Information Sciences 180 (2010) 561–583

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Debugging complex software systems by means of pathfinder networks

Emilio Serrano [a,*], Arnaud Quirin [b], Juan Botia [a], Oscar Cordón [b]

[a] Universidad de Murcia, Facultad de Informática, Campus Universitario de Espinardo, 30100 Murcia, Spain
[b] European Centre for Soft Computing, Edificio Científico-Tecnológico, 33600 Mieres, Asturias, Spain

ARTICLE INFO

Article history:
Received 25 March 2009
Received in revised form 14 September 2009
Accepted 3 November 2009

Keywords:
Multi-agent systems
Debugging
Forensic analysis
Pathfinder networks
Complex systems

ABSTRACT

This paper introduces a new methodology based on the use of *Pathfinder networks* (PFNETs) for the debugging of *multi-agent systems* (MASs). This methodology is specifically designed to develop a forensic analysis (i.e. a debugging process performed on previously recorded data of the MAS run) of MASs showing complex tissues of relationships between agents (i.e. a high complexity in their social level). Like previous works in the field of forensic analysis of MASs, our approach is performed by considering displays of the system activity which aim to be understandable by human beings. These displays allow us to understand the social behavior of the system, discover emergent behaviors, and debug possible undesirable behaviors. However, it is well known that the visualization of information in a humanly comprehensible way becomes a complex task when large amounts of information have to be represented, as is the case of the social behavior of large-scale MASs. Our methodology tackles this problem through the use of PFNETs, which are considered to reduce the data complexity in order to obtain simple representations that show only the most important global interactions in the system. In addition, the proposed methodology is customizable thanks to the use of two thresholds allowing the user to define the desired specificity level in the display. The proposal is illustrated with a detailed case study considering a complex customer–seller MAS.

1. Introduction

The term "multi-agent system" (or MAS) is applied to a system that has, among others, the following elements [16]: (1) an environment En, (2) a set of objects O, (3) an assembly of agents A with $A \subseteq O$, and finally (4) an assembly of relations R which link objects (and thus agents) to each other. A special case for systems in which $A = O$ and $En = \emptyset$ is a *purely communicating MAS*. In this case, the relations R define a network: each agent is directly linked to an assembly of other agents, which are called its *acquaintances*. These systems are very common in distributed artificial intelligence and they are characterized by the fact that the interactions are essentially intentional communications [16]. The interaction among agents is performed by complex dialogues because the agents are autonomous and therefore they should have the ability to refuse requests from other agents. For this reason, these interactions are complex and often unexpected [43]. This is also a consequence of the MAS' complex system nature, which causes sophisticated behaviors to arise out of a multiplicity of relatively simple interactions among the independent agents composing it (what is usually called emergency [22]).

If debugging traditional software is considerably complex, it is much more complex to debug a MAS in which intelligent or emergent behaviors may appear. Recent studies indicate the difficulties involved in testing and debugging

* Corresponding author. Tel.: +34 868 887882.
E-mail addresses: emilioserra@um.es (E. Serrano), arnaud.quirin@softcomputing.es (A. Quirin), juanbot@um.es (J. Botia), oscar.cordon@softcomputing.es (O. Cordón).

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.11.007

service-oriented applications [56], which are often carried out by agents. *Brute force* – the most common approach for software debugging as stated by Meyers [33] – is not advisable in this context (i.e. it usually consists of scattering statements throughout a program under test to display the values of variables). This is due to several facts such as the importance of software in society, the economic cost of the bugs, and the complexity in the process of debugging software, in general, and MASs, in particular. Therefore, MAS debugging requires the use of the latest technologies with the aim of automating and facilitate this complex process as much as possible.

Three main MAS debugging approaches have been considered in the specialized literature according to the MAS' level of abstraction tackled. All of them are based on recording and analyzing events of the system executions, thus they are categorized in the forensic analysis field [41]. The first approach [34] works by showing the recorded events to the programmer through a series of simple displays which must be studied by him in order to discover failures in the software. This operation mode is harshly criticized by Poutakidis et al. [43]. These researchers mainly argue that the displays present too much information to developers, making it difficult to understand what really happened in the system. The second approach [36] is based on defining the protocols which regulate the interactions between agents, testing such protocols automatically and debugging through some kind of display. This constitutes a more abstract vision of the MAS, considering it in its group level (i.e., the study of specific agent groups in the system without covering the whole society). The main problem with this approach is that it only debugs what has been defined previously, and a MAS usually presents emergent behaviors which will not be analyzed. The third approach [52] is focused on discovering knowledge in the event log and later displaying the extracted event insights in some way to give clues about what happened or might have gone wrong in the MAS execution. Note that, the debugging is no longer developed at the MAS group level in this third approach. Instead, it deals with the *MAS society level* (i.e., the level where the whole system is studied, not individual or group components [17]).

Some preliminary research based on data mining [14] has been carried out to attend to the debugging of a MAS at its social level [5,52]. The latter proposals present several advantages with respect to other analysis methods, so allowing the discovery of unexpected bugs. However, they also carry a series of complications and shortcomings. The first and most important one is that the user must have expertise in the basis of data mining and in the use of data mining tools. This is due to the fact that knowledge extraction is performed through an iterative process requiring user's feedback in a closed loop and that the data mining techniques considered are very sensitive to their parameter values.

The second problem is explained now. The display of the behavior of a complex system using any abstraction mechanism (which is the base of the latter three MAS debugging approaches) always comes with the problem of the information overload for the user [6,7]. In the MAS debugging context, this overload can mean the software developer gets lost in a large amount of useless information, due to complex representation, thus making him miss the really interesting items leading to debugging possible undesirable behaviors. In fact, obtaining user comprehensible displays is a key condition in certain critical systems like air traffic management [42]. Notice that the latter problem specially affects the third debugging approach since the amount of information handled is significantly larger than in the previous two.

One possible mechanism to reduce such complexity and to obtain proper displays of a MAS behavior could be the use of techniques based on social networks [50], specifically the *Pathfinder* algorithm [13]. This algorithm is aimed at generating a class of networks called *Pathfinder networks* (PFNETs) whose objective is to reveal the underlying organization of a system from a data sample. PFNETs are based on distance estimates or accurate measures between pairs of entities. The network only represents the main global relationships between the entities of the domain, allowing us to generate smaller and thus more comprehensible displays. PFNETs have already been used successfully in various fields such as psychology, to represent the cognitive structure of a subject [49,13], or as scientometrics, for the analysis of large scientific domains [7,32].

The contribution of this paper is a new forensic analysis methodology for the debugging of *large-scale MASs* (systems composed of hundreds of agents) at their social level. Our methodology is based on displays of the system activity which allow us to understand the social behavior of the system, discover emergent behaviors, and debug the possible undesirable behaviors. It aims to solve the two problems stated of these kinds of MAS debugging techniques, i.e. the obtaining of complex, non comprehensible displays due to the information overload and the need of user's data mining skills to apply them. To do so, we consider the use of PFNETs as a more formal and automatic alternative. Therefore, the obtained PFNET is going to reveal the most important global underlying interactions in the organization of a MAS. Starting from the data generated by the MAS itself after running a software test, PFNETs are used to reduce the complexity of that data in order to obtain simple displays which can be more easily understood by human beings. Moreover, the proposed methodology is customizable thanks to the use of two thresholds allowing the software developer to define the desired specificity level in the display. In this paper we focus the discussion on the debugging of purely communicating MAS. However, this does not mean that we restrict the work developed to only this kind of systems. The ideas presented in this paper are always valid for studying interactions between agents although the studied MAS is not purely communicating. Besides, this work can deal with the agent to environment interactions provided that they are typified. The taxonomy of the latter type of communications is currently the subject of some investigations [24].

The paper setup is presented as follows. Section 2 describes related works, explains the research trends in MAS debugging, and reviews the classic uses of PFNETs. Section 3 reviews the existing methods for debugging MAS at the social level. These methods are based on data mining techniques, and they present some deficiencies. This section also summarizes the basis of the proposal presented in this paper to relate it with the state of the art. Section 4 gives a short introduction to PFNETs and to the Pathfinder algorithm, detailing how the execution time can be optimized and how these nets should be displayed. Section

5 introduces our novel approach for large-scale MAS debugging based on PFNETs. Section 6 illustrates the operation of our methodology with a detailed case study. Finally, some conclusions and future work are given in Section 7.

2. Related works

2.1. Forensic analysis

Different approaches to debugging a MAS have often required recording relevant data from system executions. Therefore, MAS debugging can be categorized as forensic analysis (or postmortem analysis). Forensic analysis is the process of understanding, re-creating, and analyzing arbitrary events that have previously occurred [41].

We can find a number of works on forensic analysis applied to the field of intrusion detection. The underlying concept in these works is the same as for debugging a MAS using forensic analysis: recording the events of an execution, analyzing these events (sometimes using different tools) and finding evidences of intrusions into the events. The obvious difference is that instead of looking for intrusions, the developer tries to find bugs in the system. Therefore, there are important ideas and techniques in these works which can be used to debug MASs. For example, Kumar et al. show an approach that targets intrusion detection in computer networks and models intrusion patterns using Colored Petri–Nets [27]. The approach is very interesting but it still lacks an implementation that shows its efficiency. Dwyer et al. [15] have works where flow graphs are used to represent potential communication activity between the processes of a distributed system. Meanwhile, properties are represented using quantified regular expressions. This approach requires deep knowledge of tiny details in the processes of the system tested. Finally, a similar approach is used in the GrIDS tool [55]. This system is capable of detecting large-scale intrusion attacks on network systems. The most interesting part is that it builds activity graphs of the executions of the various processes in the system by monitoring them individually.

Serrano et al. have detailed specific techniques for the extraction of data needed for forensic analysis of MAS [51]. The basic problem in forensic analysis applied to MASs is the distributed nature of these systems. Agents can be working on several machines. They can even migrate from one computer to another. These authors try to define the infrastructure to record and sort the events in MASs. The proposal requires modifying the source code of the MAS platform. That is why aspect oriented programming [31] is used to get some genericity in the proposal motivated by the variety of existing MAS platforms.

2.2. Simple displays

The basics of forensic analysis allow the user to have techniques to record relevant data from MAS software tests. Nevertheless, in order to proceed with the analysis, data representation techniques are needed. Usually, such representation is based on graphical displays [42]. Such graphical displays have been used to represent the activity of single agents and groups of agents and, lately, of the whole society, as in our approach.

There is abundant research to debug at the agent level which provides interesting displays that could be used for the upper levels. One of the most illustrative examples is "*micro tool*" for MASs developed in the ZEUS platform [35]. It allows the user to look at the internal processing of an agent to see: (1) the messages being received by the agent, (2) the messages being sent out by the agent, (3) the actions taken in response to (1), and so on. Another example of debugging at the agent level is the "Mind Inspector" Tool [3] in the Jason platform. This tool allows the user to inspect the agents' internal (i.e., "mental") states. Another interesting work is the *Agent Viewing Tool* [4] for systems specified in Agent Factory Agent Programming Language. This tool provides a graphical interface that provides the developer with a number of views of an agents' internal state. The INGENIAS platform [40] also includes representations of the single agent's mental states among its debugging facilities. These displays represent the information contained within the mental state of any agent following the same notation used to specify the whole MAS in this platform. This contribution is aimed at the whole agents society. However, these previous works are very interesting for the early stages of developing a MAS, where agents typically have abundant specific bugs.

Nevertheless, testing single agents is not only limited to monitoring their internals. Single agents may also be tested by following the *unit testing* philosophy [21]. In this case, agents are seen as black boxes that should appropriately respond to specific messages. Mock agents are used [10] for this. A Mock agent does not belong to the MAS under testing. It is specifically designed to test agents in the MAS of interest. These agents are defined by the messages that should be sent to the agent under test and the messages that should be received from it. Therefore, the user is forced to consider the agent's interactions with other agents even though he is dealing with an isolated agent. Another way of looking at interactions by using single agents may be found in the *Tracer Tool* [28]. It uses recorded elements like beliefs, goals, intentions, messages, as categories for data and a concept graph is created with them. This graph is used as normative in the test. In consequence, all the logged data must comply with the relations indicated within the graph. Such data also include messages, so interactions are debugged at this level.

Displays to study the interactions of agents at the group level typically offer filtering of messages to reduce the shown data, but nothing more elaborated. An example of this approach is given by Ndumu et al. with the *society tool* [35] in the ZEUS platform. This tool allows a user to select a set of agents (it is a tool for the group level despite its name) and views the organizational relationships and the messaging between them. This allows users to identify bugs, either in the way in which the agents are organized, or in the manner in which coordination proceeds within the organization.

These works often omit a discussion about the order in which events that are displayed occur. A MAS is mainly a distributed system [37] and therefore requires specific techniques (e.g. logical clocks [29]) to know the order in which messages between agents occur. In previous works [51], Serrano and Botia proposed specific techniques to label events with logical clocks for subsequent management. The use of *order graphs* [51], which are based in discrete mathematics, was also introduced to display MAS events in order to debug these systems.

David Poutakidis et al. [43] criticize simple approaches of displays from the recorded data, like all those we have mentioned in this section. These researchers argue that in many cases, too much information is presented to developers, making it difficult to understand what really happens in the system. It is clear that when the system increases complexity and the data collections size is growing, automated methods are required. In the following paragraph, it is shown how when the number of agents in the system increases the number of messages exchanged in a MAS increases much more.

