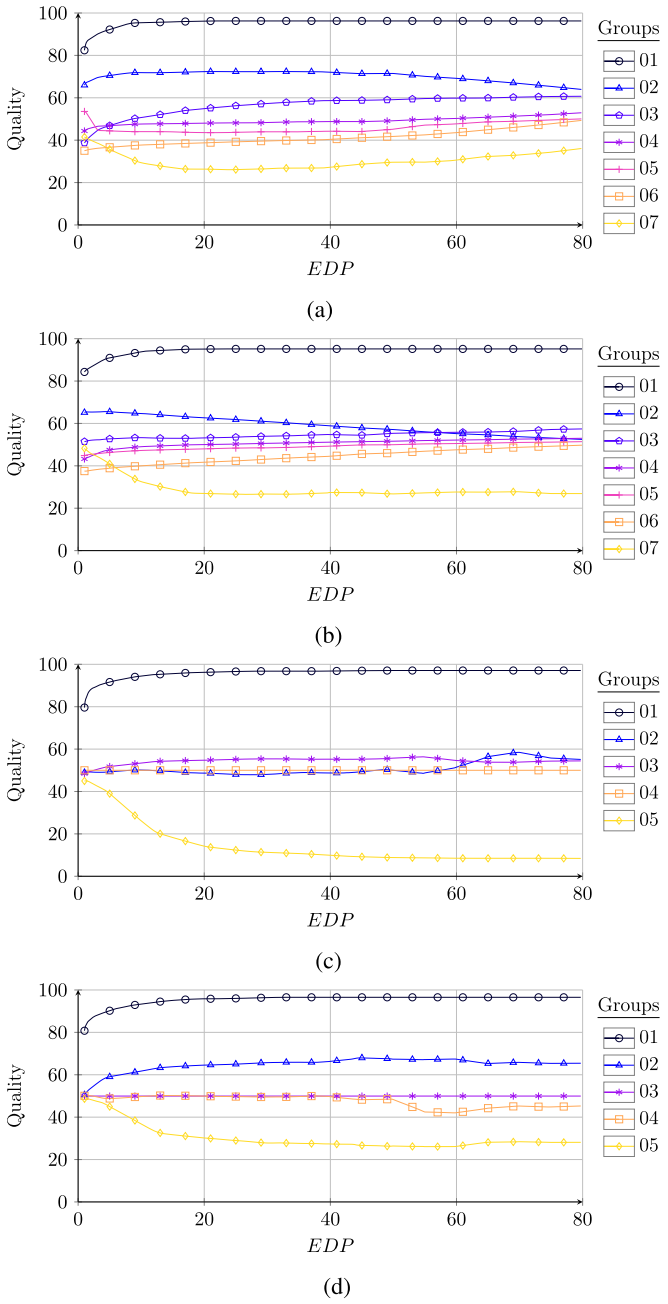

Fig. 8. EDP for all MAS versions. (a) NCNO MAS. (b) NCO MAS. (c) CNO MAS. (d) CO MAS.

failure with the faulty agent. Using the previous notation, we define

$$\text{EDP} = \left(\frac{c_{11}(f)}{c_{11}(f) + c_{10}(f)}\right) * 100\% \tag{10}$$

where f is the location of the faulty agent. By varying the EDP ratio, we are assessing the response of the ESFL-MAS diagnostic quality for different error detection mechanisms.

Each faulty version of our test suite has an inherent value for EDP fluctuating from 3.31% to 97.77%. We vary EDP using two methods: 1) excluding time steps that activate the faulty agent, but for which no system error has been detected decreasing $c_{10}(f)$, and increasing EDP and 2) excluding time

steps that activate the faulty agent and for each an system error has been detected decreasing $c_{11}(f)$, and decreasing EDP. In order to unbias our experiment, we randomly sample passed and failed time steps from the set of available ones to control EDP within a 95% confidence interval. Yet, similarly to the experiment of observation quantity impact (Section VII-A), this random selection of passed and failed time steps maintained their sequence as the original spectrum to sustain consistency. These time step exclusions can be seen in practice as incomplete monitoring on agents (instrumentation blindness). The experiments serve well to demonstrate whether or not the approach is robust to such incompleteness.