Let us consider a MAS with a set of n agents A. Let us also consider that each agent communicates with n_a agents in the system and finally that each agent sends m_a messages each time it communicates with another agent (including responses to received messages). Then, the number of elements of the set of messages M is the result of the following expression: $|M| = n \cdot n_a \cdot m_a$. If we consider as an example the very simple case in which agents send a single message to another agent for a consultation and they respond with a single message, then $n_a = (n - 1)$ and the number of messages sent by each agent is $m_a = 2$, values corresponding to a MAS with a very poor communication. In this case, $|M| = n \cdot (n - 1) \cdot 2 = 2n^2 - 2n$. This shows that the number of messages grows rapidly with the number of agents, even in systems with very low exchange of messages. In consequence, the principal line of investigation that has been followed in the testing and debugging of MAS is the automation of the test of agent's interactions. These kinds of approaches usually need three steps: (1) The first step is to define the protocols which specify the interaction between agents. (2) The second one is to test automatically that these protocols were correctly performed. The idea is that if the agents do not violate the specifications, the developer can guess that these agents are right. (3) Finally, the developer has to locate the errors found through some sort of display.

2.3. Protocols analysis

Petri-nets are one of the most popular methods to specify and debug communications between agents [12,36]. They consist of a generalization of automata theory so as to be able to express events occurring simultaneously. Poutakidis et al. [43] propose specifying protocols with AUML and translating them into Petri-net formalism. These networks will be used by the debugger to monitor conversations and provide error messages when protocols are not followed correctly. With this approach, the same authors classify typical mistakes that can be found in MASs [44]: *uninitialized agent, failure to send, wrong recipient, message sent multiple times, and wrong message sent*.

Besides the AUML diagrams and Petri-nets, other approaches are the use of extensions to the *propositional dynamic logic* [39], *statecharts* [19], or *dooley graphs* [38]. Each method has its pros and cons in the definition of protocols, testing and debugging [39]. We may summarize such pros and cons with the following sentence: the more accurate and complete a method to define a protocol is, the harder it is to understand the resulting definition and the reasons which originated failures.

There are works on the use of *causality graphs* [58,59] to facilitate the task of finding the cause of a failure in the execution of a protocol. In these graphs, each node corresponds to an agent state and each edge of the graph, or cause, is a message sent by the agent whose state is represented in the origin node to the agent whose state is represented in the destination node (the former agent causes the new state in the latter agent). Once again, it is necessary to use logical clocks [29] to determine the order in which events have happened in a distributed system which is a MAS in this case. We argue that these contributions are within the approaches of definition and verification of protocols because the definition of the protocols is required to generate causality graphs and to analyze where the problem occurred. In a broader framework than that of MASs, causality graphs have been a classic tool for debugging distributed systems since their proposal [60].

2.4. Societies analysis

The fundamental problem of analyzing protocols in MASs is that only what has been previously defined can be tested and debugged. Thus, a model of correct traces is required in order to know if an execution trace is correct. Such models correspond, in general, to protocol specifications. The individual agents within a MAS are autonomous and they can act in complicated and sophisticated ways. For example, (1) these agents may be adaptive in uncertain and unknown environments, (2) they may also organize themselves autonomously, and (3) they could also exhibit an "emergent" behavior [48]. Furthermore, the interactions between agents are complex and often unexpected [43]. A well-known feature of agent technologies is their ability to generate surprising, complex and emergent behaviors from very simple rules [1]. These behaviors usually imply interactions between the agents, which is unexpected by the developers of these systems. As a consequence of all these reasons, complex MAS behaviors can not be totally predefined and, in consequence, techniques analysed in Section 2.3 are not sufficient to debug complex MAS.

In order to do so, it might be of interest to register such unexpected behavior which can appear in a MAS. An example is the emergent behavior mentioned in the introduction. If these kinds of behaviors are to be detected, the analysis cannot be limited to a group of agents and its defined interactions. It is necessary to study the artificial society as a whole, i.e. the social level of the system. Besides, since the approach based on defining and testing protocols is not valid for our purposes (because it only deals with the predefined protocols), we must go back to the database with the events recorded from MAS executions and to the displays from these data. The problem with these two elements was the excessive amount of information which cannot be processed by humans. It is particularly advisable to obtain representations understandable by humans in the case of critical systems that should always be supervised, such as air traffic management. Therefore, special techniques are needed to discover knowledge about the MAS and also to translate this knowledge into representations understandable by humans. Our work focuses on this difficult task of displaying the social behavior of a MAS, with the aim of understanding, testing, and debugging it.

One of the most intuitive solutions to this problem is the use of data mining techniques [14]. Data mining is the non-trivial process of identifying valid, novel, potentially useful, and ultimately (but not always) understandable patterns in data [61]. It is an effective tool for reducing the complexity of the data, discovering knowledge in them, and even for getting simple representations. With these features, data mining has been widely applied to the visualization of complex data [25] and to the testing and debugging of software [30]. Serrano et al. have developed this idea in the context of testing and debugging MAS software [5,52] showing how data mining techniques can be used to obtain displays which clearly reflect groups of agents which behave in a similar manner as well as the most prominent agents within the society. Such concepts are useful because if, for example, among the agents under study there is a group of them which behave similarly, only one agent (i.e., a representative) should be considered for further debugging. In consequence, the results of such single agent debugging may be extrapolated to the rest of agents represented by it. Besides, the most prominent agents in the society should be carefully studied in the debugging process because faults in their design are critical for the whole society.

Social Network Analysis (SNA) has emerged as a very active research field in the last few years [50]. SNA depicts and measures the complex tissue of relationships between individuals, groups, organizations and other information entities. One of the more extended tools for SNA are PFNETs [13], which allow us to represent the most meaningful social interactions existing in a specific domain in an easy and human comprehensible way. They are generated by the Pathfinder algorithm, a network scaling algorithm used to prune many different kinds of networks, including citation networks, random networks, and social networks. The resulting network is graphically represented using a graph drawing algorithm, such as Kamada–Kawai [23]. PFNETs are used in a large variety of applications, including author co-citation analysis [45], latent knowledge visualization [8], scientific domain visualization [7,32], communication networks [53], animated visualization models of toxins [9], and mental models [26].

3. Review of previous MAS societies debugging strategies and the new proposal

This section seeks to give a global overview of the existing strategies to debug MAS societies and details the advantages of our new proposal, based on the use of PFNETs, with respect to the state of the art.

3.1. Existing strategies: collaboration graphs, collapsed collaboration graphs, and collapsed similarity graphs

The debugging approach chosen is the visualization of the MAS behavior. The effectiveness of the displays lies in a simple assumption: summarizing the data of a MAS execution in a display (without losing relevant information) allows us to identify undesired behaviors in a better way than by analyzing all the data generated. The visualizations must allow us to understand, and analyze and validate the behavior of the system at its social level in order to debug it (see Section 2). The following sections show that the proposal presented in this paper achieves these goals.

The first approaches proposed to put this task into effect were based on simple graphs. A *simple graph* is a pair $G = (V, E)$ comprising V, a nonempty set of vertices, and E, a set of unordered pairs of distinct elements of V called *edges* [47]. It is easy to get a sort of simple graph to be used to visualize the social level of a MAS. This graph is called *collaboration graph* and is defined as follows.

Definition 1. Let us consider a MAS software program P whose execution leads to a purely communicating MAS comprised by a set of agents A and a set of relations between agents R. Let us also consider a test t for P. The application of t to P allows us to obtain a set of observations denoted by M. We define a collaboration graph for the pair (P, M) as the graph $G(V, E)$ such that V is included in or equal to A and $(a_i, a_j) \in E$ if a_i sent some messages to a_j as indicated by M.

Fxample 1. Consider a very simple MAS in which there are 4 agents, with 3 of them being sellers (agents s_1, s_2, and s_3) and the fourth being a customer (agent c). In this MAS a particular test t has been performed. The collaboration graph for this simple system is the one shown in the left part of Fig. 1. The graph shows that the customer buys from the three sellers and the sellers do not communicate between themselves.

The main problem associated with using this simple representation of the social level, the collaboration graph, arises when the system becomes more complex and displays become as large and incomprehensible as the data they came from, as in the right part of Fig. 1 where more than 100 customers buying from several sellers are represented. Section 2.2 showed how the number of messages exchanged in a MAS increases quickly when agents are added to the system. The efficient visualization of large information domains is a very complex issue [32]. The reader will notice that MASs can be complex for many reasons, MASs are complex in nature. In the main, this is due to their autonomy and non-deterministic behavior. More-

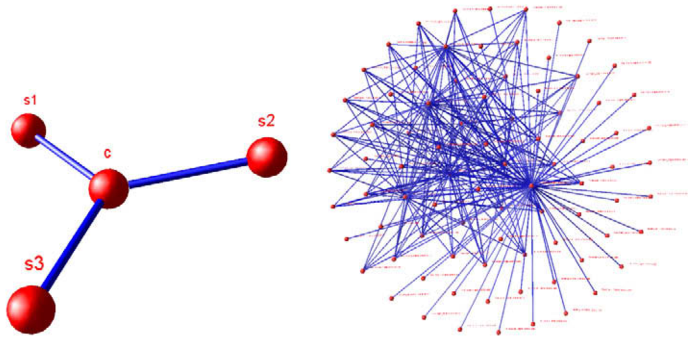

Fig. 1. Collaboration graph in a simple MAS (left). Collaboration graph in a complex MAS (right).

over, there is an extra complexity derived of their size. The representations must deal with general MASs. The common element to all MAS is the existence of agents and messages exchanged between them. The exponential growth in the number of messages with the number of agents is often the common factor which makes the representations useless.

In this framework, where we seek visualizations of a complex and unpredictable system with the aim of debugging it, data mining arises as a natural solution to simplify data and to discover patterns in them. Cluster analysis has become one of the most popular forms of unsupervised learning, where no *a priori* knowledge is needed. It has been used in a wide variety of applications, such as catalog segmentation, Web document categorization, e-mail overload management, and sales forecasting [20]. Some previous works which use clustering for MAS debugging at the social level can be found in the literature [17,52].

Actually, starting from the collaboration graph, as occurs in Fig. 1, a clustering process is equivalent to a transformation of the graph by collapsing nodes (i.e. agents) into more complex nodes (i.e. groups or clusters). How these nodes are collapsed and, in consequence, the meaning of the fact that two agents belong to the same group, define two types of graphs: collapsed similarity graphs and collapsed collaboration graphs.

Definition 2. Let us consider a MAS software program P whose execution leads to a purely communicating MAS comprised by a set of agents A and a set of relations between agents R. Let us also consider a software test t such that it is applied to the software and we obtain a set of observations from this test denoted by M. If we obtain a collaboration graph for (P, M) and denote it with $G(V, E)$, we may define a collapsed graph from G as a new graph $G' = (V', E')$ such that V' is a particular partition of V where each node in V' is a subset of the nodes/agents belonging to V and an edge $(v, v') \in E'$ if some agent in $v \in V'$ communicated with some agent in $v' \in V'$.

Now we can explain the collapsed similarity and collaboration graphs.

Definition 3. Let us consider a collaboration graph $G = (V, E)$. Suppose now we obtain $G' = (V', E')$ as a collapsed graph from G. We call this G' a collapsed similarity graph if the partition V' reflects an arrangement of agents such that agents belonging to the same element of the partition indicate that they share some degree of similarity.

In a nutshell, two agents are similar if these agents have the same kind of communications with the same agents. Serrano et al. [52] detailed the algorithm to obtain degrees of similarity between agents.

Definition 4. Let us consider a collaboration graph $G = (V, E)$. Suppose now we obtain $G' = (V', E')$ as a collapsed graph from G. We call this G' a collapsed collaboration graph if the partition V' reflects an arrangement of agents such that agents belonging to the same element of the partition indicate that they manifest a high interaction through cooperation.

In a nutshell, two agents have a high interaction (or they form a collaborative core) if the communication between them is larger (in terms of bytes or number of messages) than that developed by the other pairs of agents. Serrano et al. [52] detailed the algorithm to obtain degrees of interaction between agents.

In previous contributions [17,52], these methods are used in large-scale MASs. In these works, the implementation, application, and usefulness of data mining to obtain the aforementioned graphs are explained in depth.

3.2. Overview of our new proposal: MAS debugging by means of PFNETs

Collaboration graphs have some interesting topological properties which it is often valuable to display graphically, like the connection between agents which have interacted, the similarity between agents in a vertex of a *collapsed similarity graph*, or the high interaction between agents in a vertex of a *collapsed collaboration graph*. However, the raw graphs cannot often be visualized easily, especially when the size of the graphs grows proportionally with the number of data to be dealt with, and thus specific algorithms for simplifying such large graphs have been developed. Graph scaling algorithms, whose goal is to take proximity data and to obtain structures revealing the underlying organization of those data, use similarities, correlations or distances to prune a graph-based on the proximity between a pair of nodes. One of the best known methods to do this is the Pathfinder algorithm [13]. This is due to the various sound mathematical properties PFNETs (the pruned networks graphs obtained as output of the algorithm) present. These properties include [13]: (1) the conservation of the triangle inequalities (see Section 4.1) in any path of any number of links, (2) the capability of modeling asymmetrical relationships, (3) the representation of the most important global relationships present in the data, and (4) the fact that hierarchical constraints in most cluster analysis techniques do not apply to PFNETs.

In view of the latter, in this work we propose a methodology to debug MASs based on pruning their collaboration graphs by means of an advanced Pathfinder algorithm variant (see Section 5).

3.3. The PFNETs approach vs. the clustering approach for MAS debugging

The use of clustering to obtain graphs representing the society of a MAS often generates good summaries of the system. These graphs are used to debug the system, i.e., to verify that the system performance is adequate, at least in those aspects for which the graphs are meant. These aspects are similarity and collaboration between agents. However, the clustering-based approach has some shortcomings that are explained below. We show here how techniques based on PFNETs could help to overcome these shortcomings.