Fig. 8 shows how the diagnostic quality changes with respect to the EDP. One can see that, on average for all cases, a detection precision greater than 40% have marginal contribution to a better fault diagnosis. This does not mean that the community needs to give up improving error detection techniques; this means that, when coupled with a diagnosis phase, error detection needs a solid (but nonoptimal) performance. Moreover, we confirm group 01 as the best set of similarity coefficients for MAS also regarding EDP variation. We show that ESFL-MAS can achieve high accuracy even for low EDP being the borderline EDP $\geq 10\%$.

C. Discussion

From observation quality impact, the N_p effect over ESFL-MAS performance degrades as agents work cooperatively in an organized fashion. Hence, our results suggest that a single faulty agent can more easily be pinpointed in a team organization (without significant cascading errors) rather than in a selfish MAS. Additionally, adding N_f improves the diagnostic quality until 200 time frames when its contribution starts to become marginal.

Regarding the effect of the EDP, ESFL-MAS shows to be robust for a broad range of EDP and groups NCNO-01, NCO-01, CNO-01, and CO-01 produced the best and more stable performance being their near-optimal response achieved when EDP $\geq 10\%$.

Experiments suggest that ESFL-MAS accuracy might be jeopardized by cascading faults produced by highly interacting agents. Furthermore, our experiments determined that the best similarity coefficients for ESL-MAS, are accuracy, coverage, Jaccard, Laplace, least contradiction, Ochiai, Rogers and Tanimoto, simple-matching, Sorensen-dice, and support yield the best results in our experiment.

In agreement with our findings, literature also reports the Ochiai coefficient has yielded the best diagnostic quality when SFL is applied to diagnose faults in practical domains [3]–[5]. This suggests that Ochiai is a good candidate to be incorporated in a practical implementation of our ESFL-MAS.

Several granularity levels might be considered to diagnose faults in MASs. The proposed ESFL-MAS approach works at the granularity of agent behavioral faults, since its aim is to increase system reliability at run-time. A finer granularity may help designers to better debug agents' internal behavior. However, to diagnose agents at this level of granularity, information regarding, e.g., rules and/or plans, is needed. As a result, other instrumentation infrastructure has to be put in place to support such a level of granularity. Note, however, that the approach proposed in this paper suits all types of granularities well.

VIII. Conclusion

MASs are constantly susceptible to several threats that can jeopardize their nominal performance. To detect any abnormality on agent behaviors is a paramount step to ensure performance; however, in practice, error detection rarely exhibits 100% precision. Moreover, when there are multiple cooperative agents, errors might be masked by other agents and possibly go undetected. As stated by Kalech and Kaminka [32], "diagnosis is an essential step beyond the detection of the failures. Mere detection of a failure does not necessarily lead to its resolution." Motivated by this, we devised and discussed the ESFL-MAS, which is able to identify agents that may jeopardize the overall performance through runtime profiles of the system while requiring minimal information about it.

We mapped MAS concepts to basic elements of SFL, being agents and their expected behavior at given time, respectively, mapped to components and transactions, thus incorporating time within the spectrum. Diagnosis at the system level steer the proposed technique toward a performance-oriented direction, resulting in the so-called performance spectrum. At last, some data do not effectively contributed to localize the faulty agent given the intensive monitoring of agents; thus, we suggested MAS filter that increases diagnostic accuracy by excluding low-entropy spectrum's rows.

The ESFL-MAS performance greatly depends on particular factors, namely: 1) similarity coefficient; 2) quantity of observations; and 3) quality of error detectors. These dependencies have been thoroughly studied by the empirical assessments, which yielded prominent results giving a good prospect for the ESFL-MAS application. Results show that accuracy, coverage, Jaccard, Laplace, least contradiction, Ochiai, Rogers and Tanimoto, simple-matching, Sorensen-dice, and support yield the best diagnostic accuracy for the created test suite. They yield roughly 96.26% diagnostic accuracy and are stable when varying both EDP and quantity of observations.