The currently identified shortcomings of the clustering approach are:

1. *The absence of intra and inter cluster visual information.* One of the weaknesses of clustering is that it does not provide information within a single cluster; its elements simply belong to that group. Hence, it does not show the relationship between the elements (the agents) who belong to the same cluster. One the one hand, the distance between two elements which appear within the same cluster does not indicate if these elements are more similar or more cooperative. On the other hand, it provides no information either on the relationship between elements of different clusters, or between different clusters.
2. *The absence of information about the most important elements in the representation.* Another defect is that clustering does not express where the most important agents[1] are and, thus, what would be the most important information to be displayed. However, this information could be easily expressed by putting a stronger emphasis on the central elements within the clusters, representing the important agents using nodes of larger size than those used by the rest, or emphasizing central clusters to indicate a greater importance in comparison to the remaining elements of the display.
3. *The amount of knowledge required by the user.* From our point of view, this is the fundamental reason which justifies changing the representation technique. The use of clustering requires some data mining skills to set the parameters of the algorithms correctly. For a newcomer, this fact is much more critical because, depending on the chosen parameters, the result will have a high variability in quality and in computation time. An expert should not experience major limitations in the way he understands the parameters of the data mining tasks or by performing a lengthy iterative process to achieve good results. In this domain, the achievement of visualizations which can be generated, manipulated and understood by inexperienced users, in an automatic and efficient way, is still a challenge.

Of course, some alternatives of basic clustering approaches already exist that deal with some of these limitations. One of these alternatives is *hierarchical clustering*. Dendrograms [14], a typical representation for this kind of clustering-based on trees, would directly address the first defined shortcoming (about the intra and inter cluster visualization). However, the new problem would be how to get a non-confusing representation of the hierarchy when a few hundred agents have to be processed. Another possible solution for the visualization problem is the use of *fuzzy clustering* [2]. With this technique, the relationships of each sample with all the clusters are clarified, but the relationships between the samples themselves are not. Besides, this model does not provide sufficient information to derive which samples are the most significant, thus the second defined shortcoming (about the most important elements emphasis) would still exist.

The use of PFNETs solves all the problems mentioned. By definition, PFNETs are connected, so each element is linked to the rest [45]. We propose to use this property to address the first problem (about the intra and inter cluster visualization). PFNETs are often combined with *graph drawing algorithms* such as Kamada–Kawai [23] that locate the most important elements[2] in the centre of the representation (also called the backbone) and the less relevant elements in the periphery [57]. Thus, thanks to their backbone behavior, PFNETs can also address the second problem (about the most important elements emphasis). Finally, PFNET generation algorithms are totally autonomous algorithms that require no parameterization from the user. This advantage efficiently addresses the third shortcoming mentioned (concerning the required user knowledge).

The advantages shown by PFNETs concerning the last three issues (the absence of intra and inter cluster visual information, the absence of information about the most important elements in the representation, and the amount of knowledge

[1] Note that, in our domain, when collaboration among agents is studied, an important agent is that with a significant activity in the collaboration graph (i.e., exchanging a significant amount of messages with the others) in comparison with the remaining agents in the MAS. When the similarities are studied, an important agent is that which is most similar to the rest of the agents (i.e. a representative).

[2] The importance of a node in the network is related to the weights of its edges in the weight matrix. This matrix collects the similarity values for each pair of nodes. The similarity measure considered depends on the application domain. For example, these values could be distances between each pair of nodes. For MAS debugging, when a similarity PFNET is built, these values will be related to the similarity within the set of communicating agents (see Definition 7 in Section 5). When a collaboration PFNET is built, they will be related to the number of messages exchanged between the two agents (see Section 5.2).

required by the user) have led us to propose their use to debug MASs at the level of society. We think that PFNETs obtained from collaboration graphs becoming expressive representations of the MAS social behavior are useful tools to debug these complex systems. In the next section, basic concepts about the generation of PFNETs are detailed, before explaining how these graphs allow the derivation of an understandable representation of a MAS in Section 5.

4. How to generate PFNETs

In this section, we give the formal definition of a PFNET and describe the algorithm we use to generate them.

PFNETs are generated by the Pathfinder algorithm introduced by Schvaneveldt [13]. Their objectives are to reveal the subjacent organization of a set of data, derived from a MAS in our case. Because of this explicative behavior, they are extensively used in SNA [32]. These graphs are based on the estimation or the measure of the distance between pairs of entities. Their formal definition is given as follows, taken from [13]:

Definition 5 (*PFNET*). A PFNET(r,q) is a septuple *(N, E, W, LLR, LMR, r, q)* where:

1. $N = \{a_1, \dots, a_n\}$ is the set of nodes of the network and $N \neq \emptyset$.
2. $E \subseteq \{(a_i, a_j) \in N \times N : i \neq j\}$ with cardinality m is the subset of edge names in the complete original graph. E is usually considered in a square matrix representation of dimensions $n \times n$ where e_{ij} is the name of the link connecting nodes a_i and a_j. Note that, by definition, there is a unique link between each pair of nodes and autolinks are not allowed.
3. W is the subset of weights associated to the m edges in E. Again, it usually takes a square matrix representation where $w_{ij} \in \mathbb{R}_0^+$. Thus w_{ij} denotes the weight of edge e_{ij}. The weights on the main diagonal are assumed to be zero and the remaining existing link weights are assumed to be finite and non negative.
4. *LLR*, the link-labeling rule, is the procedure used to determine a label for each link, according to some classification scheme.
5. *LMR*, the *link-membership rule*, is the procedure used to determine whether or not each element of the E matrix is added to the *PFNET(r,q)*.
6. r is the value of the r-metric considered to build the PFNET, and $1 \leqslant r \leqslant \infty$.
7. q is the value of the q parameter considered to build the PFNET, and $1 \leqslant q \leqslant n-1$, where n is the number of nodes.

The algorithm to get this kind of networks is called Pathfinder and will be introduced in the next subsection.

4.1. The Pathfinder algorithm

Pathfinder was introduced by Dearholt and Schvaneveldt [13] as a technique to yield structures revealing the underlying organization of proximity data in SNA. It is an alternative to other approaches for the same task such as clustering or multidimensional scaling. Pathfinder is based on graph theory and specifically on shortest paths in graphs. It defines estimates of the proximity between pairs of items as input and defines a network representation of the items that preserves only the most important links (a weighted pruned graph called PFNET). Hence, this PFNET is characterized by keeping only those links in the original graph whose weights do not violate the triangle inequality. When applied to graph theory, this classical mathematical property states that the direct distance between two nodes must be less than or equal to the total distance of any path joining them by passing through any group of intermediate nodes. More formally, $weight(i,j) \leqslant weight(i,k) + weight(k,j), \forall$ nodes i,j,k. As stated by its creators, PFNETs provide unique representations of the underlying structure for domains in which objective measures of distance are available [13].

The Pathfinder algorithm is applied on graphs where each link has a weight, and returns a pruned graph. The semantic of the network weights is problem-dependent and can be customized for the specific application domain being tackled. In this contribution, we propose the use of some similarity metrics allowing us to design PFNETS for MAS debugging at their social level. As will be introduced in Section 5, the weights in the links of our PFNETs are based on a metric considering the number of messages exchanged between the two linked agents when the collaboration is studied. When the similarity is studied, the weights are an index measuring how similar the agents' behavior is in terms of their cooperation with the remainder.

The Pathfinder algorithm is based on two main parameters. The first parameter is $r \in [1, \infty]$, which defines the adaptive metric, the *Minkowski r-metric*, considered to measure the distance between two graph nodes not directly connected:

$$D = \left\{ \sum_i d_i^r \right\}^{\frac{1}{r}}.$$

When r takes value 1, the Minkowski metric results in the sum of the link weights; when it takes value 2, it becomes the usual Euclidean metric; and when r tends to ∞, the path weight is the same as the maximum weight associated with any link along the path.

The second parameter is $q \in [2, n-1]$ (with n being the number of nodes in the graph), which limits the number of links in the paths for which the triangle inequality is ensured in the final PFNET. Hence, every path connecting two nodes that violate the triangle inequality, having an associated Minkowski distance greater than any other path between the same two nodes composed of up to q links, will be removed.

```
1. Compute $W^{i+1} = W \odot W^i$, as follows: $w_{jk}^{i+1} = \mathrm{MIN}((w_{jm})^r + (w_{mk}^i)^r)^{1/r}$,
   for $1 \le m \le n$.
2. Compute $D^i$, as follows: $d_{jk}^i = \mathrm{MIN}(w_{jk}^1, \ldots, w_{jk}^i)$, for $j \ne k$.
3. Iterate until $W^q$ and $D^q$ are computed.
4. Compare $W^1$ and $D^q$ : all the links having the same values in these two
   matrices will belong to the final PFNET.
```

Fig. 2. The Pathfinder algorithm.

The PFNET structure becomes sparser (has fewer links) as either r or q increases [13], whereas its interpretability increases. Of course it is always easier for a human being to interpret a graph with a significantly lower number of links representing only the most important global relations. This explains why PFNETs $(\infty, n-1)$ where both parameters are set to $q = n-1$ and $r = \infty$ have been used in a large variety of applications, as was shown at the end of related works review (Section 2).

To build a PFNET, two different kinds of auxiliary matrices are used. The first matrix is W_{jk}^i, which stores the minimum cost to go from node j to node k by following exactly i links. This matrix is computed recursively using matrix W_{jk}^{i-1}, with W^1 being the original weight matrix. The second matrix is D_{jk}^i, which stores the minimum cost to go from node j to node k by following any path in the graph composed of *i or less* links. This matrix is computed recursively using matrices $W_{jk}^1, \ldots, W_{jk}^i$.

The original Pathfinder algorithm pseudo-code is shown in Fig. 2 (w is the weight of an edge as in the PFNET definition, see Definition 5). Note that the algorithm has a time complexity order $O(q \cdot n^3)$ as q steps have to be performed to build the q matrices W^i and D^i. Each of the latter matrices stores n^2 weights, so a loop of this order is needed to compute them in each step. Finally, an additional loop of n steps is needed to compute each component of W^{i+1}, as seen in line 1 of the algorithm. As the maximum possible value for q is $n-1$, Pathfinder has a time complexity of $O(n^4)$ in that case. On the other hand, the resulting space is thus of complexity $2 \cdot q \cdot n^2$ ($2 \cdot n^3 - 2 \cdot n^2$ when $q = n-1$), since there is a need to build q matrices W^i and q other matrices D^i, as seen above.

4.2. Variants of the Pathfinder algorithm

The original Pathfinder algorithm can provide the user with all kinds of pruning, for all the possible acceptable parameter values for r and q, but this algorithm shows a high time and space complexity. A study conducted by Quirin et al. showed that the pruning of a graph of 1000 nodes could take more than one hour on a modern computer [45]. Since these kinds of graphs are frequent in MAS debugging, a better response time is a critical aspect in the elaboration of a better interaction with the user. Thus, three variants have been proposed to reduce its run time. Guerrero-Bote et al. [18] and Quirin et al. [46] recently proposed, respectively, the *Binary Pathfinder* algorithm and the *Fast Pathfinder* algorithm two improved variants of the original Pathfinder aimed at reducing both its time and its space complexity.

The third improved Pathfinder variant is called *MST-Pathfinder* introduced by Quirin et al. [45]. To make a comparison with the original Pathfinder, with this new algorithm, the same graph of 1000 nodes can be pruned in less than one second. This small run time has been obtained thanks to some restrictions defined on the main parameters, for which only the values $r = \infty$ and $q = n-1$ can be used. These values, and consequently, the small run time, make its use optimal for our application. As this will be the graph pruning algorithm considered in this contribution, we will describe its operation mode in detail.

It is well-known that there is a relationship between the results obtained with a *Minimum Spanning Tree* (MST)[3] algorithm and the Pathfinder algorithm. Dearholt [13] explicitly stated that, for a given symmetric cost matrix W, r and q, the union of all the MSTs extracted from a PFNET (r, q) is its PFNET $(\infty, n-1)$. As we have seen in the previous section, for each couple of nodes (i, j), PFNET $(\infty, n-1)$ is the set of links with the minimum cost among all the paths between the nodes i and j.

Given these considerations, a new algorithm based on the relation between MSTs and the Pathfinder algorithm parameterized with $r = \infty$ and $q = n-1$ (the typical values in the most applications) has been written and detailed [45]. It applies the idea that the generation of PFNET $(\infty, n-1)$ is the simple union of all the MSTs of a given graph. This algorithm, called *MST-Pathfinder*, gives the same result than Pathfinder (i.e., the same PFNET $(\infty, n-1)$), but with the same efficiency as the usual MST algorithms. The algorithm pseudo-code is shown in Fig. 3. The algorithm has several sub-functions: (1) *CREATE-CLUSTER(v)* creates a single cluster of size 1 including the node v as member, (2) *CLUSTER(v)* returns the cluster associated to node v, and (3) *MERGE-CLUSTER(u, v)* performs the union between the cluster containing node u and containing node v.

The algorithm needs $O(|E| \cdot \log(|E|))$ operations to sort the list of the links by their weights, where $|E|$ is the number of links. Using some data structures, such as path compression [11], it can be proved that the time complexity of this algorithm is $O(|E| \cdot \log(|E|))$. When having a dense network, $|E|$ is close to n^2 and $\log(n^2)$ is $O(\log(n))$, so the theoretical time complexity of the full algorithm can be simplified to $O(n^2 \cdot \log(n))$. In conclusion, this algorithm is much faster than the original Path-

[3] Let $G = (V, E)$ be a non-directed weighted and connected graph where V is the set of the nodes and E is the set of the links valued by their costs. A Minimum Spanning Tree of G is a sub-graph $T = (V, E')$ of $G, E' \subset E$, including all the nodes of G, where T is a tree and where the sum of the costs of each link is minimal.

```
1.  Define a tree, $T = \emptyset$
2.  Define $V[G]$, the set of the nodes of the graph $G$.
3.  Define $W$, the matrix of the costs for each link of $G$.
4.  FOR each node $v \in V[G]$
5.      CREATE-CLUSTER($v$).
6.  Create $F$, a set of all the links of $G$ sorted by their weights.
7.  FOR each link $e(u,v)$ remaining in $F$
8.      $H = \emptyset$.
9.      FOR each link $e(u',v')$ remaining in $F$ where $w(u,v) = w(u',v')$
10.         $F = F - \{e(u',v')\}$.
11.         IF CLUSTER($u'$) $\neq$ CLUSTER($v'$), THEN
12.             $T = T \cup \{e(u',v')\}$.
13.             $H = H \cup \{e(u',v')\}$.
14.     FOR each link $e(u',v') \in H$
15.         MERGE-CLUSTER($u',v'$).
16. Return $T$.
```

Fig. 3. The MST-Pathfinder algorithm.

finder ($O(n^4)$, when applied with $q = n - 1$). Concerning the memory, as only two link sets (T and F) have to be stored, the complexity is only $2 \cdot n^2$ (instead of $2 \cdot n^3 - 2 \cdot n^2$).