This novel application of spectrum-based diagnosis in MASs opens up room for new research directions. Regarding the exploration of other granularity levels, on the one hand, a following work is to instrument Goldminers scenario at the AgentSpeak code level. However, solely extending the instrumentation is not sufficient to apply SFL to this finer-grained level. Out of the four proposed extensions, one might have to consider three, namely: 1) encoding time within a transaction; 2) collecting data across different agents' choices; and 3) applying the MAS filter. Assuming that SFL approach indeed excels at agent code level diagnosis, an automatic debugging tool could be integrated within the native one of Jason's, providing a more practical application of the proposed technique. On the other hand, the agent-level diagnosis would

TABLE VIII
SIMILARITY COEFFICIENTS AND THEIR FORMULAS

#	Coefficient	Formulae
C_1	Accuracy	$P(XY) + P(\overline{X}\overline{Y})$
C_2	Added Value	$max\left(P(Y\|X) - P(Y), P(X\|Y) - P(X)\right)$
C_3	Anderberg	$\frac{P(XY)}{P(XY)+2(P(XY)+P(X\overline{Y}))}$
C_4	Certainty Factor	$max\left(\frac{P(Y\|X)-P(Y)}{1-P(Y)}, \frac{P(X\|Y)-P(X)}{1-P(X)}\right)$
C_5	Collective Strength	$\frac{P(XY)+P(\overline{X}\overline{Y})}{P(X)P(Y)+P(\overline{X})P(\overline{Y})} \times$ $\times \frac{1-P(X)P(Y)-P(\overline{X})P(\overline{Y})}{1-P(XY)-P(\overline{X}\overline{Y})}$
C_6	Confidence	$max\left(P(Y\|X), P(X\|Y)\right)$
C_7	Conviction	$max\left(\frac{P(X)P(\overline{Y})}{P(X\overline{Y})}, \frac{P(Y)P(\overline{X})}{P(Y\overline{X})}\right)$
C_8	Coverage	$P(X)$
C_9	Example and Counterexample	$1 - \frac{P(X\overline{Y})}{P(XY)}$
C_{10}	Gini Index	$max(P(X)[P(Y\|X)^2 + P(\overline{Y}\|X)^2]+$ $+ P(\overline{X})[P(Y\|\overline{X})^2 + P(\overline{Y}\|\overline{X})^2]-$ $- P(Y)^2 - P(\overline{Y})^2, P(Y)[P(X\|Y)^2 +$ $+ P(\overline{X}\|Y)^2] + P(\overline{Y})[P(X\|\overline{Y})^2+$ $+ P(\overline{X}\|\overline{Y})^2] - P(X)^2 - P(\overline{X})^2)$
C_{11}	Goodman and Kruskal	$\sum_i \sum_j \frac{max_j P(X_i Y_j) + max_i P(X_i Y_j)}{2 - max_i P(X_i) - max_j P(Y_j)} -$ $- \sum_i \sum_j \frac{max_i P(X_i) + max_j P(Y_j)}{2 - max_i P(X_i) - max_j P(Y_j)}$
C_{12}	Information Gain	$(-P(Y)logP(Y) - P(\overline{Y})logP(\overline{Y}))-$ $- (P(X) \times (-P(Y\|X)logP(Y\|X))-$ $- P(\overline{Y}\|X)logP(\overline{Y}\|X) - P(\overline{X})\times$ $\times (-P(Y\|\overline{X})logP(Y\|\overline{X}))-$ $- P(\overline{Y}\|\overline{X})logP(\overline{Y}\|\overline{X}))$
C_{13}	Interest	$\frac{P(XY)}{P(X)P(Y)}$
C_{14}	Interestingness Weighting Dependency	$\left(\left(\frac{P(XY)}{P(X)P(Y)}\right)^k - 1\right) P(XY)^m$, where k, m are coefficients of dependency and generality respectively
C_{15}	J-Measure	$max(P(XY)log\left(P(Y\|X)/P(Y)\right) +$ $+ P(X\overline{Y})log(P(\overline{Y}\|X)/P(\overline{Y})),$ $P(XY)log\left(P(X\|Y)/P(X)\right) +$ $+ P(\overline{X}Y)log(P(\overline{X}\|Y)/P(\overline{X})))$