4.3. Visualization of PFNETs

As stated in Section 3.2, collaboration graphs have some interesting topological properties which it is often valuable to display graphically. The Pathfinder algorithm allows us to process the original raw graphs properly by reducing their complexity in order to obtain PFNETs that contain only the most interesting global relations existing between the agents. Hence, the PFNETS derived from the MAS social behavior will show some interesting topological properties that have a visual impact which will guide the expert in the MAS debugging. These properties include the spatial proximity between the nodes, the location of a specific node regarding the center of the map, and the degree of clustering of the ensemble of nodes. Nevertheless, they are only highlighted when the network is displayed graphically. It is in fact the combination of a pruning algorithm and a visualization algorithm which will reveal the organization of the network by emphasizing some of these properties.

Chaomei Chen was the first to promote the combination of the Pathfinder algorithm and Kamada–Kawai's algorithm [23] for the visualization of PFNETs applied to co-citation networks [6]. Kamada–Kawai's algorithm generates networks with criteria such as the maximum use of available space and the forced separation of nodes, so building balanced and aesthetic maps. Chen showed that the visual-spatial features obtained with this combination (for instance the organization of a central backbone and the rearrangement of the least important nodes on the periphery) is highly relevant for the interpretability of these co-citation networks. Hence, the Kamada–Kawai's algorithm, which displays the more relevant elements in the center of the PFNETs, allows us to solve one problem of our previous work mentioned in Section 3.3 which is the absence of information about the most important elements in the representation.

5. Debugging MASs with PFNETs

This section explains the way to obtain a PFNET of a purely communicating MAS. The networks obtained are equivalent to the two types of graphs which were explained in Section 3, collapsed similarity and collapsed collaboration graphs.[4]

Let us consider a MAS software program P whose execution includes a set of agents A and a set of relations between agents R. Let us also consider a test t for P. The application of t to P allows us to obtain a set of messages exchanged between agents denoted by M.

The set n of nodes $N = \{n_1, n_2, \dots, n_n\}$ in both PFNETs corresponds to the set of agents in the system $A = \{a_1, a_2, \dots, a_n\}$. That is, $N = A$. The Pathfinder variant algorithm used, *MST-Pathfinder*, applies its *LMR* (*link-membership rule*), setting the parameters $r = \infty$ and $q = n - 1$. In this work, the *LLR* (*link-labeling rule*) is not applied. Therefore, only the set of edges E and the set of weights W remain to end our definition of a PFNET sextuple. A PFNET will either be a similarity PFNET or a collaboration PFNET in function of these two sets, E and W. This section is devoted exclusively to obtaining a set of edges and weights to form the desired PFNETs.

[4] From now on, we will avoid using the term "collapsed" for the sake of simplicity.

5.1. Similarity PFNET

Obtaining the similarity PFNET requires the edges and weights to indicate similarity relations between the agents. That is, the weight function is going to define a measure which reflects the level of similarity between agents. We refer to similarity in terms of who the agents communicate with and with what intensity they carry out this communication. Before introducing the weight function definition, a series of previous definitions are shown to help to define and understand that function.

The basis to establish the collaboration or similarity between agents for this proposal is the number of messages exchanged.

Definition 6. A function $n_m(a_i, a_j)$ called number of messages is assumed, whose domain and codomain are $n_m : A^2 \rightarrow \mathbb{N}$. This function returns the number of messages in M which have been exchanged between any two agents $(a_i, a_j) \in A$.

The number of messages allows the set of agents which communicate with an agent or a couple of them to be established.

Definition 7. Let us define the set of communicating agents with an agent. This is the set $A_{a_i} \subseteq A$ including all the agents which communicate with an agent $a_i \in A$, i.e., $A_{a_i} = \{a \in A : n_m(a_i, a) \neq 0\}$. Let us also define the set of communicating agents with a couple of agents, composed of all the agents which communicate with a couple of agents $(a_i, a_j) \in A$. This set is defined as $A_{a_i,a_j} = (A_{a_i} \cup A_{a_j}) - \{a_i, a_j\}$.

The weight function defines a measure that reflects the agents similarity in terms of who the agents communicate with and with what intensity they carry out this communication. Let us consider two sets of agents: the *common agents* set (composed of the agents which both communicate with a_i and a_j) and the *different agents* set (composed of those agents which communicate with only one agent of the pair $(a_i, a_j) \in A$). In this way, for the similarity PFNET, the weight function should return a greater value when two agents communicate with the same agent. That is, the larger the set of common agents is, the higher the value the weight function returns. Moreover, the weight is greater when those communications with common agents have a similar number of messages. This would indicate that the couple of agents treated is more similar. It is also sought to reduce the weight returned with the size of the set of different agents. That is, the larger the set of different agents is, the lower the value the weight function returns. With the same reasoning, the weight is lower when those communications with different agents have a higher number of messages. This would indicate that the couple of agents treated is more different.

Definition 8. The weight function for the similarity PFNET, $W_s : A^2 \rightarrow (0, 1] \subset \mathbb{R}$, is defined as:

$$W_s(a_i, a_j) = \frac{1}{1 + \sum_{\forall a \in A_{a_i,a_j}} |n_m(a_i, a) - n_m(a_j, a)|}.$$

This function returns a weight of 1 if the agents passed as parameters communicate with the same agents and with the same intensity.

The set A_{a_i,a_j} contains the union between the *common agents* set and the *different agents* set. When the function considers an agent of the *different agents* set in the summation, one of the values of the difference is zero. Hence, the summation is increased and the value returned by the function is reduced. So, the more messages that have been sent to that different agent a, the larger the summation result will be. On the other hand, when the function considers an agent from the *common agents* set in the summation and agents (a_i, a_j) have exchanged the same number of messages, the summation is not increased with a. However, if the number of messages exchanged with a is different, the summation is increased on the basis of that difference. Thus, the value returned by the weight function is reduced. Therefore, the proposed weight function allows us to achieve the pursued objectives. For example, we can consider the case in which agents (a_i, a_j) only communicate with their agents in common and share the same number of messages. For this case, it is trivial to see that the final value returned by the function W_s is 1, indicating the highest possible similarity provided by the function.

The only element missing to complete the definition of a PFNET is the set of edges. The function W_s is bounded by $(0, 1]$. Note that weights asymptotically tend to 0. So if eliminating the least significant connections after the Pathfinder pruning is wished, thresholds for edge consideration have to be defined. For example, we could consider only edges with weights greater than 0.1, i.e., with a similarity greater than 10%. By the same reasoning, very high similarities can be eliminated using another threshold if doing so is considered to be interesting. A higher threshold μ_u and a lower threshold μ_l can thus be defined to limit the set of considered edges. The use of these thresholds is discussed in Section 5.3.

Definition 9. The set of edges for the similarity PFNET is $E = \{(a_i, a_j)\}$ with $a_i, a_j \in A$. If thresholds are used, the set of edges that would complete the similarity PFNET would be:

$$E = \{(a_i, a_j) : [W_s(a_i, a_j) \leqslant \mu_u \wedge W_s(a_i, a_j) \geqslant \mu_l]\}.$$

The reader can see how the similarity PFNET is very close to the classic use of PFNETs in the field of psychology to get the cognitive structure of the subject [49]. The difference is that, instead of concepts and their similarities, we have agents and their similarities. This paper extrapolates previous work with PFNETs to the field of debugging.

5.2. Collaboration PFNETs

In the same way as with the case of the similarities, for the collaboration PFNET, the definition of a weight function reflecting the degree of collaboration between agents and the set of edges is required. The weight function should return a value depending on how much two agents cooperate. The cooperation is understood as a exchange of messages (the more number of messages exchanged, the higher the cooperation).

Definition 10. The weight function for the collaboration PFNET sought with domain and codomain, $W_c : A^2 \rightarrow [0,1) \subset \mathbb{R}$, is:

$$W_c(a_i,a_j) = 1 - \frac{1}{1 + n_m(a_i,a_j)^2}.$$

The weight returned by the function is 0 if the agents do not cooperate at all and it tends to 1 as the communication between them becomes greater.

Note that the number of messages is squared to amplify this factor in the expression. Therefore, the function tends to 1 faster.

The only element missing to complete the sextuple of a collaboration PFNET is the set of edges.

Definition 11. The set of edges for the collaboration PFNET is $E = \{(a_i,a_j) : \ W_c(a_i,a_j) \neq 0\}$, with $a_i, a_j \in A$. If thresholds are used, the set of edges that would complete the collaboration PFNET would be:

$$E = \{(a_i,a_j) : [W_c(a_i,a_j) \neq 0 \wedge W_c(a_i,a_j) \leqslant \mu_u \wedge W_c(a_i,a_j) \geqslant \mu_i]\}.$$

Note that the function W_c (see Definition 10) is bounded between $[0,1)$. Therefore, it is trivial not to consider edges with a weight of 0, because a null weight indicates that there is no cooperation at all between the linked agents.

The reader can see how the collaboration PFNET is very similar to the implementation of the PFNETs to obtain scientograms of major scientific domains [32]. The difference is that instead of nodes as fields of science and edges as citations between the scientific disciplines, the nodes are agents and the edges are messages between agents. Again, this paper extrapolates previous work with PFNETs to the field of debugging.

5.3. Using thresholds

This section specifies a strategy to use and choose the weight thresholds that limit the set of edges. The thresholds are used exclusively to improve the visualization and understanding of the resulting PFNETs. This paper proposes the use of PFNETs to large-scale MASs, large systems which consequently cause huge representations. The thresholds can take values between 0 and 1 because they are used to limit the network edges (the weighted edges and the nodes that bind these edges), which also take values from 0 to 1.

Suppose we obtain a complex network, in terms of nodes and links. Suppose that its complexity is so high that the information displayed is overwhelming the user. This is a typical situation in which thresholds should be employed. The strategy we propose in one that has proved to be useful some time. It is based on a simple but effective routine. (1) First, set up the extreme values for $\mu_i = 0$ and $\mu_u = 1$ and obtain a new PFNET. Obtain conclusions about it. (2) Second, it is time now to study strong links. In order to do is, we have to progressively increase μ_i until we obtain a simple and easy to understand network. Again, we obtain conclusions about it. At this point, notice that deleting weakest links by augmenting the value of μ_i will often generate a set of different unconnected PFNETS. (2) Third, if we are interested now in studying weak links, we should progressively reduce the value of μ_i but also the value of μ_u, until we get a connected network again, with no strong links and the desired level of occurrence of weak links in order to obtain, again, conclusions about it. The weak links tend to link large groups of closely related elements. If the MAS test has been explored enough, finish the process. If not, simply go to step (2) or (3) depending on what is being sought. The reader may ask why an upper threshold is needed. The first strategy is increasing the lower threshold until an easy to understand network is obtained (step 2). Therefore, a difficult to understand network will be obtained if we reduce only the lower threshold in step 3. The solution is to reduce the upper threshold too.

There is a significant concern about the moment of pruning with thresholds to stop showing strong relations. First, an array of weights representing a network is obtained, second, this network is pruned using the Pathfinder algorithm that removes the less representative relationships (getting a PFNET); third the PFNET is represented graphically with a drawing algorithm, the Kamada–Kawai one for example. Reducing the upper threshold means deleting links needed by Pathfinder to remove irrelevant weak links. Therefore, the additional pruning through the use of thresholds must be made exactly between the application of Pathfinder algorithm, *MST-Pathfinder* for example, and the drawing algorithm. As a final comment, note that the networks obtained by setting thresholds often make us go back to previous networks, and especially to the global network, to supplement the provided information. In this process a user simply has to select the weights that are going to be shown. In any case, these representations no longer require data mining skills to set the parameters values.

5.4. Collaboration PFNETs for a concrete agent

Some agents can be highlighted during the strategy for the choice of thresholds (see previous subsection), for example the central elements of the PFNETs. The communications of these agents deserve special treatment because they can reveal bugs or undesired behaviors. To analyze the communications highlighted, the user can study the raw data obtained from the MAS. These data are the set of messages M which is obtained by applying a test t on a software program MAS P (see the beginning of Section 5). The basic forensic analysis (Section 2.1) can be used to perform this task. However, it is also possible to obtain more information using *PFNETs for a concrete agent*. This section studies the creation of those PFNETs.

The collaboration PFNETs for a concrete agent have all the elements in common with the collaboration PFNETs (see Section 5.2) except for the nodes set N, i.e. the agents studied. The only nodes which are interesting to study in these PFNETs are the concrete agent and the agents which communicate with it.

Definition 12. A collaboration PFNET for a concrete agent a_i, with $a_i \in A$, is a collaboration PFNET where $N = A_{a_i}$.

Note that the nodes are equivalent to the set of communicating agents with an agent a_i (Definition 7).

6. A case study

Now let us see the application of the proposed PFNETs to assist the process of debugging a MAS of high complexity at its social status. More specifically, PFNETs allow us to understand the behavior of the system to first identify deviations from expected behavior (i.e. errors) and then locate them and fix them (i.e. clean them). A complex MAS is used to appreciate the social level.

The complex system studied in this section models a society of users who want to obtain tickets to watch their preferred movies. This is *cinema example* which has been used and detailed previously [52] to illustrate how clustering works in the process of generating collaboration and similarity graphs. This system has been created with the INGENIAS methodology [40] and tools for development [51].