C_{16}	Jaccard	$\frac{P(XY)}{P(X)-P(Y)-P(XY)}$
C_{17}	Kappa	$\frac{P(XY)+P(\overline{X}\overline{Y})-P(X)P(Y)-P(\overline{X})P(\overline{Y})}{1-P(X)P(Y)-P(\overline{X})P(\overline{Y})}$
C_{18}	Klosgen	$\sqrt{P(XY)} \times max(P(Y\|X) - P(Y),$ $P(X\|Y) - P(X))$
C_{19}	Laplace	$max\left(\frac{P(XY)+1}{P(X)+2}, \frac{P(XY)+1}{P(Y)+2}\right)$
C_{20}	Least Contradiction	$\frac{P(XY)-P(X\overline{Y})}{P(Y)}$
C_{21}	Leverage	$P(Y\|X) - P(X)P(Y)$
C_{22}	Loevinger	$1 - \frac{P(X)P(\overline{Y})}{P(X\overline{Y})}$
C_{23}	Normalized Mutual Information	$\frac{\sum_i \sum_j P(X_i Y_j) log_2 \frac{P(X_i Y_j)}{P(X_i)P(Y_j)}}{-\sum_i P(X_i) log_2 P(X_i)}$
C_{24}	Ochiai	$\frac{P(XY)}{\sqrt{P(X)P(Y)}}$
C_{25}	Ochiai II	$\frac{P(XY)+P(\overline{X}\overline{Y})}{\sqrt{(P(XY)+P(\overline{X}\overline{Y}))(P(XY)+P(X\overline{Y})}} \times$ $\times \frac{1}{\sqrt{(P(\overline{X}\overline{Y})+P(\overline{X}Y))(P(\overline{X}\overline{Y})+P(X\overline{Y}))}}$
C_{26}	Odd Multiplier	$\frac{P(XY)P(\overline{Y})}{P(Y)P(X\overline{Y})}$
C_{27}	Odds Ratio	$\frac{P(XY)P(\overline{X}\overline{Y})}{P(X\overline{Y})P(\overline{X}Y)}$
C_{28}	One-way Support	$P(Y\|X) log_2 \frac{P(XY)}{P(X)P(Y)}$
C_{29}	Piatetsky-Shapiro	$P(XY) - P(X)P(Y)$
C_{30}	Relative Risk	$\frac{P(Y\|X)}{P(Y\|\overline{X})}$
C_{31}	Rogers and Tanimoto	$\frac{P(XY)+P(\overline{X}\overline{Y})}{P(XY)+P(\overline{X}\overline{Y})+2(P(X\overline{Y})+P(\overline{X}Y))}$
C_{32}	Sebag-Schoenauer	$\frac{P(XY)}{P(X\overline{Y})}$
C_{33}	Simple-Matching	$P(XY) + P(\overline{X}\overline{Y})$
C_{34}	Sorensen-Dice	$\frac{2P(XY)}{2P(XY)+P(\overline{X}Y)+P(X\overline{Y})}$
C_{35}	Support	$P(XY)$
C_{36}	Tarantula	$\frac{P(XY)/P(Y)}{P(X\overline{Y})/P(\overline{Y})+P(XY)/P(Y)}$
C_{37}	Two-way Support	$P(XY) log_2 \frac{P(XY)}{P(X)P(Y)}$
C_{38}	Two-way Support Variation	$P(XY)log_2 \frac{P(XY)}{P(X)P(Y)} +$ $+ P(\overline{X}Y)log_2 \frac{P(\overline{X}Y)}{P(\overline{X})P(Y)} +$ $+ P(X\overline{Y})log_2 \frac{P(X\overline{Y})}{P(X)P(\overline{Y})} +$ $+ P(\overline{X}\overline{Y})log_2 \frac{P(\overline{X}\overline{Y})}{P(\overline{X})P(\overline{Y})}$
C_{39}	Yule's Q	$\frac{P(XY)P(\overline{X}\overline{Y})-P(X\overline{Y})P(\overline{X}Y)}{P(XY)P(\overline{X}\overline{Y})+P(X\overline{Y})P(\overline{X}Y)}$
C_{40}	Yule's Y	$\frac{\sqrt{P(XY)P(\overline{X}\overline{Y})}-\sqrt{P(X\overline{Y})P(\overline{X}Y)}}{\sqrt{P(XY)P(\overline{X}\overline{Y})}+\sqrt{P(X\overline{Y})P(\overline{X}Y)}}$
C_{41}	Zhang	$\frac{P(XY)-P(X)P(Y)}{max(P(XY)P(\overline{Y}),P(Y)P(X\overline{Y}))}$
C_{42}	ϕ-coefficient	$\frac{P(XY)-P(X)P(Y)}{\sqrt{P(X)P(Y)(1-P(X))(1-P(Y))}}$

suite well to be applied hierarchically to first discover faulty agents and then focus the diagnosis on wrong conditions in a rule (potentially using model-based software debugging).

Another research path would be to encompass more information about the MAS besides spectra. More specifically, collaboration graphs [19] and/or matrix representation of coordination [17] (both gathered by monitoring communication) provide valuable information about social structures that together with spectrum-based reasoning techniques [4] could more easily (i.e., less test cases and executions) localize faults in MASs with greater diagnostic accuracy.

APPENDIX
SIMILARITY COEFFICIENTS FORMULAS

The mathematical formulas for computing each coefficient are shown in Table VIII. In this context, the variable X and Y are the 2-D of our dichotomy matrix, corresponding to healthy state of the agent and status of the MAS performance, respectively. The table also uses standard notation from probability and statistics, namely $P(X)$ is the probability of X, $P(\overline{X})$ is the probability of not X, and $P(XY)$ is the joint probability of X and Y. ESFL-MAS gathers frequencies rather then probabilities, but

TABLE IX
DEFINITIONS OF THE PROBABILITIES USED BY COEFFICIENTS IN THE CONTEXT OF THIS PAPER

$$P(X) = \frac{c_{11}+c_{10}}{c_{11}+c_{10}+c_{01}+c_{00}} \qquad P(\overline{X}) = \frac{c_{01}+c_{00}}{c_{11}+c_{10}+c_{01}+c_{00}}$$

$$P(Y) = \frac{c_{11}+c_{01}}{c_{11}+c_{10}+c_{01}+c_{00}} \qquad P(\overline{Y}) = \frac{c_{10}+c_{00}}{c_{11}+c_{10}+c_{01}+c_{00}}$$

$$P(XY) = \frac{c_{11}}{c_{11}+c_{10}+c_{01}+c_{00}} \qquad P(\overline{X}Y) = \frac{c_{01}}{c_{11}+c_{10}+c_{01}+c_{00}}$$

$$P(X\overline{Y}) = \frac{c_{10}}{c_{11}+c_{10}+c_{01}+c_{00}} \qquad P(\overline{XY}) = \frac{c_{00}}{c_{11}+c_{10}+c_{01}+c_{00}}$$

$$P(X|Y) = \frac{P(XY)}{P(Y)} \qquad P(Y|X) = \frac{P(XY)}{P(X)}$$

these can be easily substituted during actual computation (see Table IX).

ACKNOWLEDGMENT

The authors would like to thank A. N. Fernandes, L. A. Piedade, N. Cardoso, and Z. Kokkinogenis for the useful discussion during the development of this paper.

REFERENCES

[1] C. D. Nguyen, A. Perini, C. Bernon, J. Pavón, and J. Thangarajah, "Testing in multi-agent systems," in *Agent-Oriented Software Engineering X* (LNCS 6038), M.-P. Gleizes and J. J. Gomez-Sanz, Eds. Heidelberg, Germany: Springer, 2011, pp. 180–190.

[2] M. Fisher, R. H. Bordini, B. Hirsch, and P. Torroni, "Computational logics and agents: A road map of current technologies and future trends," *Comput. Intell.*, vol. 23, no. 1, pp. 61–91, 2007.

[3] B. Hofer, A. Perez, R. Abreu, and F. Wotawa, "On the empirical evaluation of similarity coefficients for spreadsheets fault localization," *Autom. Softw. Eng.*, vol. 22, no. 1, pp. 47–74, 2015.

[4] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *J. Syst. Softw.*, vol. 82, no. 11, pp. 1780–1792, 2009.

[5] T.-D. B. Le, F. Thung, and D. Lo, "Theory and practice, do they match? A case with spectrum-based fault localization," in *Proc. 29th IEEE Int. Conf. Softw. Maint. (ICSM)*, Eindhoven, The Netherlands, Sep. 2013, pp. 380–383.

[6] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi, "An empirical investigation of program spectra," in *Proc. ACM SIGPLAN-SIGSOFT Workshop Program Anal. Softw. Tools Eng. (PASTE)*, Montreal, QC, Canada, 1998, pp. 83–90.