In the example description, user (i.e. customer) preferences are the preferred extras available (nachos, pop-corn, drinks), preferred sessions (e.g. from 19:00 to 20:00), preferred seats (e.g. seat numbers 1–20), movies (e.g. Die Hard), and the expected price [52]. The customer looks for sellers, the cinemas with the ticket price below or equal to his expectations. After each interaction the customer will evaluate the result and update a seller preferences model. This model will serve to eliminate those sellers who have not satisfied the customer requirements. The MAS software test used as an example of application includes 7 sellers and 100 customers.

The steps followed for the analysis are detailed now. (1) First the test of the MAS is launched in the infrastructure for forensic analysis [51] (see Section 2.1). At the least, the user must log the agents of the execution (which are the nodes of the following PFNETs) and the messages exchanged by each pair of agents (which are needed to establish the matrix of weights). (2) The matrix of weights is generated. The matrix is formed following Definition 8 or 10 (see Section 5) to generate a similarity or collaboration PFNET, respectively. (3) Edges are determined using Definitions 9 or 11 (see Section 5) to

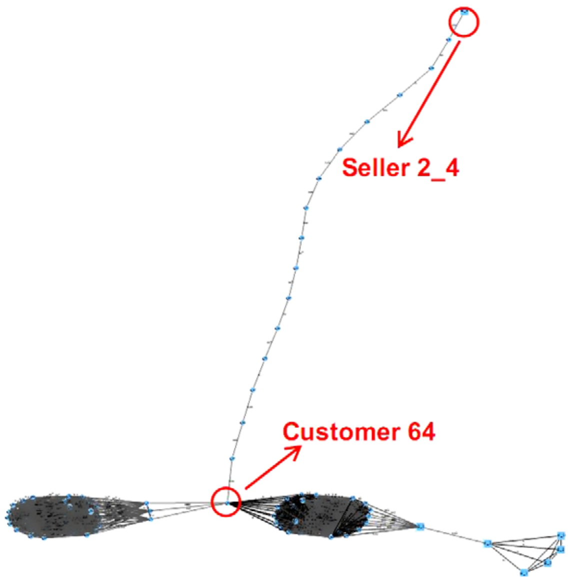

Fig. 4. Similarity PFNET.

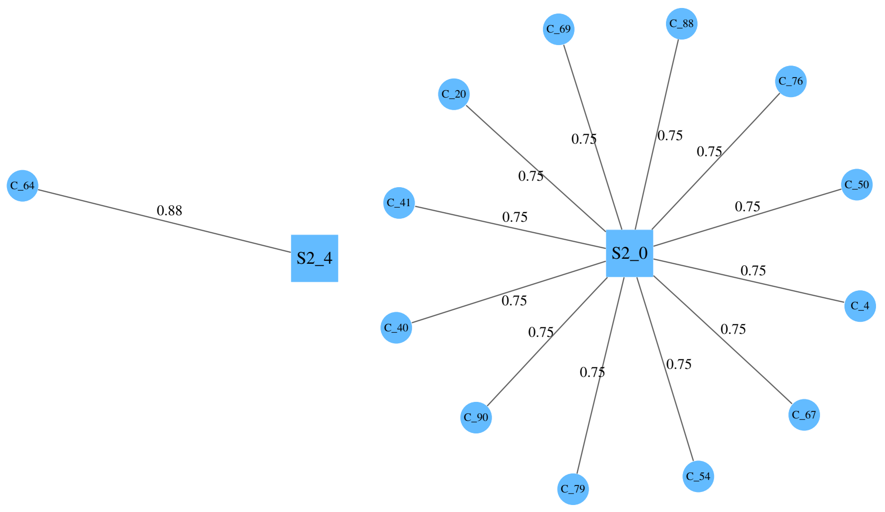

Fig. 5. Collaboration PFNETs for concrete agents: agent *Customer_64* (on the left), *Seller 2_0* (on the right).

generate a similarity or collaboration PFNET, respectively. (4) The MST-Pathfinder algorithm is executed (see Fig. 3) to prune the edges significantly. (5) The resulting PFNET is displayed with the Kamada–Kawai algorithm (see Section 4.3). (6) The user studies the network to extract interesting conclusions of the system. To complement the overall networks, the user can return to step 3 of the methodology in order to determine several thresholds. These thresholds exclude the edges to be considered in the displayed PFNET. The strategy of choice of thresholds is detailed in Section 5.3. Furthermore, data of the forensic analysis or PFNETs for concrete agents can be used to refine the findings from the overall networks (see Section 5.4).

6.1. Exploring the similarity dimension

First, similarity is treated. Let us focus on the overall similarity PFNET in Fig. 4.[5] Notice that W_s is defined to asymptotically tend to zero (see Definition 8). In other words, two agents are always going to have some relationship of similarity, maybe minimal, simply because they are both agents. To keep the graph connected, MST-pathfinder considers very weak relations, i.e. those whose weights are below 1% of the maximum weight. Therefore, after applying MST-Pathfinder, it is necessary to filter the edges which do not have interesting enough weights.

In the case of similarity, an overall net shows a customer at the center (*Customer_64*). That customer would be a prominent agent (i.e. a representative for the whole customers community in the society). It is also highlighted that *Seller 2_4* appears to be isolated from the rest. This means that it is an agent with minimal similarity with the remainder. Note that, this information about similarity can be completed if it is used with the adequate collaboration PFNET. The concrete PFNET for *Customer_64* appears on the left part of Fig. 5. It can be clearly seen that *Customer_64* only communicates with *Seller 2_4*. Again, using the concrete PFNET for *Seller 2_4*, it appears in Fig. 6, we can easily see that this seller has numerous and intense communications with many agents. Thus, we actually check that *Customer_64* is a representative of many other customers as he manifests the same behaviors with respect to them. Being the representative agent means having some degree of similarity with the majority of agents. The discovery of a representative has a great and obvious advantage: it prevents the developer from studying every single agent. Instead of that, the developer can devote his efforts to analyzing the representative only. Later, the results can be extrapolated to the rest of the agents.

Other interesting elements in this overall similarity PFNET (see Fig. 4) are the two subnets of similar customers located on the left and on the right of the representative, respectively. A subnet of sellers can also be appreciated at the rightmost side of the figure. This subnet shows six of the seven sellers, all except the mentioned *Seller 2_4*. At this point, we may question the importance of *Seller 2_4* because of its low similarity with the remaining sellers and the relationship of this agent with the representative. The rest of the study will reveal that importance.

According to the strategy used to choose the thresholds explained in Section 5.3, we move on to study the strongest relationships. The strongest connections can be appreciated if we set the weight interval to [0.2, 1], obtaining the three unconnected networks shown in Fig. 7. This figure shows a number of sellers with a similarity of a 100% at the bottom right part. That is, these sellers are communicating with the same agents and with the same number of messages sent to these agents. The overall PFNET (Fig. 4) showed these sellers at the right part of the figure. They were connected, with a lower degree of similarity, to two other sellers which do not appear in this figure (*Seller 2_0* and *Seller 1_2*). There are also two groups of sim-

[5] See the complete figure on: http://ants.dif.um.es/staff/emilioserra/ACLAnalyser/similarity0To100.pdf. The search tool in a pdf reader can be used to find specific agents.

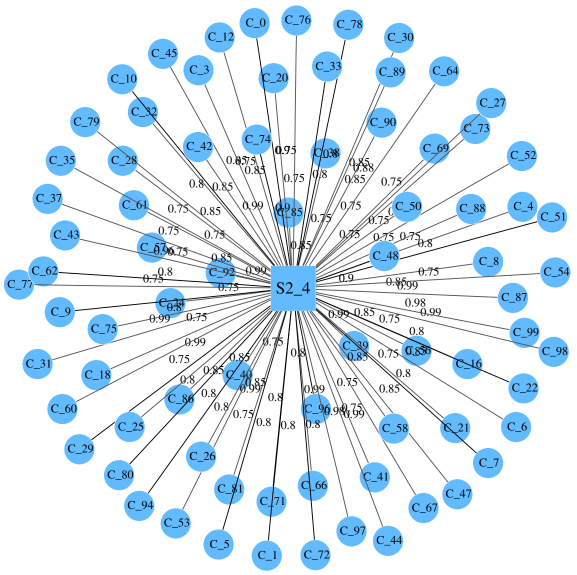

Fig. 6. Collaboration PFNETs for a concrete agent: *Seller 2_4*.

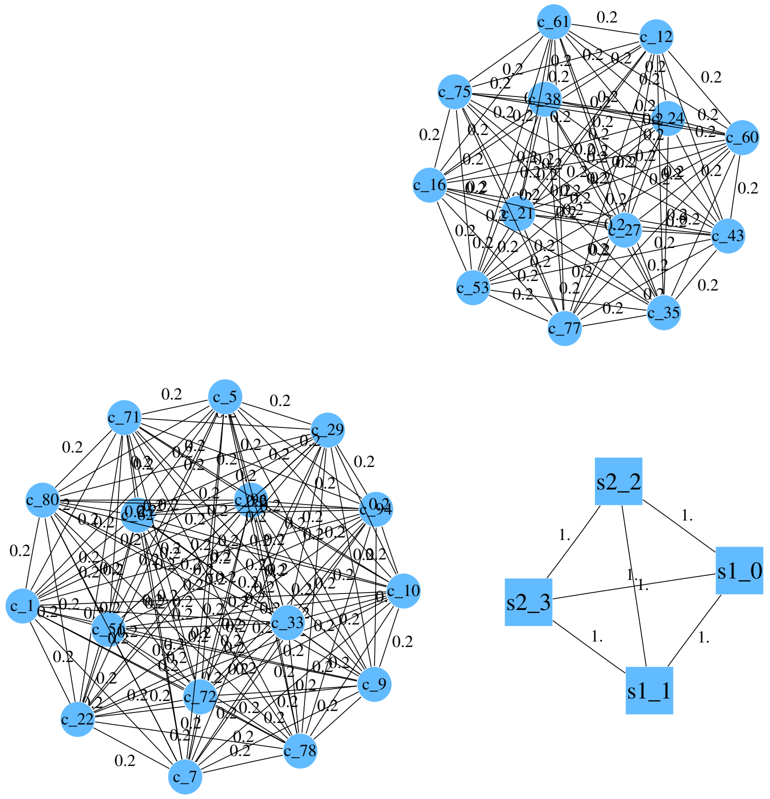

Fig. 7. Similarity PFNET only considering similarities in the range [0.2, 1].

ilar customers (with a similarity of 20%). The overall PFNET (Fig. 4) showed these two groups of customers in a subnet on the right of the representative. This subnet has been divided now into two subnetworks of customers with a high degree of similarity in this new PFNET. On reviewing the PFNETs for the concrete agents *Seller 2_0* (Fig. 5, right) and *Seller 1_2* (figure not included), it is observed that these sellers sell few times to few customers. Comparing the PFNET for the concrete agent *Seller*

2_0 with the PFNET for the concrete agent *Seller 2_4* (Fig. 6), we can see that the first seller has much fewer customers than second.

Once we have studied the whole PFNET and some of the interesting PFNETS for concrete agents, let us focus our attention on new PFNETS by using low thresholds. Thus, we will focus on the weakest relations of the whole PFNET. In consequence, edges with a weight of less than 0.1 are removed, $\mu_i = 0.1$ (that is, similarities less than 10%). The edges with a weight higher or equal to 0.2 are removed as well. The result is shown in Fig. 8. This figure shows seven different unconnected networks.

One of the networks, that in the upper right corner, consists of a small subset of sellers with a star topology. It is easily checked that this new graph is actually a sub-graph of that appearing in Fig. 4. Taking into account that link weights are limited to [0.1,0.2) and that we are working with similarity graphs, it can be concluded that there are at least three different groups of similar agents. One is composed of the tips of the start, another is only composed of the centre, and the final one is composed of the sellers which do not appear (i.e. *s2_4* and *s2_0*). They do not appear because they are very different from the rest of the seller agents in the system. One may hypothesize that they collaborate in extreme with the rest of agents (i.e. either very lightly or very intensively), but this can only be confirmed by using the collaboration PFNETS we will study in the next section.