[7] P. Zoeteweij, J. Pietersma, R. Abreu, A. Feldman, and A. J. C. Van Gemund, "Automated fault diagnosis in embedded systems," in *Proc. 2nd Int. Conf. Secure Syst. Integr. Rel. Improvement (SSIRI)*, Yokohama, Japan, 2008, pp. 103–110.

[8] S. Yoo, X. Xie, F.-C. Kuo, T. Y. Chen, and M. Harman, "No pot of gold at the end of program spectrum rainbow: Greatest risk evaluation formula does not exist," Dept. Comput. Sci., Univ. College London, London, U.K., Tech. Rep. RN/14/14, 2014.

[9] L. S. Passos, R. Abreu, and R. J. F. Rossetti, "Sensitivity analysis of spectrum-based fault localisation for multi-agent systems," in *Proc. 25th Int. Workshop Prin. Diagn. (DX)*, Graz, Austria, Sep. 2014, pp. 1–8.

[10] L. S. Passos, R. Abreu, and R. J. F. Rossetti, "Spectrum-based fault localisation for multi-agent systems," in *Proc. 24th Int. Joint Conf. Artif. Intell. (IJCAI)*, Buenos Aires, Argentina, Jul. 2015, pp. 1134–1140.

[11] P. J. F. Lucas, "Analysis of notions of diagnosis," *Artif. Intell.*, vol. 105, nos. 1–2, pp. 295–343, 1998.

[12] B. Horling, B. Benyo, and V. Lesser, "Using self-diagnosis to adapt organizational structures," in *Proc. 5th Int. Conf. Auton. Agents*, Montreal, QC, Canada, Jun. 2001, pp. 529–536.

[13] C. Dellarocas and M. Klein, "An experimental evaluation of domain-independent fault handling services in open multi-agent systems," in *Proc. 4th Int. Conf. MultiAgent Syst.*, Boston, MA, USA, 2000, pp. 95–102.

[14] F. de Jonge, N. Roos, and C. Witteveen, "Primary and secondary diagnosis of multi-agent plan execution," *Auton. Agents Multi-Agent Syst.*, vol. 18, no. 2, pp. 267–294, 2009.

[15] R. Micalizio and P. Torasso, "Cooperative monitoring to diagnose multiagent plans," *J. Artif. Intell. Res.*, vol. 51, no. 1, pp. 1–70, Sep. 2014.

[16] M. Kalech and G. A. Kaminka, "On the design of coordination diagnosis algorithms for teams of situated agents," *Artif. Intell.*, vol. 171, nos. 8–9, pp. 491–513, 2007.

[17] M. Kalech, "Diagnosis of coordination failures: A matrix-based approach," *Auton. Agents Multi-Agent Syst.*, vol. 24, no. 1, pp. 69–103, 2012.

[18] G. A. Kaminka, D. V. Pynadath, and M. Tambe, "Monitoring teams by overhearing: A multi-agent plan-recognition approach," *J. Artif. Intell. Res.*, vol. 17, pp. 83–135, Aug. 2002.

[19] E. Serrano, A. Muñoz, and J. Botía, "An approach to debug interactions in multi-agent system software tests," *Inf. Sci.*, vol. 205, pp. 38–57, Nov. 2012.

[20] F. Koca, H. Sözer, and R. Abreu, "Spectrum-based fault localization for diagnosing concurrency faults," in *Testing Software and Systems* (LNCS 8254), H. Yenigün, C. Yilmaz, and A. Ulrich, Eds. Heidelberg, Germany: Springer, 2013, pp. 239–254.

[21] L. Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi, "Extended comprehensive study of association measures for fault localization," *J. Softw. Evol. Process*, vol. 26, no. 2, pp. 172–219, 2014.

[22] T. Lettmann, M. Baumann, M. Eberling, and T. Kemmerich, "Modeling agents and agent systems," in *Transactions on Computational Collective Intelligence V* (LNCS 6910), N. T. Nguyen, Ed. Heidelberg, Germany: Springer, 2011, pp. 157–181.

[23] R. N. Bhattacharya and E. C. Waymire, *Stochastic Processes With Applications*. New York, NY, USA: Wiley, 1990.

[24] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Harlow, U.K.: Addison-Wesley, 1999.