Going back again to Fig. 8, note that all the customers belonging to the biggest network (i.e. that at the bottom left part of the figure) appear also in the PFNET in Fig. 7, arranged in two separate networks. Note that, they also appear in a connected subnet in the main PFNET at the right of the representative (Fig. 4). It is interesting to see a central axis of customers whose members connect with all the other customers at both sides of the network. The most immediate conclusion we may draw from such a representation is the following: all customers there are similar. However, if we work with this network and the two networks of customers in the Fig. 7, we discover that customers in the central axis of the network in Fig. 8 are the members of the network of customers at the top of Fig. 7. Moreover, the rest of agents in the same network are the members of the network of customers at the bottom of the same figure. Then, we see that by manipulating thresholds appropriately, it is possible to discover new similarity structures: a subnetwork of similar agents comprises those in the central axis and those at the left of the network, and another one composed of those in the central axis again and those at the right of the network. Hierarchical clustering cannot obtain such representations where the same elements belong to different clusters with no

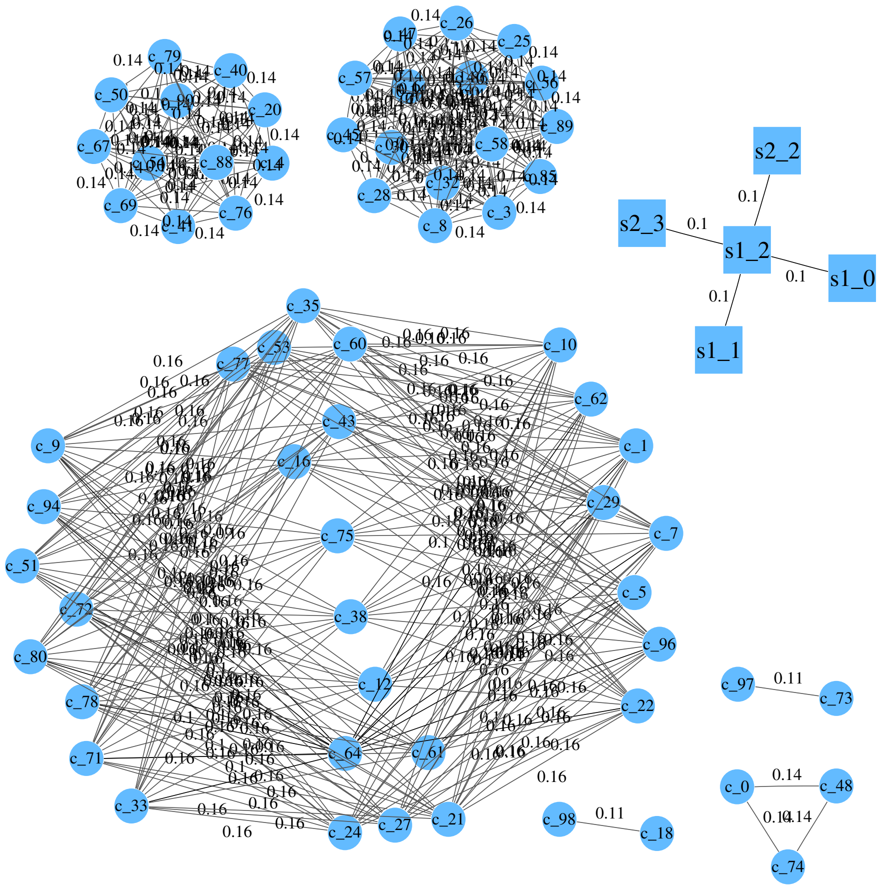

Fig. 8. Similarity PFNET only considering similarities in the range [0.1,0.2).

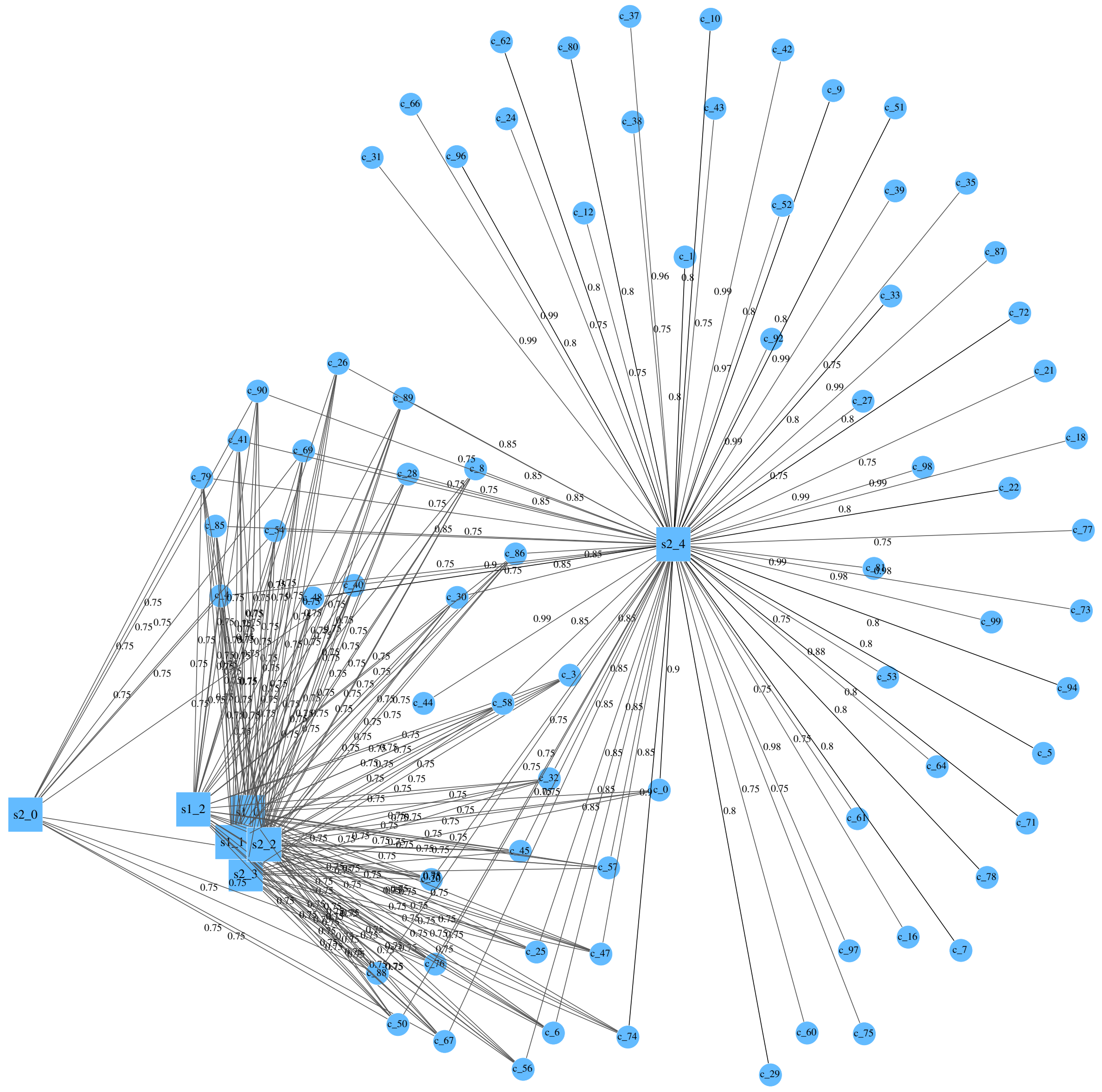

Fig. 9. Collaboration PFNET.

hierarchical relationship between them. Besides, we may obtain further information still. Note that, we may alternatively see such network as a set of similar agents whose representatives are the customers of the central axis. This axis includes the customers that are the most similar in comparison to all other customers in the graph. Therefore, in the absence of resources to study the behavior of each agent in this graph, it would be enough to select agents from the central axis for the study and to extrapolate the results to the rest of agents. For example, if the objectives of the agents in the central axis were accomplished, probably the objectives of the agents in the lateral axes would have been accomplished as well.

The remaining elements of Fig. 8 are subnets where customers are in the majority similar at a level of 14%. The other two subnets of customers of this PFNET (at the upper right part) appeared on the left of the overall PFNET. These customers have very low similarity between them. Therefore, they cannot be found in the previous PFNET, which studies the high similarities.

6.2. Exploring the collaboration dimension

Now let us see the collaboration network. In the case of collaboration, an overall PFNET is shown in Fig. 9.[6] The centerpiece is the agent *Seller 2_4* who accumulates collaborations with most of the customers. In the left part of Fig. 9, one can distinguish a

[6] See the complete figure on: http://ants.dif.um.es/staff/emilioserra/ACLAnalyser/co.pdf.

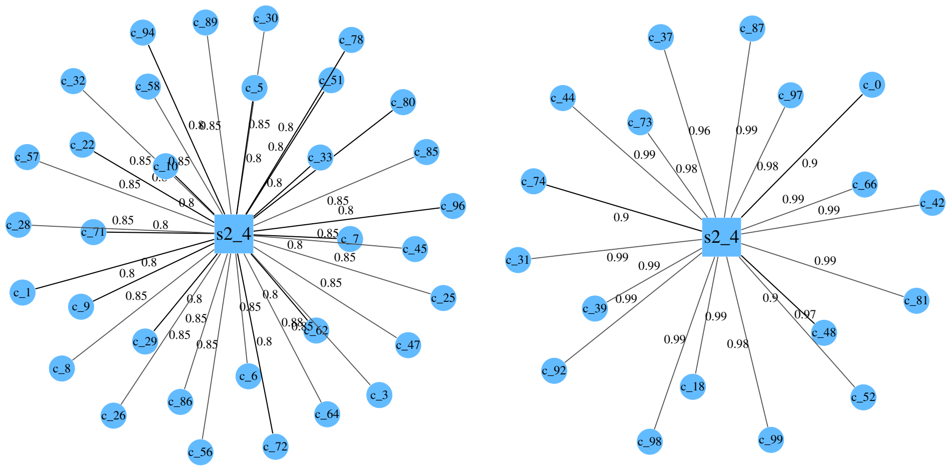

Fig. 10. Collaboration PFNET only considering collaborations in the range [0.8, 0.9) (on the left) and [0.9, 1] (on the right).

set of five sellers with a high collaboration with customers (*Sellers 1_0, 1_1, 1_2, 2_2 and 2_3*). Finally, we can see *Seller 2_0* at the left most part of the PFNET, wich collaborates poorly (weights are the lowest in the PFNET, 0.75) and only with few customers (there are few edges from this seller). On reviewing the PFNETs for these concrete agents (figure only included for *Seller 2_0*, see Fig. 5, right), it is observed that these sellers sell few times to few customers. Therefore, it is true that the central elements in the PFNET are the most important ones while the most irrelevant elements are in the peripheral part of the network, which is a feature of PFNETs combined with Kamada–Kawai's algorithm. In addition, the network helped us to discover anomalies at the social behavior of the system in an immediate and intuitive way. It is interesting to note that the observer can separate the PFNET mentioned into various subnets in a subjective way, somehow like clusters in hierarchical clustering. In this way, the observer can see a subnet around the most collaborative seller, *Seller 2_4*, composed of all the customers who buy from it. Perhaps the observer prefers to group *Seller 2_4* with the customers which only buy from this seller (at least with sufficient intensity to appear in the PFNET). These customers are easily distinguishable in the right part of the PFNET. The observer can also group together the five sellers on the left with the customers which buy from them. Let us denote this group as C_1. And he can also group *Seller 2_0*, with customers who buy from it. Let us denote this group of customers as C_2. We can clearly see that $C_2 \subset C_1$ in the PFNET, therefore it would be difficult to make this grouping using a hierarchical clustering technique.

Recall that in the similarity PFNET the agent *Seller 2_0* was considered to be very different from the other sellers. Now this network shows that this is because very few customers buy from this seller and they do it very infrequently. Quite the opposite of that seller is *Seller 2_4*, which is different from the others because it attracts a large amount of the market.

According to the strategy to choose thresholds explained in Section 5.3, we move on to study the strongest relationships. We can discuss the stronger collaborations by setting the weights interval to [0.9, 1], that is $\mu_i = 0.9$ and $\mu_u = 1$ (see the right part of Fig. 10). Now, the only seller which appears is the agent *Seller 2_4* with several customers around it. The relationship with the overall collaboration PFNET (Fig. 9) is quite clear. The central element is the same, *Seller 2_4*, and only its main collaborations are shown. Collaborations reach degrees of more than 99%. This figure recalls the heliocentric graph, used to obtain more detailed information about a specific node when PFNETs are used to get science maps [32] (see Section 2). In this case, the graph is a PFNET itself and its shape is a pure coincidence. The study of this PFNET gives clues for forensic analysis in order to discover bugs and undesired behaviors. Specifically, those customer agents have the capacity to spend much money (maybe too much) and for some reason they decided to buy almost always from *Seller 2_4*.

The weights interval [0.8, 0.9) can be used to discuss the collaborations with an average importance within the overall PFNET. This shows the collaborations between 80% and 90% (see the left part of Fig. 10). This range is chosen simply because the overall PFNET (Fig. 9) shows that there are many edges whose weights are within this interval. Besides, it allows us to study the average collaborations because there are also many weights higher (to study the highest collaborations) and lower (to study the lowest collaborations) than the limits of the interval chosen. As was expected, *Seller 2_4* appears again as the only seller with few customers which buy significantly from this seller. The relationship with the overall collaboration PFNET (Fig. 9) is again quite clear. The central element is the same, *Seller 2_4*, and only its average collaborations are shown. Using the collaboration PFNET, it is confirmed that the agent *Seller 2_4* plays a key role in the system and should be studied extensively by the forensic analysis (together with the customers which buy most intensively from this seller if it is possible).

If the collaborations of low intensity must be observed, a range of weights [0, 0.8) may be fixed (see Fig. 11).[7] Now the central elements are not sellers. Instead, we have a central axis of customers which accumulate most of the low intensity col-

[7] See the complete figure on: http://ants.dif.um.es/staff/emilioserra/ACLAnalyser/co2.pdf.