[25] A. Avižienis, J.-C. Laprie, and B. Randell, "Dependability and its threats: A taxonomy," in *Building the Information Society* (IFIP International Federation for Information Processing), vol. 156, R. Jacquart, Ed. Boston, MA, USA: Springer, 2004, pp. 91–120.

[26] E. Platon, N. Sabouret, and S. Honiden, "A definition of exceptions in agent-oriented computing," in *Proc. 7th Int. Conf. Eng. Soc. Agents World VII (ESAW)*, Dublin, Ireland, 2007, pp. 161–174.

[27] M. Klein, J.-A. Rodriguez-Aguilar, and C. Dellarocas, "Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death," *Auton. Agents Multi-Agent Syst.*, vol. 7, nos. 1–2, pp. 179–189, Jul. 2003.

[28] T. Zamir, R. T. Stern, and M. Kalech. (2014). *Using Model-Based Diagnosis to Improve Software Testing*.

[29] R. Reiter, "A theory of diagnosis from first principles," *Artif. Intell.*, vol. 32, no. 1, pp. 57–95, Apr. 1987.

[30] A. Gonzalez-Sanchez, R. Abreu, H.-G. Gross, and A. J. C. Van Gemund, "Spectrum-based sequential diagnosis," in *Proc. 25th AAAI Conf. Artif. Intell.*, San Francisco, CA, USA, 2011, pp. 189–196.

[31] J. de Kleer and B. C. Williams, "Diagnosing multiple faults," *Artif. Intell.*, vol. 32, no. 1, pp. 97–130, 1987.

[32] M. Kalech and G. A. Kaminka, "Coordination diagnostic algorithms for teams of situated agents: Scaling up," *Comput. Intell.*, vol. 27, no. 3, pp. 393–421, 2011.

[33] M. W. P. Savelsbergh and M. Sol, "The general pickup and delivery problem," *Transp. Sci.*, vol. 29, no. 1, pp. 17–29, 1995.

[34] K. Fischer, J. P. Müller, and M. Pischel, "Cooperative transportation scheduling: An application domain for DAI," *Appl. Artif. Intell.*, vol. 10, no. 1, pp. 1–34, 1996.

[35] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," *Eur. J. Oper. Res.*, vol. 223, no. 2, pp. 295–303, 2012.

[36] J. L. Posadas, J. L. Poza, J. E. Simó, G. Benet, and F. Blanes, "Agent-based distributed architecture for mobile robot control," *Eng. Appl. Artif. Intell.*, vol. 21, no. 6, pp. 805–823, 2008.

[37] M. Grunow, H.-O. Günther, and M. Lehmann, "Dispatching multi-load AGVs in highly automated seaport container terminals," in *Container Terminals and Automated Transport Systems*, H.-O. Günther and K. Kim, Eds. Heidelberg, Germany: Springer, 2005, pp. 231–255.

[38] M. Dastani, J. Dix, and P. Novák, "The second contest on multi-agent systems based on computational logic," in *Computational Logic in Multi-Agent Systems* (LNCS 4371), K. Inoue, K. Satoh, and F. Toni, Eds. Heidelberg, Germany: Springer, 2007, pp. 266–283.

[39] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language," in *Agents Breaking Away* (LNCS 1038), W. Van de Velde and J. W. Perram, Eds. Heidelberg, Germany: Springer, 1996, pp. 42–55.
[40] R. H. Bordini, J. F. Hübner, and M. J. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*. Chichester, U.K.: Wiley, 2007.
[41] R. J. F. Rossetti *et al.*, "Using BDI agents to improve driver modelling in a commuter scenario," *Transp. Res. C Emerg. Technol.*, vol. 10, nos. 5–6, pp. 373–398, 2002.
[42] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms," *Knowl. Eng. Rev.*, vol. 19, no. 4, pp. 281–316, Dec. 2004.
[43] Z. Huang, R. Alexander, and J. Clark, "Mutation testing for Jason agents," in *Engineering Multi-Agent Systems* (LNCS 8758), F. Dalpiaz, J. Dix, and M. van Riemsdijk, Eds. Cham, Switzerland: Springer, 2014, pp. 309–327.
[44] L. S. Passos and R. J. F. Rossetti, "Traffic light control using reactive agents," in *Proc. 5th Iber. Conf. Inf. Syst. Technol. (CISTI)*, Santiago de Compostela, Spain, Jun. 2010, pp. 1–6.
[45] P. Leitão, J. Barbosa, and D. Trentesaux, "Bio-inspired multi-agent systems for reconfigurable manufacturing systems," *Eng. Appl. Artif. Intell.*, vol. 25, no. 5, pp. 934–944, 2012.
[46] L. L. McQuitty, "Similarity analysis by reciprocal pairs for discrete and continuous data," *Educ. Psychol. Meas.*, vol. 26, no. 4, pp. 825–831, 1966.
[47] K. Pearson, "Note on regression and inheritance in the case of two parents," *Proc. R. Soc. London*, vol. 58, nos. 347–352, pp. 240–242, 1895.
[48] H. Shimodaira, "Approximately unbiased tests of regions using multistep-multiscale bootstrap resampling," *Ann. Stat.*, vol. 32, no. 6, pp. 2616–2641, 2004.

Lúcio S. Passos received the B.Eng. (Hons.) degree in electrical engineering, specialized in telecommunication and electronics, from the Federal University of Uberlândia, Uberlândia, Brazil. He is currently pursuing the Ph.D. degree with the Faculty of Engineering, University of Porto, Porto, Portugal.

From 2008 to 2009, he was an Erasmus Exchange Student of Electrical Engineering with the Faculty of Engineering, University of Porto, and a Trainee Engineer with General Electric Company, Vila Nova de Gaia, Portugal. He is a Researcher with the Artificial Intelligence and Computer Science Laboratory, University of Porto.

Rui Abreu (M'06) received the M.Sc. degree in systems and computer engineering from the University of Minho, Braga, Portugal, carrying out his graduation thesis project at Siemens, Amadora, Portugal, and the Ph.D. degree from the Delft University of Technology, Delft, The Netherlands, in 2009.

From 2002 to 2003, he was an Erasmus Exchange Student of Software Technology with the University of Utrecht, Utrecht, The Netherlands. He was an Intern Researcher with Philips Research Laboratories, Eindhoven, The Netherlands, from 2004 to 2005. Since 2014, he has been an Applied Research Scientist with Palo Alto Research Center, Palo Alto, CA, USA. He is currently an Assistant Professor with the Faculty of Engineering, University of Porto, Porto, Portugal.

Rosaldo J. F. Rossetti (M'04) received the B.Eng. (Hons.) degree in civil engineering from UFC, Fortaleza, Brazil, in 1995, and the M.Sc. and Ph.D. degrees in computer science from UFRGS, Porto Alegre, Brazil, in 1998 and 2002, respectively.

He carried out his doctoral project as a Graduate Research Student with the Network Modelling Group, Leeds University's Institute for Transport Studies, Leeds, U.K. He is currently a Senior Research Fellow and a member of the Directive Board of the Artificial Intelligence and Computer Science Laboratory, and an Assistant Professor with the Department of Informatics Engineering, University of Porto, Porto, Portugal. His research interests include complex system analysis, behavioral modeling, social simulation, multiagent systems, spatiotemporal data analytics, application of multiagent systems as a modeling metaphor to address issues in artificial transportation systems, future mobility paradigms and urban smartification, industrial applications, behavior modeling, potential uses of serious games, and gamification in transportation and mobility systems.

Dr. Rossetti served as a Board of Governors Member of the IEEE ITS Society from 2011 to 2013. He is currently the Chair of the ATS and Simulation Technical Activities Sub-Committee, an Associate Editor of the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, a member of the ITS Podcast Editorial Board, an ITS Department Editor of the *IEEE Intelligent System Magazine*, and a member of the Steering Committee of IEEE Smart Cities Initiative, where he serves as the Editor-in-Chief of the *Smart Cities News Bulletin* and an Editor of *Readings on Smart Cities*. He is also a member of the Technical Committee on Intelligent Industrial Systems of the IEEE Systems, Man, and Cybernetics Society. He will be the General Chair for the 2016 IEEE 19th International Intelligent Transportation Systems Conference, the flagship conference of IEEE ITS Society. He is also a member of ACM, APPIA (Portuguese AI Society), and a Founder Member of the Portuguese-Brazilian Society for Modelling and Simulation.