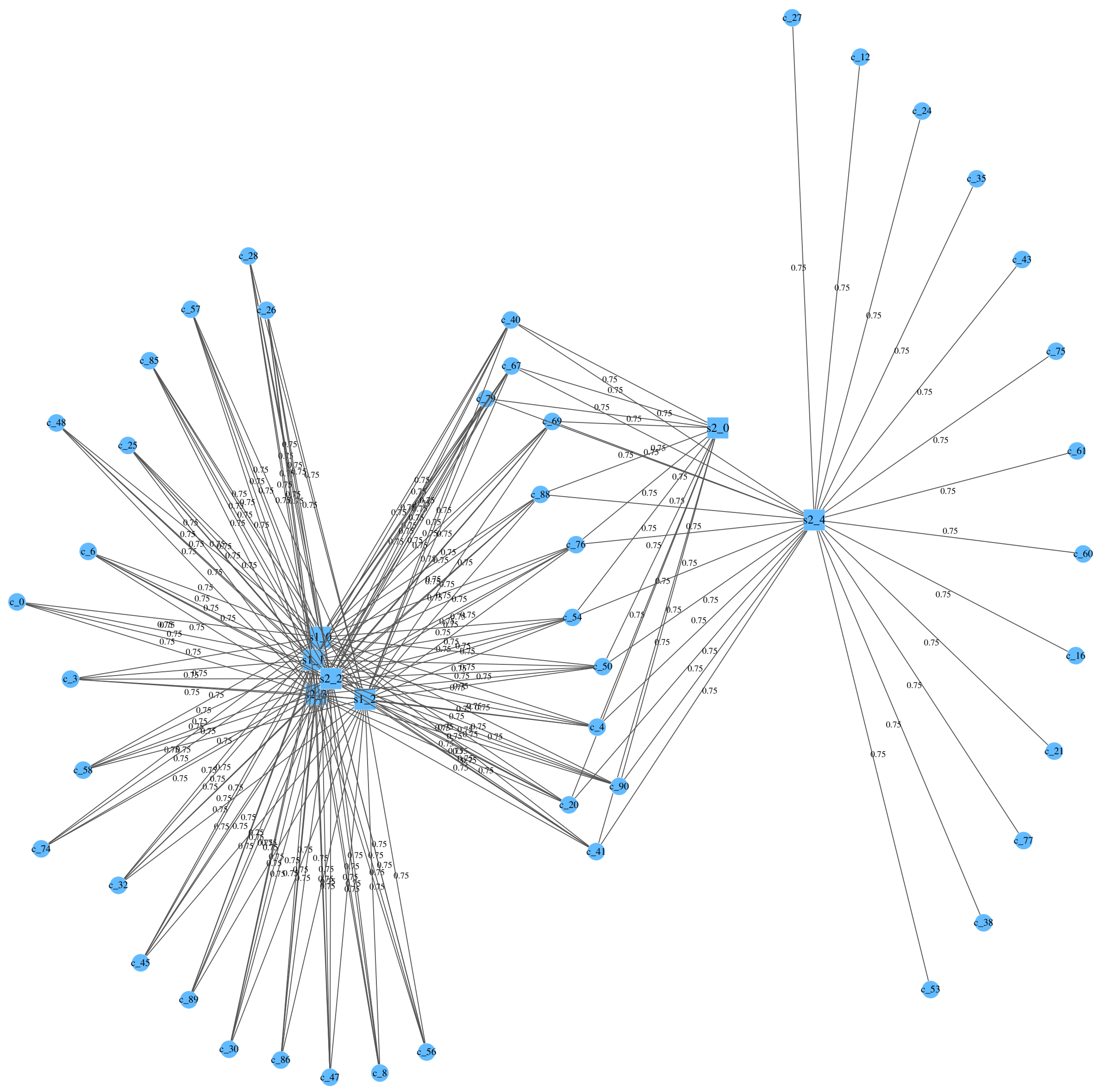

Fig. 11. Collaboration PFNET only considering collaborations in the range [0, 0.8).

laborations. Immediately on the right of this axis, the most centered seller (*Seller 2_0*) is located. This agent, which showed up as the least important seller in the overall PFNET (Fig. 9), is now the most important seller in the collaboration PFNET of low intensity.

Seller 2_4 still appears with few customers, who have a little collaboration with it. These customers do not appear in the two previous PFNETS (Fig. 10) because the ranges of thresholds selected are disjoint. On the other hand, some of the customers who collaborate with less popular sellers (customers at the left part of the PFNET) appear in previous PFNETs. This is because a customer may have a limited partnership with a seller (and appear in this PFNET which studies the low collaborations) and at the same time, this customer may have an intense relationship with other seller (and appear in PFNETs which study higher collaborations). The customer *Customer_0* shown in the left part of Fig. 10 is an example. This client is linked to *Seller 2_4* with a weight of 0.9. At the same time, the client is also shown in Fig. 11 linked with four other sellers with a weight of 0.75. Besides, the overall PFNET shows that customer and all his mentioned collaborations. Again, the study of this PFNET gives clues for forensic analysis in order to discover bugs and undesired behaviors. Specifically, *Seller 2_0* should be studied to determine why it has so few customers. It should also be studied why these customers still buy from this seller. The PFNET of the concrete agent *Seller 2_0* (Fig. 5, right) showed clearly who those customers are.

6.3. Discussion

In the light of the previous study, we have obtained a lot of useful points which denote possible clues for a successful forensic analysis.

1. There is a customer similar, in some way, to all the agents in the system, and it is *Customer_64* (see Fig. 4). The only agent this customer communicates with is *Seller 2_4* (see Fig. 5, left).
2. *Seller 2_4* is very different from the remaining agents (Fig. 4).
3. The remaining sellers have some similarity between them (although it can be very small). These sellers are well separated and distinguished of the customers (Fig. 4).
4. There are sellers with a degree of similarity of 100%: *Sellers 1_0, 1_1, 2_2 and 2_3* (Fig. 7).
5. The maximum similarity between customers is 20% (see Fig. 7).
6. *Seller 1_2* has a 10% similarity with the set of sellers mentioned in (4) (see Fig. 8).
7. *Seller 2_4* is the seller with the most customers (Fig. 9).
8. Other sellers share customers with *Seller 2_4* (Fig. 9).
9. *Seller 2_4* is the only seller used intensively by the customers (Fig. 10).
10. *Seller 2_0* is the seller with the least sales (Fig. 11).
11. There are customers who buy from the most popular seller, but also from other sellers (although with less intensity).

The study of PFNETs reveals interesting features in the execution of the MAS, which can guide the forensic analysis of the obtained data. As stated, the existence of a representative (item 1) allows choosing the agent to be analyzed when there are limited resources. It also highlights *Seller 2_4*. This seller is the most different agent in the system (item 2). These two agents should be studied extensively. Other sellers are more or less similar, so there is no preferred one to be revised (item 3). However, the high similarity of four of the sellers (item 4) makes one suspect an undesired behavior. A degree of similarity of 100% means that those sellers have sent messages to the same customers and exactly the same number of messages. It is also noted that customers are very different, reaching a maximum similarity of 20% (item 5). This last point contrasts with the existence of some sellers being so similar. Therefore, it can be deduced that customers buy at very different frequencies from those sellers which are not so similar (making the customers more different). Item (6) indicates that *Seller 1_2* is slightly similar to the aforementioned four equal sellers. Item (7) says that *Seller 2_4* is the most successful seller. This explains why it is the most different agent of the system (item 2). Moreover, this is the only seller which receives collaborations with high intensity (item 9). Therefore, it is confirmed that this seller is a key agent in the system and should be studied intensively. In particular, analyzing the set of messages *M* obtained from the execution it is noted that the communications with this seller are 75.4% of all messages exchanged in the system. Another interesting point to study is why the other sellers do not have loyal customers to collaborate with. The customers who buy from the most popular seller sometimes buy from the remaining sellers (item 11). Therefore, it is clear that customers have access to all sellers. More specifically, the forensic analysis should study why *Seller 2_0* is the seller with the least number of customers (item 10). If this seller is clearly inferior to the others, perhaps it should be able to associate with others by establishing communications between sellers.

At this point of the paper, it is possible that some readers may still wonder about the actual convenience of using networks in which the only links appearing are those with low weights. To illustrate their utility more clearly, let us move to another application domain for a while. Instead of speaking about sellers and customers, let us focus our attention on the Air Traffic Management [42] arena. Suppose that instead of using seller agents, we model airstrips as active elements of a particular airport. In the sellers and customers domain, a weak link within a collaboration PFNET reflected that specific sellers received a low number of demands from a customer. But maybe in the air traffic management scenario it is even clearer. In this case, a specific airstrip with weak links would be underused. No one would doubt about the convenience of a deep study of the reasons why such a resource is not sufficiently employed. Thus, studying weak links is totally justified. Moreover, by simply changing defined weight functions as defined in this paper with their corresponding inverse, new PFNETs would be generated in which the most different (if we refer to similarity networks) or the least collaborative (if we refer to collaboration networks) appear in the centre of the representation.

It is interesting to note that the centerpiece in the global collaboration network is the agent *Seller 2_4*, which accumulates collaborations with most of the agents. Remember that this agent was considered the least similar to the others in the similarity network. This reinforces the idea that the similarity and the collaboration between agents are complementary concepts. An agent interacts with another agent because the former wants to get a service. Therefore, the former agent does not provide the wanted service and is different from the latter. Perhaps in a social network environment, rather than in a customer-server one, the opposite happens and the collaboration indicates a similarity between agents. Therefore, the collaboration can be understood in different ways depending on the problem domain. In view of the above, one can see that using the social science and social-based-simulation techniques sometimes allows one of the main goals of this science disciplines to be reached: the empirical verification of sociological hypotheses. In this case, the hypothesis is that the similarity between agents and the collaboration between them are contrary concepts.

With PFNETs we have interesting information for the compression and debugging of the social level of a MAS like: very similar customers between them, identical sellers, sellers different from the remaining ones because they do not attract customers, sellers different from the remaining ones because they attract the most customers, customers which only buy from one seller with intensity, customers buying from several sellers, etc. The representations have split a set of a hundred customers in to a handful of similar groups of customers. This simple fact means that PFNETs are a good way to divide the complex study about the social level of this MAS. That is, by using PFNETs, hierarchies have been found in the system. By hierarchy we mean a system that is composed of interrelated subsystems in the same way that atoms compose molecules, nations compose continents or galaxies compose the universe. Complexity frequently takes the form of hierarchy and those hierarchical systems have some common properties that are independent of their specific content [54]. Hence, it is possible to develop techniques which capture the social hierarchy of a MAS as shown in this paper. This section shows that PFNETs offer excellent views about collaborations or similarities between agents to understand and debug a MAS. Moreover, this technique is automatic and requires no obscure user parameters to configure its behavior (except for the intuitive weights interval to be shown).

7. Conclusions and future work

This paper deals with assistance to the process of debugging a multi-agent system (MAS) of high complexity in its social level. More specifically, tools for understanding the behavior of the system have been given. They allow us to detect deviations from the expected behavior (i.e. possible errors) and then to locate and fix them (i.e. debugging). These tools consist of Pathfinder networks, PFNETs, which allow us to understand the social behavior of the system, discover emergent behaviors and debug possible undesirable behaviors.

The *Introduction* shows the inherit complexity in MAS testing and debugging. The *Related works* section explains the problem of studying the social level of MASs and how techniques to discover knowledge are necessary, like the clustering or PFNETs. Previous approaches to testing and debugging MASs are, with this, invalid for the social level. The *Review of previous MAS societies debugging strategies and the new proposal* section shows a criticism of previous attempts to capture and visualize the social behavior of the MAS in order to understand and debug it using data mining. In particular, this section concluded that the complicated set of parameters that data mining often requires is an impediment for inexperienced users. We have presented PFNETs as a solution to this and other problems mentioned. The section *How to generate PFNETs* explains the fundamental concepts of PFNETs regardless of their field of application. Moreover, the classic Pathfinder algorithm to generate PFNETs, with a time complexity of $O(n^4)$, is detailed and the use of MST-Pathfinder is proposed as an alternative of high efficiency with a time complexity of $O(n^2 \cdot log(n))$. The section *Debugging MASs with PFNETs* explains how representations about the social behavior of MASs can be obtained with only a few extra terms to the classical definition of PFNETs. In particular, the performance of representations about similarities between agents and collaboration cores in the agent society is described. Finally, the section *A case study* applies the above representations to the understanding and treatment of a specific *large-scale MAS*. Throughout the section it is shown how PFNETs reveal interesting information about the behavior of the system, information about possible malfunctions of society, emergent behaviors, and even empirical testing of sociological hypotheses. PFNETs, which are used successfully in different fields, are presented in this paper as an effective tool in the process of understanding, testing and debugging a complex MAS in its social level.

Future work includes: studying new aspects in agent societies to be represented and improving the representations, studying alternative techniques to get representations, improving the degree of automation.

The first future work is the development of new representations of the MAS execution to debug these systems. In the current contribution, we have only described the similarity and collaboration PFNETs. Of course, there are many interesting elements to be represented in a system. For example, according to the methodology of development and programming platform, elements which can appear are: the class implemented by an agent, the role played, the state of its beliefs, intentions and desires, etc. In this way, for example, the shape of the node could indicate that the major role played by the agent is petitioner of information and services, reporter/server, etc. Another example is the use of the range of colors to indicate, from light to dark, the degree in which agents have achieved their objectives in the execution.

The basic representation of PFNETs which has been shown in this paper can be improved to show more information and to be a more effective debugging tool. For example, the thickness of the nodes and edges can have a useful meaning, as regards the collaboration degree. In this way, if an edge between two nodes is very thick, it might indicate that the agents have been collaborating extensively. Or it may indicate an accumulated collaboration of that agent with the rest of the nodes which are connected to it. We could make an analogous reasoning about similarity.

It is also interesting to automate the process of selecting a representative range of weights. For example, fuzzy partitions can be used to split the weights in to: low, middle, and high. Besides, the use of fuzzy technology allows some weights to be treated as low and middle at the same time (for example) in order to make partitions more homogeneous and representative.

As we improve the techniques described here, we should not despise other methods for the visualization of systems that facilitate understanding, testing, and debugging. We have already worked in multivariate statistical analysis with clustering and in the field of social networks with Pathfinder. Another major approach to be considered is neural networks. For example, self organizing maps could be used as an alternative to clustering. In general, each approach will have certain advantages that the others do not show. Therefore, combinations or iterative approaches can be interesting to understanding the social level of the MAS. For example, clustering could be used to monitor large groups of similar agents in a bird's eye view, and then PFNETs could be used to detail each of the previously obtained groups.

The current paper shows that when the level of abstraction to study a MAS is increased from the group level to the social level, we come back to the first approaches for debugging: providing the developer with displays which show several kinds of information (see Section 2). Therefore, our most challenging future work is to try to reach the degree of automation, effectiveness and efficiency that is achieved in the testing and debugging of the MAS protocols, but applied to our problem: the study of agent societies where previously undefined behaviors can appear.

Acknowledgments

This research work is supported by the Spanish Ministry of Education and Science in the scope of the Research Projects TSI-020302-2009-43, TIN2008-06441-C02-02, TIN2009-07727, including EDRF fundings and through the Fundación Séneca within the Program "Generación del Conocimiento Científico de Excelencia" (04552/GERM/06), Facultad de Informática, Campus Universitario de Espinardo, 30100 Murcia, Spain.

References

[1] A.L. Bauer, C.A.A. Beauchemin, A.S. Perelson, Agent-based modeling of host-pathogen systems: the successes and challenges, Information Sciences 179 (10) (2009) 1379–1389. URL <http://dx.doi.org/10.1016/j.ins.2008.11.012>.
[2] J. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, New York, 1981.
[3] R.H. Bordini, A.C. d. R. Costa, J.F. Hübner, F.Y. Okuyama, l.F. Moreira, R. Vieira, {MAS-SOC}: a social simulation platform based on agent-oriented programming, Journal of Artificial Societies and Social Simulation 8 (3) (2005) 7. URL <http://jasss.soc.surrey.ac.uk/8/3/7.html>.
[4] R.H. Bordini, M. Dastani, J. Dix, A.E. Fallah-Seghrouchni, (Eds.), Programming Multi-Agent Systems, Fourth International Workshop, ProMAS 2006, Hakodate, Japan, May 9, 2006, Revised and Invited Papers, Lecture Notes in Computer Science, vol. 4411, Springer, 2007.
[5] J.A. Botia, J.M. Hernansaez, A.F. Gómez-Skarmeta, On the application of clustering techniques to support debugging large-scale multi-agent systems, in: Programming Multi-Agent Systems, Fourth International Workshop, ProMAS 2006, Hakodate, Japan, May 9, 2006, Revised and Invited Papers, Lecture Notes in Computer Science, vol. 4411, Springer, 2006, pp. 217–227. URL <http://dx.doi.org/10.1007/978-3-540-71956-4_13>.
[6] C. Chen, Information Visualization and Virtual Environments, Springer, Berlin, Germany; New York, 1999.
[7] C. Chen, Information Visualization: Beyond the Horizon, Springer, Berlin, Germany, 2004.
[8] C. Chen, J. Kuljis, R. Paul, Visualizing latent domain knowledge, IEEE Transactions on Systems, Man, and Cybernetics, Part C 31 (4) (2001) 518–529.
[9] C. Chen, S. Morris, Visualizing evolving networks: Minimum spanning trees versus pathfinder networks, in: Information Visualization, IEEE Symposium on 09.2003. URL <http://doi.ieeecomputersociety.org/10.1109/INFVIS.2003.%1249010>.
[10] R. Coelho, U. Kulesza, A. von Staa, C. Lucena, Unit testing in multi-agent systems using mock agents and aspects, in: SELMAS '06: Proceedings of the 2006 International Workshop on Software Engineering for Large-scale Multi-agent Systems, ACM, New York, NY, USA, 2006, pp. 83–90. URL <http://doi.acm.org/10.1145/1138063.1138079>.
[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, 2001.
[12] R.S. Cost, Y. Chen, T.W. Finin, Y. Labrou, Y. Peng, Using colored petri nets for conversation modeling, in: Issues in Agent Communication. Springer-Verlag, London, UK, 2000, pp. 178–192.
[13] D. Dearholt, R. Schvaneveldt, Properties of pathfinder networks, in: R. Schvaneveldt (Ed.), Pathfinder Associative Networks: Studies in Knowledge Organization, Ablex Publishing Corporation, 1990, pp. 1–30.
[14] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, second ed., Wiley-Interscience, 2000. November URL <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike%09-20&amp;path=ASIN/0471056693>.
[15] M.B. Dwyer, L.A. Clarke, Data flow analysis for verifying properties of concurrent programs, in: SIGSOFT: Proceedings of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering, ACM, New York, NY, USA, 1994, pp. 62–75. URL <http://doi.acm.org/10.1145/193173.195295>.
[16] J. Ferber, Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley Professional, February 1999. URL <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike%09-20&amp;path=ASIN/0201360489>.
[17] J.J. Gómez-Sanz, J. Botia, E. Serrano, J. Pavón, Testing and debugging of MAS interactions with INGENIAS, in: Agent-Oriented Software Engineering IX: 9th International Workshop, AOSE 2008 Estoril, Portugal, May 12–13, 2008 Revised Selected Papers, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 199–212. URL <http://dx.doi.org/10.1007/978-3-642-01338-6_15>.
[18] V. Guerrero-Bote, F. Zapico-Alonso, M. Espinosa-Calvo, R. Gómez-Crisóstomo, F. Moya-Anegón, Binary pathfinder: an improvement to the pathfinder algorithm, Information Processing and Management 42 (2006) 1484–1490.
[19] D. Harel, M. Politi, Modeling Reactive Systems with Statecharts: The Statemate Approach, McGraw-Hill, Inc., New York, NY, USA, 1998.
[20] H.-L. Hu, Y.-L. Chen, Mining typical patterns from databases, Information Sciences 178 (19) (2008) 3683–3696. URL <http://dx.doi.org/10.1016/j.ins.2008.05.036>.
[21] D.M. Huizinga, D. Phung, T. Ryu, Automated defect prevention with visual studio team system: a case study for software engineering, in: H.R. Arabnia, H. Reza (Eds.), Software Engineering Research and Practice, CSREA Press, 2008, pp. 32–38.
[22] S. Johnson, Emergence: The Connected Lives of Ants, Brains, Cities, and Software, Scribner, September 2002. URL <http://www.amazon.com/exec/obidos/redirect?tag=citeulik%e07-20&path=ASIN/0684868768>.
[23] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Information Processing Letters 31 (1) (1989) 7–15.
[24] D. Keil, D.Q. Goldin, Indirect interaction in environments for multi-agent systems E4MAS, in: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Lecture Notes in Computer Science, vol. 3830, Springer, 2005, pp. 68–87.
[25] D.A. Keim, Information visualization and visual data mining, IEEE Transactions on Visualization and Computer Graphics 08 (1) (2002) 1–8. URL <http://doi.ieeecomputersociety.org/10.1109/2945.981847>..
[26] U. Kudikyala, R. Vaughn, Software requirement understanding using pathfinder networks: discovering and evaluating mental models, Journal of Systems and Software 74 (1) (2005) 101–108.
[27] S. Kumar, E. Spafford, An application of pattern matching in intrusion detection, Technical Report, Purdue University, Department of Computer Sciences, 1994.
[28] D.N. Lam, K.S. Barber, Comprehending agent software, in: AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, New York, NY, USA, 2005, pp. 586–593.
[29] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM 21 (7) (1978) 558–565. URL <http://doi.acm.org/10.1145/359545.359563>.
[30] M. Last, M. Friedman, A. Kandel, The data mining approach to automated software testing, in: KDD '03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2003, pp. 388–396. URL <http://doi.acm.org/10.1145/956750.956795>.
[31] R. Miles, AspectJ Cookbook, 2004. ISBN 10: 0-596-00654-3.
[32] F. Moya-Anegón, B. Vargas-Quesada, Z. Chinchilla-Rodríguez, E. Corera-Álvarez, A. González-Molina, F. Muñoz-Fernández, V. Herrero-Solana, Visualizing the marrow of science, Journal of the American Society for Information Science and Technology 58 (14) (2007) 2167–2179.

[33] G.J. Myers, C. Sandler, T. Badgett, T.M. Thomas, The Art of Software Testing, second ed., Wiley, 2004. URL <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike%09-20&amp;path=ASIN/0471469122>.
[34] D.T. Ndumu, H.S. Nwana, L.C. Lee, J.C. Collis, Visualising and debugging distributed multi-agent systems, in: AGENTS '99: Proceedings of the Third Annual Conference on Autonomous Agents, 1999a, pp. 326–333.
[35] D.T. Ndumu, H.S. Nwana, L.C. Lee, J.C. Collis, Visualising and debugging distributed multi-agent systems, in: AGENTS'99: Proceedings of the Third Annual Conference on Autonomous Agents, ACM, New York, NY, USA, 1999b, pp. 326–333. URL <http://doi.acm.org/10.1145/301136.301220>.
[36] M. Nowostawski, M. Purvis, S. Cranefield, Modelling and visualizing agent conversations, in: AGENTS '01: Proceedings of the Fifth International Conference on Autonomous Agents, ACM, New York, NY, USA, 2001, pp. 234–235. URL <http://doi.acm.org/10.1145/375735.376297>.
[37] G.M.P. O'Hare, M.J. Wooldridge, A software engineering perspective on multi-agent system design: experience in the development of MADE, in: Distributed Artificial Intelligence: Theory and Praxis, Kluwer Academic Publishers, Norwell, MA, USA, 1992, pp. 109–127. ISBN 0-7923-1585-5.
[38] H.V.D. Parunak, Visualizing agent conversations: using enhanced dooley graphs for agent design and analysis, in: Proceedings of the Second International Conference on Multiagent Systems, AAAI Press, Menlo Park, California, 1996, pp. 275–282.
[39] S. Paurobally, Developing agent interaction protocols using graphical and logical methodologies, in: PROMAS, LNCS, vol. 3067, Springer, 2003, pp. 149–168.
[40] J. Pavón, J. Gómez-Sanz, R. Fuentes, Agent-oriented methodologies, Chapter the INGENIAS Methodology and Tools, Idea Group Publishing, 2005, pp. 236–276.
[41] S.P. Peisert, A model of forensic analysis using goal-oriented logging, Ph.D. Thesis, La Jolla, CA, USA, Adviser-Sidney Karin, 2007.
[42] W. Peng, W. Krueger, A. Grushin, P. Carlos, V. Manikonda, M. Santos, Graph-based methods for the analysis of large-scale multiagent systems, in: AAMAS '09: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2009, pp. 545–552.
[43] D. Poutakidis, L. Padgham, M. Winikoff, Debugging multi-agent systems using design artifacts: the case of interaction protocols, in: AAMAS '02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, New York, NY, USA, 2002, pp. 960–967. URL <http://doi.acm.org/10.1145/544862.544966>.
[44] D. Poutakidis, L. Padgham, M. Winikoff, An exploration of bugs and debugging in multi-agent systems, in: AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, New York, NY, USA, 2003, pp. 1100–1101. URL <http://doi.acm.org/10.1145/860575.860815>.
[45] A. Quirin, O. Cordón, V.P. Guerrero-Bote, B. Vargas-Quesada, F. Moya-Anegón, A quick MST-based algorithm to obtain pathfinder networks, Journal of the American Society for Information Science and Technology 59 (12) (2008) 1912–1924.
[46] A. Quirin, O. Cordón, J. Santamaría, B. Vargas-Quesada, F. Moya-Anegón, A new variant of the pathfinder algorithm to generate large visual science maps in cubic time, Information Processing and Management 44 (4) (2008) 1611–1623.
[47] K.H. Rosen, Discrete Mathematics and Its Applications, McGraw-Hill Higher Education, 1998.
[48] M. Schut, On model design for simulation of collective intelligence, Information Sciences, 2009. URL <http://dx.doi.org/10.1016/j.ins.2009.08.006>.
[49] R.W. Schvaneveldt, F.T. Durso, D.W. Dearholt, Network structures in proximity data, The Psychology of Learning and Motivation: Advances in Research and Theory 5 (24) (1989) 249–284.
[50] J. Scott, Social Network Analysis: A Handbook, second ed., Sage, Newberry Park, 2000.
[51] E. Serrano, J. Botia, Infrastructure for forensic analysis of multi-agent systems, in: Programming Multi-Agent Systems: Sixth International Workshop, PROMAS 2008 Estoril, Portugal, May 12–13, 2008, Revised Selected Papers. Springer-Verlag, Berlin, Heidelberg, 2009, pp. 168–183. URL <http://dx.doi.org/10.1007/978-3-642-03278-3_11>.
[52] E. Serrano, J.J. Gómez-Sanz, J.A. Botia, J. Pavón, Intelligent data analysis applied to debug complex software systems, Neurocomputing 72 (13–15) (2009) 2785–2795. URL <http://dx.doi.org/10.1016/j.neucom.2008.10.025>.
[53] S. Shope, J. DeJoode, N. Cooke, H. Pedersen, Using pathfinder to generate communication networks in a cognitive task analysis, in: Proceedings of the Human Factors and Ergonomics Society, 48th Annual Meeting, 2004, pp. 678–682.
[54] H.A. Simon, The architecture of complexity, Proceedings of the American Philosophical Society 106 (6) (1962) 467–482. URL <http://dx.doi.org/10.2307/985254>.
[55] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J.H.K. Levitt, C. Wee, R. Yip, D. Zerkle, The design of grids: a graph-based intrusion detection system, Technical Report, Department of Computer Science, University of California at Davis, 1999.
[56] W.-T. Tsai, X. Zhou, Y. Chen, X. Bai, On testing and evaluating service-oriented software, Computer 41 (8) (2008) 40–46. URL <http://doi.ieeecomputersociety.org/10.1109/MC.2008.304>.
[57] B. Vargas-Quesada, F. de Moya-Anegón, Visualizing the Structure of Science, Springer, New York, 2007.
[58] G. Vigueras, J.A. Botia, Tracking causality by visualization of multi-agent interactions using causality graphs, in: M. Dastani, A.E. Fallah-Seghrouchni, A. Ricci, M. Winikoff, PROMAS, Lecture Notes in Computer Science, vol. 4908, Springer, 2007, pp. 190–204.
[59] G. Vigueras, J.J. Gómez, J.A. Botia, J. Pavón, Using semantic causality graphs to validate MAS models, in: Hybrid Artificial Intelligence Systems Workshop (HAIS 07), Salamanca (Spain), 2007.
[60] D. Zernick, M. Snir, D. Malki, Using visualization tools to understand concurrency, IEEE Software 9 (3) (1992) 87–92. URL <http://dx.doi.org/10.1109/52.136185>.
[61] W. Zhu, Relationship between generalized rough sets based on binary relation and covering, Information Sciences 179 (3) (2009) 210–225. URL <http://dx.doi.org/10.1016/j.ins.2008.09.015>.